Open University Of Cyprus

School of Pure and Applied Sciences

Postgraduate Thesis
In Computer and Network Security

An Advanced Adaptive Learning Intrusion Prevention System

Christos Constantinides

Supervising Professor
Stavros Shiaeles

December 2018

Open University Of Cyprus

School of Pure and Applied Sciences

An Advanced Adaptive Learning Intrusion Prevention System

Christos Constantinides

Supervising Professor
Stavros Shiaeles

A thesis submitted in partial fulfillment for the postgraduate degree in

MSc Computer and Network Security

to the School of Pure and Applied Sciences
Open University Of Cyprus

December 2018

ABSTRACT

Computer and network attackers are continuously evolving their attack vectors to evade intrusion detection systems. Commercial and real-world intrusion detection prevention systems suffer with low detection rates and high false positives which require substantial optimization and network specific fine tuning. Furthermore, the majority of those systems rely on signatures to detect potential attacks and therefore unknown attacks to the public - "zero day attacks", are by definition, undetectable by such systems.

Intrusion Detection Prevention Systems fail to satisfy the organizations security requirements in detecting newly published attacks or variants of existing attacks, effectively responding to attacks launched by sophisticated attackers and resisting attacks that are intended to circumvent them. This is the result of Intrusion Detection Prevention Systems lack of adaptation to new information.

Introducing "intelligence" to Intrusion Detection Prevention Systems could solve the problems mentioned above.

This thesis propose a novel Network Intrusion Prevention System that utilizes Self Organizing Incremental Neural Networks along with SVMs, not relying on signatures or rules and capable to mitigate known and unknown attacks on a high accurate level in an "online" and incremental manner.

Based on the experimental results with NSL KDD dataset the proposed framework can achieve on-line updated incremental learning, suitable for efficient and scaling industrial applications with high accuracy results.

Περίληψη

Οι επιτιθέμενοι υπολογιστών και δικτύων εξελίσσουν συνεχώς τις επιθέσεις τους για να αποφύγουν τα συστήματα ανίχνευσης εισβολών. Τα εμπορικά συστήματα πρόληψης ανίχνευσης εισβολών που χρησιμοποιούνται από οργανισμούς υποφέρουν από χαμηλά ποσοστά ανίχνευσης και υψηλά ποσοστά εσφαλμένων θετικών και απαιτούν σημαντική και ουσιαστική βελτιστοποίηση και συνεχή ρύθμιση από τον διαχειριστή του δικτύου.

Επιπλέον, τα περισσότερα από αυτά τα συστήματα βασίζονται σε ηλεκτρονικές υπογραφές για την ανίχνευση πιθανών επιθέσεων και ως εκ τούτου άγνωστες μη δημοσιευμένες επιθέσεις είναι εξ ορισμού μη ανιχνεύσιμες από τα συστήματα αυτά.

Τα συστήματα πρόληψης ανίχνευσης εισβολών δεν ικανοποιούν τις απαιτήσεις ασφάλειας των οργανισμών για την ανίχνευση των πρόσφατα δημοσιευμένων επιθέσεων ή παραλλαγών των γνωστών επιθέσεων, εξελιγμένων επιθέσεων ή/και επιθέσεων που αποσκοπούν στην παράκαμψη των συστημάτων αυτών. Αυτό είναι το αποτέλεσμα της έλλειψης προσαρμογής νέων δεδομένων των συστημάτων αυτών.

Μετατρέποντας τα συστήματα πρόληψης και ανίχνευσης σε "έξυπνα", θα μπορούσε να δώσει λύσεις στα προβλήματα που αναφέρθηκαν πιο πάνω.

Αυτή η μεταπτυχιακή διατριβή προτείνει ένα νέο εξελιγμένο "έξυπνο" Σύστημα Πρόληψης Ανίχνευσης Εισβολών Δικτύου το οποίο χρησιμοποιεί αλγόριθμους Self-organizing Incremental Neural Networks και Support Vector Machines και δεν βασίζεται σε ηλεκτρονικές υπογραφές ή προγραμματιστικούς κανόνες και είναι σε θέση να ανιχνεύσει γνωστές και άγνωστες επιθέσεις σε υψηλά επίπεδα ακρίβειας με την συνεχή αναβάθμισης της νοημοσύνης του.

Τα πειραματικά αποτελέσματα έδειξαν ότι η προτεινόμενη λύση θα μπορούσε μετεξελιχθεί σε εμπορική εφαρμογή που θα έχει τη δυνατότητα να προστατεύει τα δίκτυα υπολογιστών με τρόπο που να ικανοποιεί τις απαιτήσεις ασφάλειας των οργανισμών.

Dedication

I would like to dedicate this thesis to my family and my partner in life. Without their support this thesis would not be possible. I would also like to thank my supervising professor for his guidance and his valuable insights.

Table of Contents

Chapter 1
Introduction

The recent attacks in important and prestigious multinational organizations which made headlines in the news media globally are examples of how destructive a computer and network security breach can be to an organization's reputation and financial stability. Moreover, the severity and frequency of computer and network attacks has increased significantly in the past two decades and the progressive growth of devices connected to networks and the internet, resulted in the progressive growth of security incidents and events with a cost on organizations assets and a negative impact on peoples lives. Network security is one of the most important elements towards securing interconnected computerized systems and Intrusion Detection Systems are on the first line of defense and vital for securing networks and computers.

Commercial and real-world intrusion detection prevention systems suffer with low detection rates and high false positives which require substantial optimization and network specific fine tuning. Furthermore, the majority of those systems rely on signatures to detect potential attacks and therefore unknown attacks to the public - "zero day attacks" are by definition undetectable by such systems. Intrusion Detection Prevention Systems fail to satisfy an organization's security requirements in detecting newly published attacks or variants of existing attacks, effectively responding to attacks launched by sophisticated attackers and resisting attacks that are intended to circumvent them and thus failed to adapt to new information.

Crucial to the security of an organization is the ability to identify and prevent attacks on its computer and network infrastructure and currently commercial and industrial Intrusion Detection Prevention Systems fail to do so for newly published attacks or variants of existing attacks. Additionally, the luck of adaptation to dynamically adjust to new data makes them unsuitable for the ever-changing nature of

internet and organizations networks.

Despite the significant academic work by researches to introduce intelligence to NIDPS by implementing anomaly based machine learning detection methods, there has been little real-world adoption by such systems.

A machine learning based NIDPS could offer solutions to the problems signature and rule based industrial NIDPS suffer. Additionally to the lack of real-world applications based on machine learning, there has been very limited academic work focusing on incremental learning algorithms applied to NIDPS.

Incremental learning algorithms allows the machine learning algorithm to refine and improve its capabilities over time (input data) in contrast to an offline or batch learning algorithm where the classifier is assumed to be exposed to the input data once - in a batch. Network data dynamically change over time and applying static learned models degrades the detection performance significantly over time, making an offline algorithm not suitable for a modern NIDPS.

This postgraduate thesis contribution to the literature and to the computer and network security industry is to tackle those issues with the use of artificial intelligence and an online incremental machine learning framework.

This paper proposes a novel Network Intrusion Prevention System based on an incremental machine learning framework where it achieves high accurate results comparable to most of the offline neural network-based NIDS. The framework is based on a modified version of SOINN - Self-Organizing Incremental Neural Network (Shen and Hasegawa 2006 : 92) to achieve on-line clustering and multiple SVMs to perform classification. Experimentation and evaluation was performed with the NSL-KDD dataset (Dhanabal and Shantharajah 2015 : 446), which is an improved version of well-known KDD'99 dataset. The results shows that the proposed framework can achieve on-line updated incremental learning in a fast and efficient manner. The rest of the paper is organized as follows:

Chapter 2: Literature Review, in this chapter a comprehensive and critical literature review of the research work done in methodologies and techniques applied to intrusion detection, with an emphasis on anomaly based detection intrusion detection systems will be presented. The review will show that there has been limited research work introducing intelligence, implementing anomaly based machine learning detection methods in an incremental and online manner specifically to network intrusion detection prevention systems.

Chapter 3: Background information, this chapter provides the necessary background information to the

reader to understand the technologies used through out this thesis. It begins with an introduction to intrusion detection in computer and network security and then introduces intrusion detection systems and the problems these systems suffer in industrial and commercial applications. This chapter also introduces the reader to the concept of artificial intelligence and the methods machine learning algorithms use, as a way of feedback, namely: supervised learning, unsupervised learning, semi-supervised learning, active learning, reinforcement learning . A section is dedicated to the concept of incremental learning and online learning which this thesis is focusing on. Furthermore artificial neuron networks and support vector machine, technologies that are also used in the proposed framework are discussed. The final section of this chapter presents the dataset used for the experimental and evaluation process.

Chapter 4: Proposed framework, this chapter presents the proposed framework and its modules. It elaborates on each module function and provides proof-of-concept with the code available in Appendix A. The detection engine, the preprocessing module of the incoming traffic that feeds the detection engine, the validation module that evaluates the results of the detection engine and the update module where the failed predictions are fed back to the detection engine, are described in detail.

Chapter 5: Experimental results, this chapter discuss the experimental process and provides the different values used through out the evaluation of the framework. The accuracy and time cost results are presented in tables along with graphs and a comparison between offline and online results is given.

Chapter 6 : Discussion and future work, this chapter discusses the experimental evaluation results and the strengths and limitations the framework posses. The framework achieved incremental learning with promising results and a comparison between the framework and other offline algorithm is also given. Future contributions to the framework like an additional module which will detect unknown attacks in automated way and incrementally are also discussed.

Chapter 7 : Conclusion, this chapter resumes this postgraduate thesis and provides an insight into the contribution the framework delivers to the literature and the field of computers and network security. The promising accuracy results, along with its life-long learning and scaling qualities makes the proposed framework a promising prospect towards further development for a future industrial application.

Chapter 2
Literature Review

The aim of this literature review is to critically evaluate prior research in the areas of intrusion detection, mainly on the network level, which could be applied in a network intrusion detection and prevention systems. It will demonstrate where research work failed to provide answers to real world problems in computer and networks security and more specifically intrusion detection and how this postgraduate thesis could contribute to the academia and the research community.

Intrusion detection and prevention system implementations failed to provide organizations with a comprehensive protection against sophisticated and targeted attacks. NIDPS suffer with low detection rates and high false positives. Furthermore, the majority of those systems rely on signatures to detect potential attacks and therefore unknown attacks - "zero day attacks" or variants of existing attacks pass through undetected.

There has been limited research work introducing intelligence and implementing anomaly based machine learning detection methods in online and incremental manner to NIDPS. As such there has been little real-world adoption of intelligent network intrusion detection and prevention systems.

An intelligent machine learning based NIDPS could offer solutions to the problems signature and rule based industrial intelligent network intrusion detection and prevention systems suffer. Additionally to the lack of real-world machine learning applications, there has been very limited academic and research work focusing on incremental learning algorithms applied to intelligent network intrusion detection and prevention systems.

This postgraduate thesis will attempt to contribute a solution to the problems mentioned above with the

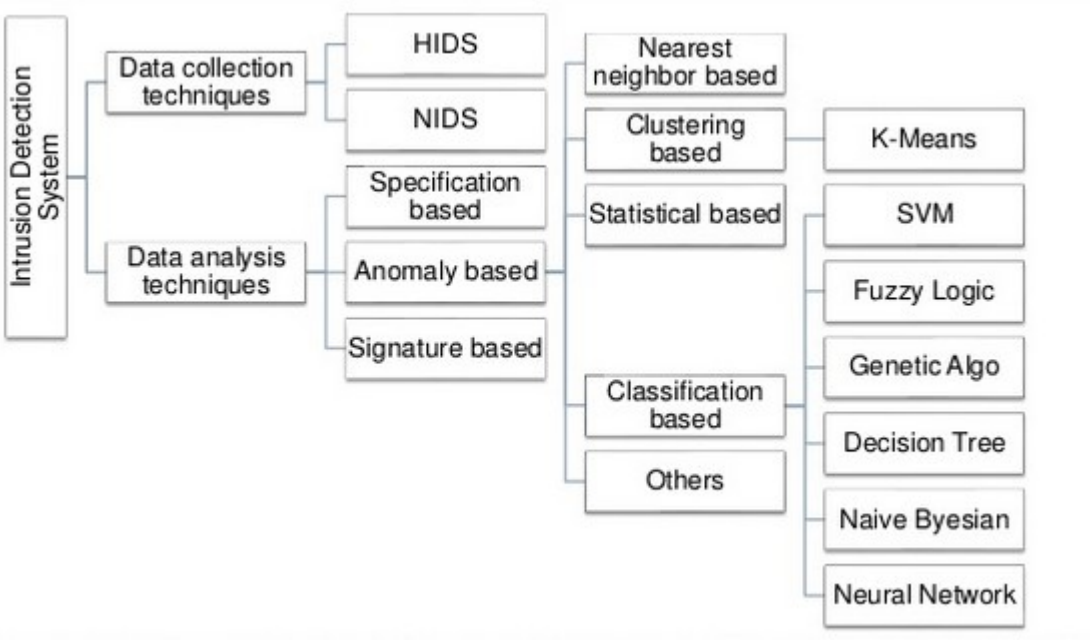
use of artificial intelligence and an online incremental machine learning framework to the literature and computer and network security industry.

2.1 Detection methods

Intrusion detection systems, can be categorized by the way of the detection method used, the two broad categories classifying these detection methods are signature - misuse; based detection and anomaly based detection (Liao et al. 2013 : 17)

The signature based detection methodology is using patterns previously defined by known attacks and is distributed in the form of signatures. The signatures are then compared against patterns found in the network traffic to discover possible attacks or threads. The disadvantages of this method are the luck of capability to discover and prevent unknown attacks and the signatures maintenance and updates for the known and newly discovered attacks could pose a thread if not updated frequently and could become a system's liability.

Figure 1: Intrusion Detection System Techniques



The anomaly based detection methodology is using the concept of discovering anomalies - unusual patterns in the network traffic compared to normal traffic defined by the system. The system is trained before by feeding it with data in order to be able to make the distinction of what is normal and what is an attack. The distinction is found using different data mining techniques which can be divided in to two broad categories, classification and clustering. There has been substantial work and research in classification techniques for Intrusion Detection Systems, among others are : neural networks (Debar et al. 1992 : 240 - 250, Ibrahim 2010 : 457-471, Moradi and Zulkernine 2004 15-18, Ryan and

Miikkulainen 1998 : 943-949, Zhang 2001 : 85 -90), fuzzy logic and genetic algorithms (Bridges and Vaughn 2000 : 109-122) and support vector machines (Li et al. 2012 : 424-430). Work in clustering techniques include k-means and outlier detection algorithm (Bhuyan et al. 2016 243–271).

Combinations of both to these technique categories and to both the detection methods have been used to form a third category called hybrid. Research work in this category (Depren et al. 2005 : 713-722, Li et al. 2010 : 169-172, Kim et al. 2014 : 1690-1700) combined anomaly and misuse detection to propose novel solutions for an intrusion detection system. Similar to the method of extending a neural based technique with the SVMs (Mukkamala et al. 2002 : 1702-1707) to propose a solution for an IDS that could recognize anomalies and known intrusions, this postgraduate thesis used artificial neuron networks and support vector machines to achieve its objective.

A lot of research has been contacted on anomaly based techniques in Intrusion Detection Systems, especially the last few years, and the accuracy rates has been improved during the past decade, however very few has high enough accuracy rates, low enough false positive rates, time and resource efficient to be considered candidates for development for real world scalable applications.

2.2 Artificial Intelligence

Rich and Knight's defined Artificial Intelligence as the study of how to make computers do things at which, at the moment, people are better (Rich and Knight, 1991 : 3). An important test to pass for *"computer intelligence"* is the Turing test, which defines the inability to distinguish computer responses from human responses. According to Stuart and Norvig the capabilities needed to pass the Turing test are natural language processing, knowledge representation, automated reasoning and machine learning to adapt to new circumstances and to detect and extrapolate patterns. (Stuart and Norvig 2010 : 2).

2.2.1 Machine Learning

Machine learning refers to the ability an agent possess to progressively improve its performance on a specific task. The methods used to accomplish the task of learning by example in machine learning as a way of feedback, are classified in to five broad categories, supervised learning, unsupervised learning, semi-supervised learning, active learning, reinforcement learning . The methods used to accomplish the task of learning by example in machine learning as a way of feedback, are classified in to five broad categories, supervised learning, unsupervised learning, semi-supervised learning, active learning, reinforcement learning . In his work Zhu resulted that the accuracy of the output is improved - if used for classification, compared to the unsupervised method and the time cost is reduced compared to the supervised method (Zhu 2006 : 9). This thesis used a combination of unsupervised and supervised

method, namely, artificial neuron networks - SOINN, which utilizes the unsupervised method to achieve clustering and incremental learning and Support Vector Machines - SVMs which utilizes the supervised method for classification. Both of the methods used above to achieve incremental learning applied to intrusion detection, an essential and desired method along with the online learning technique that could realize scalable intrusion detection applications. There is very limited research work and no industrial or commercial applications according to this thesis author knowledge utilizing incremental learning applied to intrusion detection. For example Yu and Lee proposed a supervised incremental learning method for use in anomaly detection by cascading service classifier and ITI Decision Tree Methods and their results based on the KDD-99 dataset were 92.63% detection accuracy and a 1.8% false positives (Yu and Lee 2009 : 155-160). Although their results show high accuracy, they used the KDD-99 dataset with the inherit problems discussed in Chapter 3 of this thesis and by Mahbod, et al in their work (Mahbod, et al. 2009 : 1-6). Substantial research has been made in detecting specifically Dos attacks in network traffic as well. In their paper, Robinson and Thomas (Robinson and Thomas, 2015 : 185) used three different datasets to rank the success of machine learning algorithms to detect DoS attack streams. Their findings indicate that the ensemble algorithm of Adaboost with Random Forest as the base classifier were the best method for detection, with an average of 99.77 over the three datasets. They used feature extraction on the packet level and their results show that this approach works well in distinguishing attack traffic from normal traffic. Choudhury and Bhowal (Choudhury and Bhowal, 2015 : 185) and Panda and Patra (Panda and Patra , 2009 : 472), confirmed Robinson and Thomas study of the effectiveness of Random Forests in the categorization of traffic. However, both studies suffer from a small number of training and test data.

2.2.2 Artificial Neural Networks

Most of the work and research in the past few years in artificial intelligence and more specifically machine learning applied to Intrusion Detection Systems, is focused in Artificial Neural Networks. Starting with Debar et al., the authors proposed a user behavior model where each user is represented by a neural network, the neural network component of the proposed model was just a complement of a statistical model (Debar et al 1992 : 240-250) . Ryan et al. used the back propagation algorithm to train the neural network for an IDS proposal with a 96% detection accuracy and a 7% false alarm (Ryan et al 1998 : 943-949). "HIDE" a work from Zhang 2001, proposed a Hierarchical Network Intrusion Detection system using statistical preprocessing and neural network classification, in their paper they tested five different neural networks that showed the back propagation algorithm to outperformed the

rest (Zhang 2001 : 85-90).

In their work Moradi and Zulkernine, showed that neural networks with the use of the Multi Layer Perceptron in an offline analysis, could be applied in intrusion detection and solve a multi class problem in where not only the attack could be recognized but also classified based on the type of an attack(Moradi and Zulkernine 2004 : 15-18). The paper's accuracy results was 91% with two hidden layers. Ibrahim, used a distributed time delay neural network (Ibrahim LM 2010 : 457-471) to solve a similar a multi class problem to the work by Moradi and Zulkernine, but with improved accuracy results of 97.24%.

In their paper Bouzida and Cuppens (Bouzida and Cuppens, 2006 : 28) showed that neural networks and decision tress can work together in attack classification. They found that neural networks were able to classify attacks in network traffic and decision trees could identify unknown attacks.

A recent study based on the same dataset as Bouzida and Cuppens, Kim, et al, utilized deep neural networks for intrusion detection and attack classification (Kim, et al., 2017 : 313). They showed that by using a deep neural network with four hidden layers and testing against the KDD Cup 99 dataset, they achieved a detection rate of 99.19. Yet again, another study by Tammi, et al. with the same dataset but using a combination of K-means clustering and neural networks, specifically

the Probabilistic Neural Network (Tammi, et al., 2015 : 21) which achieved 97.89% accuracy. Again just like in Chapter 2, the three papers above used the KDD-99 dataset with the inherit problems discussed in Chapter 3 of this thesis and by Mahbod, et al in their work (Mahbod, et al. 2009 : 1-6).

A paper by Xiang et al. used a modification of SOINN (Shen et.al 2006 : 90-106), a self-organizing incremental neural network, to apply it to intrusion detection. With a semi-supervised learning based IDS proposal they showed that the user input could be minimized by combining a modified SOINN and SVM to achieve semi-supervised learning with the same space efficiency to a supervised SVM (Xiang et al. 2016 : 815-823).

2.2.3 SOINN - Self-Organizing Incremental Neural Network

In their original paper Shen et.al presented an on-line unsupervised neural network for unlabeled data- SOINN, realizing incremental learning - a topological representation network, which represents the topological structure of the input data. The breakthrough SOINN introduced in relation to previous research and work, in machine learning and neural networks specifically, is the ability to learn incrementally in an online set up with out the need to predefine - and therefore predict, the size and the structure of the network(Shen et.al 2006 : 90-106).

In a following paper Shen et.al the authors proposed an enhanced and more stable version of the original SOINN called ESOINN which minimized the user input and adopted a single layer in contrast to the two layers of original SOINN. Both the original and the enhanced version of the SOINN the online incremental learning was fed by unsupervised data, meaning that the data was not labeled before classification (Shen et.al 2007 : 893-903).

In their paper Shen et.al proposed an online, incremental active learning algorithm, based on a self-organizing incremental neural network fed by semi-supervised data. They extended the original SOINN which by definition process unlabeled data to process label data. The previous work mentioned above on SOINN, was applied to solve problems in optical pattern recognition (Shen et.al 2011 : 1061-1074).

Xiang et al 2016 extended SOINN similar to ESOINN to present a semi-supervised intrusion detection framework with a Support Vector Machine classifier. The semi-supervised extension to the single SOINN - ESOINN, called MSOINN - mixture SOINN is used to process large volume of labeled and unlabeled data incrementally. From that MSOINN a kernel function is constructed to train the classifier (SVM) with the labeled and unlabeled data. Their experiments showed 84.5 % detection rate and 7.32% false positives (Xiang et al. 2016 : 815-823).

2.4 Summary

From the literature review we can conclude that there is a gap in on-line learning or life-long learning tasks applied to Intrusion Detection Prevention Systems, offline analysis is the norm which is not suitable for the ever-changing nature of internet and organizations networks. Additionally, the offline analysis method does not adapt to dynamic data and therefore unsuitable for the detection of "zero day" attacks - newly published attacks or variants of existing attacks.

This paper objective is trying to address this gap by the of use online incremental machine learning by preserving previously gained knowledge in order overcome the issues from using static - offline machine learning models applied to Network Intrusion Detection Prevention Systems designs. Furthermore, the proposed framework lays the ground work for efficient detection of "zero day" attacks and variants of existing attacks. This thesis, makes use of the NSL-KDD dataset, an updated and refined version of the KDD99 dataset (Mahbod, et al. 2009 : 1-6), which is widely used, publicly available dataset, for network-based anomaly detection systems.

Chapter 3
Background Information

In order to better understand the purpose of this thesis, a background information will be presented to the reader to familiarize with definitions, terms and technologies used to build the framework and specifically its detection engine - n-SOINN-WTA-SVM. The chapter provides an introduction in section 3.1 to intrusion detection in computer and network security and intrusion detection systems then introduces and the problems these systems suffer in industrial and commercial applications. In section 3.1.1 intrusion detection prevention systems are presented and the distinction from intrusion detection systems is given. In section 3.2 introduces the reader to the concept of artificial intelligence and in section 3.21 the methods machine learning algorithms use, as a way of feedback, namely: supervised learning, unsupervised learning, semi-supervised learning, active learning, reinforcement learning. In section 3.2.2 is dedicated to the concept of incremental learning and online learning which this thesis is focusing on. Sections 3.3 and 3.4 present artificial neuron networks and support vector machines, technologies that this framework is based on. Finally Section 3.5 provides details about the dataset used in the experimental and evaluation process.

3.1 Intrusion Detection

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network (Bace and Mell 2001 : 5). Intrusion detection technology is the technology designed to monitor computer activities for the purpose of finding security violations. It is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations of threats, computer security policies, acceptable use policies, or standard security practices (Crothers

2003 : 1).

Intrusion prevention works similar to that of an intrusion detection system but it also attempts to stop and prevent detected security incidents. Intrusion detection prevention systems primarily focus on identifying and block possible incidents, keep track of incidents in the logging system and create alerts by reporting to security administrators. In addition, organizations use Intrusion Detection Systems for other purposes, such as identifying problems with security policies, documenting existing threats, and deterring individuals from violating security policies (Bace and Mell 2001 : 23). NIDPS have become a necessary addition to the security infrastructure of nearly every organization.

Intrusion Detection Systems provide a layer of protection to an organization's computer and network security by constantly monitoring and analyzing system events and user behaviors, testing the security state of the organization's technological system resources, track system configuration changes, identify attacks based on known patterns, identify attacks based on statistical deviations and managing system's audit and logging mechanisms. Intrusion Detection Prevention Systems fail to satisfy an organization security requirements in detecting newly published attacks or variants of existing attacks, effectively responding to attacks launched by sophisticated attackers, automatically investigating attacks without human intervention and resisting attacks that are intended to defeat or circumvent them - luck of adaptation (Bace and Mell 2001 : 38).

Crucial to the security of an organization is the ability to identify and prevent attacks on its computer and network infrastructure and currently commercial and industrial Intrusion Detection Prevention Systems fail to do so for "zero day" attacks - newly published attacks or variants of existing attacks. Additionally, the luck of adaptation to dynamically adjust to new data makes them unsuitable for the ever-changing nature of internet and organizations networks. This postgraduate thesis contribution to the literature and computer and network security industry attempts to tackle those issues with the use of artificial intelligence and machine learning algorithms and techniques.

3.1.1 Intrusion Detection Prevention Systems

Intrusion detection systems monitor computer and network systems for potential malicious activity. Intrusion prevention systems prevent a malicious attack after it is identified as well. Network intrusion detection systems monitor network traffic to identify malicious activity and protect hosts connected to that network, in contrast to host intrusion detection systems, that monitor an individual's hosts processes and network activity. A network intrusion prevention system - NIPS, does not only passively logs and alerts like a network intrusion detection system - NIDS does, but it also actively protects the

network by dropping network packets that are identified as attacks. An example of network intrusion detection prevention system can be seen in **Fig.2**, an attacker bypass the firewall and the NIDPS detects the attempt on the switch level right before it reaches the host victim, then a mechanism in combination with the firewall drops the connection.

A network intrusion detection prevention system operates and analyzes network traffic on the transport layer of the open system interconnection (OSI) and TCP/IP model. If a network segment or a collection of these, signature matches to a known collection/library of attacks then, it either logs it and alerts the network administrator or either drops it. Today's NIDPS combine signature based methods and anomaly based methods to identify an attack. The signature based detection methodology is using patterns previously defined by known attacks and is distributed in the form of signatures. The signatures are then compared against patterns found in the network traffic to discover possible attacks or threads.

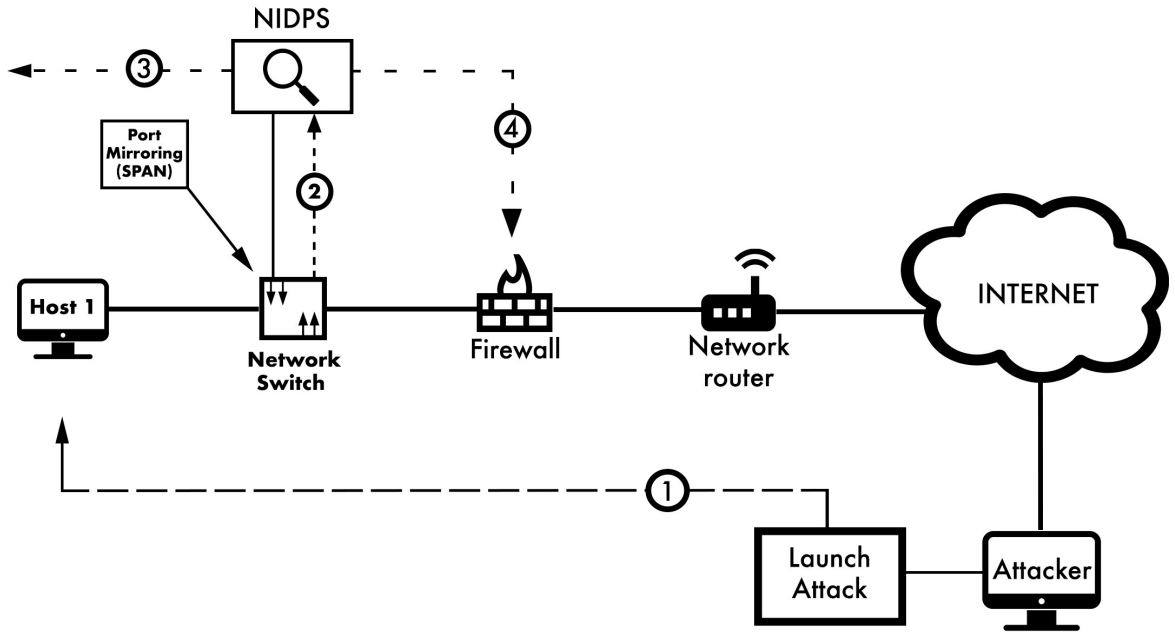


Figure 2: Network Intrusion Detection Prevention System

The anomaly based detection methodology is using the concept of discovering anomalies - unusual patterns in the network traffic compared to normal traffic defined by the system. One of the main disadvantage of industrial NIDPS is the fact that they are prone to high numbers of false alarms - false

positives. Also, an organization need to fine-tune and configure its NIDPS when it is initially installed in order to reduce the number of false positives. To properly configure the NIDPS, in order to be able to distinguish normal traffic compared to malicious traffic the system relies heavily on the network administrator which makes it prone to human errors. Therefore, NIDPS performance is directly related to the network administrator abilities, time and resources in order to be efficient and operational.

3.2 Artificial Intelligence

Following Rich and Knight's definition in which machines act as humans, they defined Artificial Intelligence as the study of how to make computers do things at which, at the moment, people are better (Rich and Knight, 1991 : 3). One of the tests to pass for a computer or a machine to be named "*intelligent*" is the Turing test, which defines the inability to distinguish computer responses from human responses. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.

To pass the Turing test and other tests a computer would need to possess the following capabilities: natural language processing to enable it to successfully translate in a language humans communicate; knowledge representation to store what it knows or hears; automated reasoning to use the stored information to answer questions and to draw new conclusions; and machine learning to adapt to new circumstances and to detect and extrapolate patterns. (Stuart and Norvig 2010 : 2). This postgraduate thesis used artificial neuron networks and support vector machines to achieve its objective, by detecting and extrapolate patterns from computer network traffic, all of which fall under the field of machine learning.

3.2.1 Machine Learning

A machine or an agent learns if it posses the ability to progressively improve its performance on a specific task, over observed data - by example. The methods used to accomplish the task of learning by example in machine learning as a way of feedback, are classified in to five broad categories, supervised learning, unsupervised learning, semi-supervised learning, active learning, reinforcement learning .

The function in supervised method learns by mapping input data with output data as the expected outcome. The supervised method is usually used in classification with the output data represented as labels that are desired for a given input.

In the unsupervised method the function learns from patterns only found in the input, with no output to learn from. This method is usually used in clustering where information from input is represented as

clusters in a compressed form, organizing what seems to be chaotic unlabeled data in a meaningful way. This postgraduate thesis used artificial neuron networks - SOINN, which utilizes the unsupervised method to achieve clustering and incremental learning and Support Vector Machines - SVMs which utilizes the supervised method for classification.

The semi-supervised method uses labeled and unlabeled data for training. It is used as hybrid method between supervised and unsupervised methodologies by introducing a relatively small amount of labeled data compared to the unlabeled data. It is stated (Zhu 2006 : 9) that the accuracy of the output is improved - if used for classification, compared to the unsupervised method and the time cost is reduced compared to the supervised method.

Active learning utilizes supervised, unsupervised and semi-supervised methods to create hybrid systems that aims to train an agent with less data without sacrificing accuracy. Active learning theory introduces an external - human or not, oracle that labels the output interactively according to the systems requirements.

Reinforcement Learning is based on the idea of *The Reward Hypothesis*, which states that all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (Sutton 1998). The agent's goal is to get as much reward as it can over the long run just like humans learn from interacting with the environment, it comes from our natural experiences. Reinforcement learning methods specify how the agent changes its policy as a result of experience with the aim to maximize its objective.

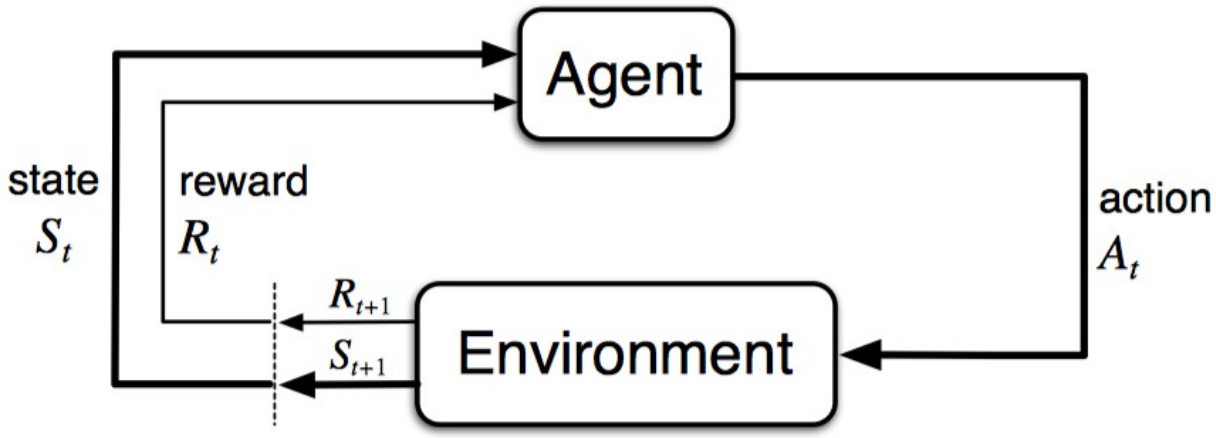


Figure 3: Reinforcement learning depiction

The system description consists of an agent which interacts with the environment via its actions at

discrete time steps and receives a reward which in turn transitions the agent into a new state.

3.2.2 Incremental Learning

Incremental learning is a method, in which input data is continuously fed to an agent with limited resources to extend its knowledge and without sacrificing the models accuracy. It represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training - streaming data becomes available gradually over time. Algorithms that can facilitate incremental learning are known as incremental machine learning algorithms. The aim of incremental learning is for the learning model to adapt to, and repeatedly learn from new data without forgetting its existing knowledge. Incremental learning applications are best fitted in scenarios with dynamically changing environments where sequential acquisition of knowledge is possible. Computer networks are characterized by a dynamic nature with abundance of new data flowing over time. Incremental learning continuous model adaptation is based on a constant stream of data (Alexander and Hammer 2016 : 1) and that property makes it a perfect candidate method for network intrusion prevention systems, where new computer and network attacks are emerging constantly. The challenge of incremental learning is to acquire new information without destroying or corrupting previously learned knowledge, the so called Stability Plasticity Dilemma (Carpenter and Grossberg 1988 : 77-88).

Online machine learning refers to the method where the agent is trained when data becomes available in a sequential order, as opposed to batch learning where the agent is trained on the entire training data set, once, and hold the assumption of complete data availability. Online learning is a common technique where the algorithm is required to dynamically adapt to new patterns in the data, in which new data arrive over time. Furthermore, online learning becomes necessary in interactive scenarios where training examples are provided based on human feedback over time [Kawewong et al. 2011].

Incremental learning make use of online learning strategies with limited memory resources available, in contrast to strategies that store all examples in memory. In order to achieve that incremental learning often utilizes a scheme where the observed input is represented in a compact form.

This postgraduate thesis proposal implements incremental learning with the use of the online learning technique by a scheme which utilizes SOINNs to compress the observed input into clusters and inherit online learning which the specific neural network posses in combination with SVMs for classification. The scheme also uses an external oracle that labels the output interactively, a characteristic of active and online learning.

3.3 Neural Networks

Artificial neural networks are computational learning systems modeled on the parallel biological neural networks architecture of animal brains. The network is based on a simple form of inputs and outputs, an interconnected network of simple processing units (artificial neurons) that learns from experience by modifying its connections (Gerven and Bohte 2018 : 12).

An Artificial neural network is a network of mathematical functions, *artificial neurons or nodes*, which is interconnected with *edges*. Artificial neurons and edges typically are arranged into layers and have a weight that adjusts as learning proceeds.

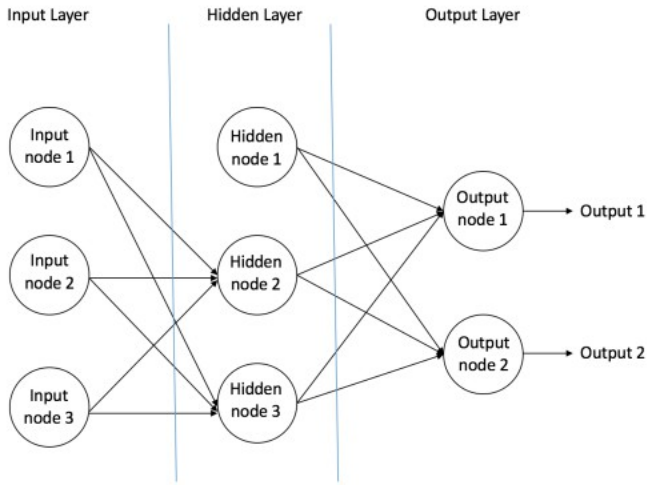


Figure 4: Feed-forward neural network arrangement

Different layers may perform different kinds of transformations on their inputs. A *neuron* takes an input *x* - usually a number and translates that - with the mathematical function into a desired output *y*. An example of a simple artificial neural network arrangement is the *feedforward neural network* which consists of multiple nodes arranged in layers. *Nodes* from adjacent layers connected with *edges* between them and have weights associated with them. The information moves in only one direction – forward – from the input nodes, through the hidden nodes and to the output nodes as seen in **Fig.4**.

The Input nodes provide information from the environment as an input to the network and are arranged as the *Input Layer*. No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes. The Hidden nodes have no direct connection with the environment and they perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a *Hidden Layer*. While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers. The Output nodes are collectively referred to as the *Output Layer* and are responsible for computations and transferring

information from the network to the environment.

Artificial networks *learn* to perform tasks by considering examples, generally without being programmed with any task-specific rules and recognize patterns - make sense of data association that the human is not possible to make. Those properties makes ANN attractive for many applications and with great success. Currently, most industrial and commercial Intrusion Detection Systems are based on rules, signatures and some statistical analysis to operate and ANN and machine learning in general will eventually replace those types of IDS with intelligent IDS which will eliminate any programming rules at their core in terms of detection.

3.3.1 SOINN

The self-organizing and incremental neural network introduced by (Shen and Hasegawa 2006: 90-106) is an online unsupervised incremental learning mechanism for unlabeled data, which represents the topological structure of the input data. As mentioned in paragraph 3.2.1, one objective of unsupervised learning is clustering, another objective in terms of applying unsupervised learning beyond classification is topology learning - the representation of the topology structure of a high-dimension data distribution.

Various models and mechanisms had been developed to address topology learning, all of which had their limitations compared to SOINN with the objective of achieving online incremental learning, starting with the self-organizing map - SOM by (Kohonen, 1982 :59–69), which generates mapping from high-dimensional signal space to lower-dimensional topological structure, is inherently limited because of its predefined structure and size (Shen and Hasegawa 2006: 91). The combination of competitive Hebbian learning - CHL and neural gas - NG by (Martinetz and Schulten 1994 : 7(3), 507–522) solved the problem of the network predefined structure but their work also requires a prior decision about the network size and finally (Fritzke, 1995) addressed the problem mentioned above with their work in growing neural gas - GNG but it had its disadvantage as well, the permanent increase in the number of nodes of the network and therefore the size of the network will grow exponentially.

In order to achieve online incremental learning in difficult problems of non-stationary data distributions, on-line learning or life-long learning tasks, the above-mentioned methods are not suitable (Shen and Hasegawa 2006: 91), the problem with applying the methods above for online learning is the acquisition of new information without destroying or corrupting previously learned knowledge, the so called Stability Plasticity Dilemma (Carpenter and Grossberg 1988 : 77-88) as mentioned in the paragraph 3.2.3 above. SOINN's objective is to develop a network that operates autonomously, on-line or life-long, and in a non-stationary environment. The network grows incrementally, learns the number of nodes

needed to solve a current task, learns the number of clusters, accommodates input patterns of on-line non-stationary data distribution, and dynamically eliminates noise in input data (Shen and Hasegawa 2006: 91).

SOINN's proposed method objectives dictates that for unsupervised on-line learning tasks, Shen and Hasegawa separate unlabeled non-stationary input data into different classes without prior knowledge such as how many classes exist. Similarly they address the same principal to learn input data topologies.

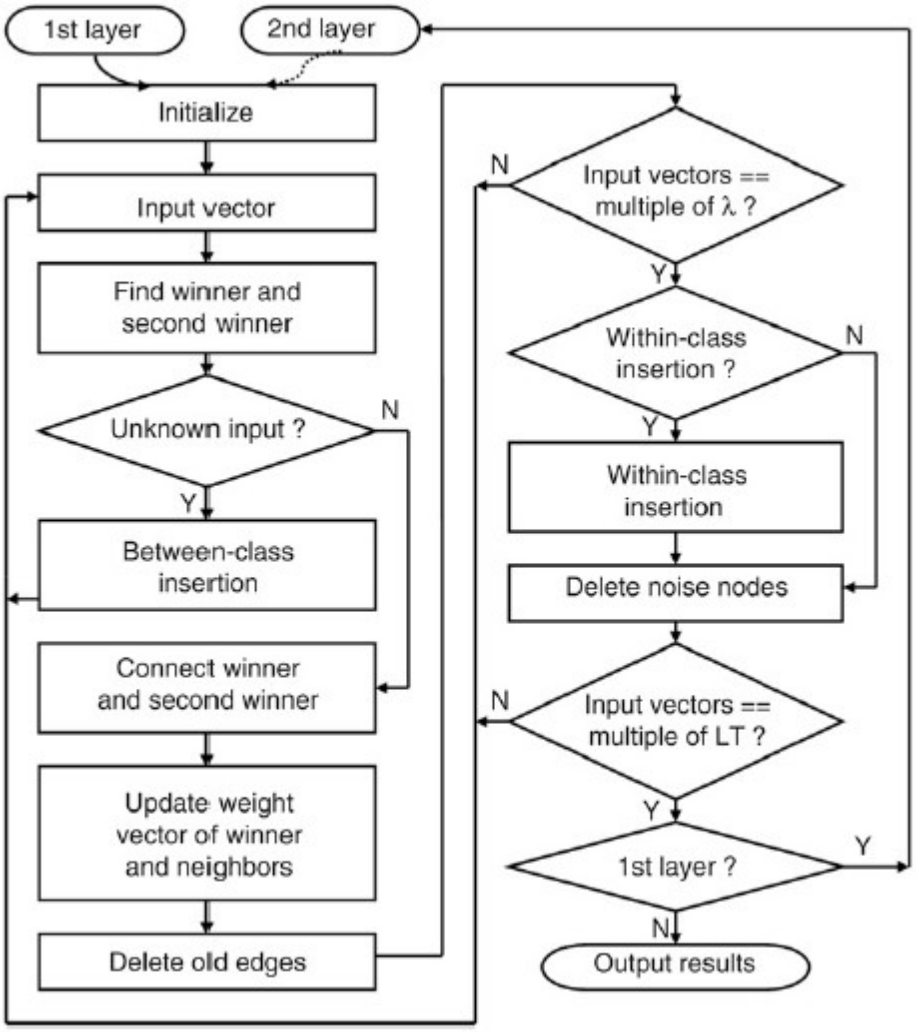


Figure 5: SOINN algorithm flowchart

Their proposed proposed algorithm achieved to process on-line or life-long learning non-stationary data, no prior conditions such as a suitable number of nodes or knowing how many classes exist, to conduct unsupervised learning is needed, report a suitable number of classes and to represent the

topological structure of input probability density. The algorithm also separate classes with low-density overlap's and detects the main structure of clusters that are polluted by noise - garbage collection (Shen and Hasegawa 2006: 92).In a following paper of the original work on SOINN (Shen et.al 2007 : 893-903) proposed an enhanced and more stable version of the original SOINN called E-SOINN which minimized the user input and adopted a single layer in contrast to the two layers of original SOINN. Both the original and the enhanced version of the SOINN the online incremental learning was fed by unsupervised data, meaning that the data was not labeled before classification.

The breakthrough SOINN introduced in relation to previous research and work, in machine learning and neural networks specifically, is the ability to learn incrementally in an online set up with out the need to predefine - and therefore predict, the size and the structure of the network.

SOINN's properties of online incremental learning and representation of topological structure of input data makes the use of the network attractive for researchers in various applications. SOINN as an online incremental clustering method, offers relatively high computational speed with low computational cost. Furthermore, SOINN network size is controlled and its stability is achieved with a "garbage collector" technique. The technique defines a parameter called age which is the time period nodes will be removed if they are not updated for a specified time and thus dynamically eliminate noise data. This property makes it attractive for dynamic environments where long-term learning is required.

Just like in this postgraduate thesis other studies (Xiang et al.. 2016 : 815-823) used SOINN as a clustering method with supervised data to be applied in intrusion detection. The SOINN in this thesis will be fed with unlabeled and thus unsupervised data for the clustering but the framework devised along with SVM, more details will be given in the next chapter, will be using supervised, labeled data.

Although, in the proposed framework of this paper, the size and growth of the network will be controlled by a new parameter called n where multiple pairs of SOINNs will be used as a supervised clustering method. SOINN is a two-layer neural network of which the first layer is used to generate a topological structure of input pattern and the second layer, use nodes identified in the first layer as the input data set. In order to determine if an input sample belongs to previously learned clusters or to a new cluster the Euclidean distance between the nodes is used as seen below.

SOINN as seen in **Fig. 5** initializes the network with an empty set of nodes and then the first two nodes are added to the list with the weight vectors set as the two input vectors.

After the initialization, for every input vector it finds the nearest node (winner) and the second-nearest node (second winner) of the input vector by measuring the distance S1 and S2 from every input to every

node with the following equations:

$$s_1 = \text{argmin}_{c \in \mathbb{A}} dist\,(\mathbf{x}, \mathbf{w}_c) \quad (1)$$

$$s_2 = \text{argmin}_{c \in \mathbb{A} - \{s_1\}} dist\,(\mathbf{x}, \mathbf{w}_c) \quad (2)$$

Figure 6: SOINN distance equations

If the input vector belongs to the same cluster of the winner or second winner based on the distances calculated against a similarity threshold criterion, update the weight vector of the node and its neighbors with the weight vector of the input vector and connect it to the node by an edge. If the input vector does not belong to the same cluster of the winner or second winner, add a new node to the network.

3.3.2 *n*- SOINN

Inspired by n- SOINN (Kawewong et.al. 2013 : 496 - 502), where the original SOINN is modified in two ways to utilize multiple pairs of SOINNs as a supervised clustering method, the proposed framework of this paper also uses the same technique. First by adding a global parameter to control the topology of the network and second the standard Euclidean distance is used to calculate the distance between the input and the nodes instead of the normal-basic Euclidean distance.

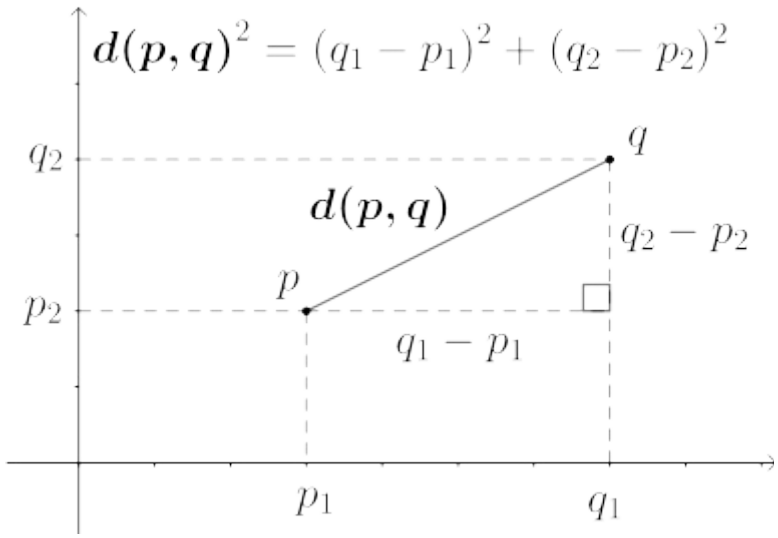


Figure 7: Euclidean Distance in Euclidean space

In order to control the number of output nodes of the network, the difference of how accurate the compressed information will be among the two types of SOINNs - the negative and the positive, a parameter named n is introduced.

This parameter, dictates that any first winner node that wins more than n times assign a win to the second winner node. Only and only if the second winner node also has a winning times of more than n, a new node is generated. Setting n = 0 then the network behaves exactly like the original SOINN. Setting n > 0 then the number of nodes created will be less than n = 0 and therefore less accurate.

The normal Euclidean distance used in the original SOINN was intended for the purpose of a single SOINN to realize the unsupervised incremental learning task. The Euclidean distance between points p and q is the length of the line segment connecting them. In Cartesian coordinates, if p = (p1, p2,..., pn) and q = (q1, q2,..., qn) are two points in Euclidean n-space, then the distance (d) from p to q, or from q to p is given by the following formula in In the context of Euclidean geometry for n dimensions:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

Figure 8: Euclidean Distance equation

The proposed framework uses multiple SOINNs with the objective of clustering and to calculate the distance function dist, as seen in **Fig. 6** between the input and the weight vector of n-SOINN, by the normal Euclidean distance would be a non-trivial task, the distance - the variance of data in each class is calculated by the standardized - normalized Euclidean distance instead of the normal-basic Euclidean distance.

$$d^2(\mathbf{p}, \mathbf{q}) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2.$$

Figure 9: Normalized - Squared - Euclidean Distance equation

This helps in balancing the different properties of the attacks and multiple SOINNS provide, the reason for that is, the original input data data are not preserved by the framework and a representative node of the input data can be eliminated by the way of the SOINN algorithm's garbage collector whenever it

becomes non-popular.

3.4 Support Vector Machine

Support Vector Machines were first introduced in 1992 in a work by Boser et al. where they proposed a training algorithm that maximizes the margin between the training patterns and the decision boundary. (Boser et al. 1992 : 1). Their objective was to solve classification problems in the machine learning field.

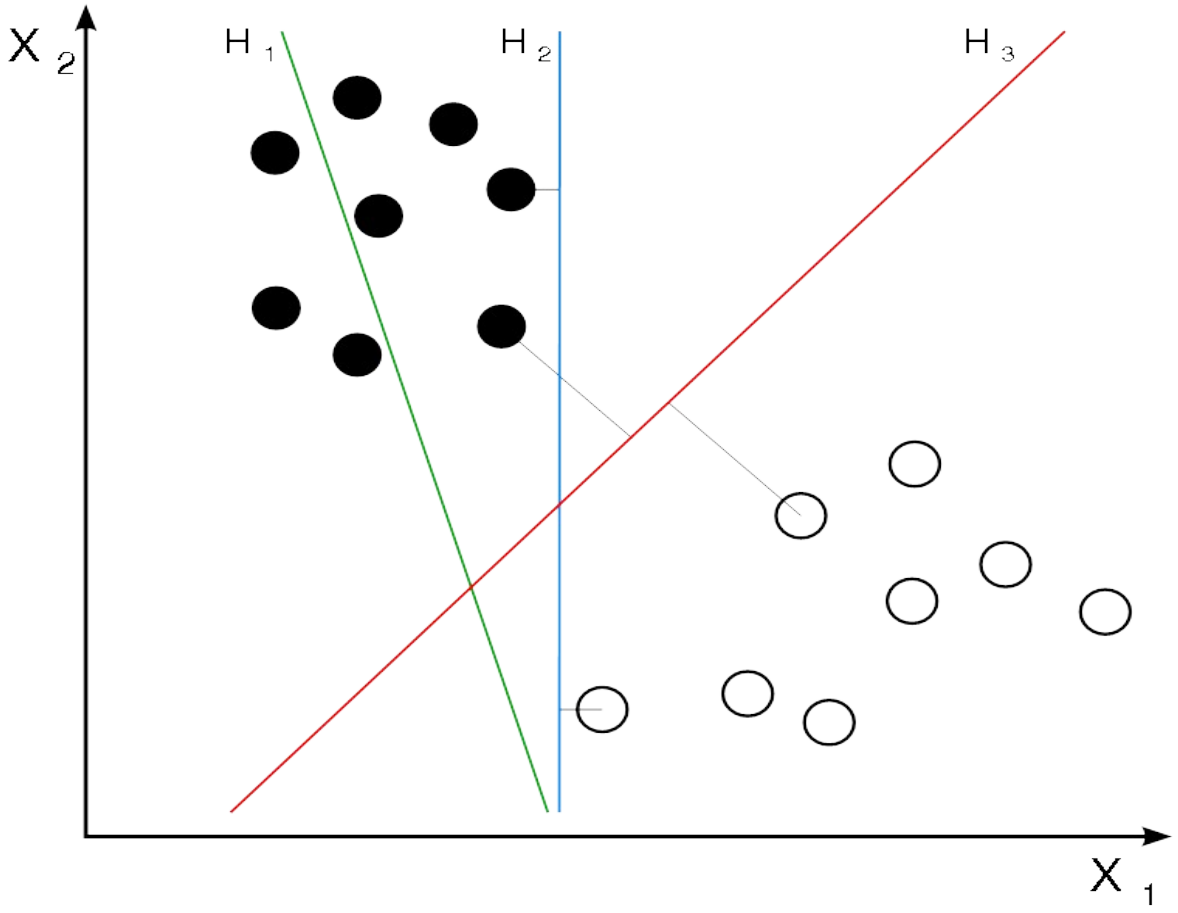


Figure 10: SVM separation hyperplanes

Support vector machine - SVM fall in to the category of supervised learning and just like a neural network, its function(s) learns by example. It generates input-output mapping functions - i.e classification or regression, from a set of labeled training data. For classification, nonlinear kernel functions are often used to transform input data to a high-dimensional feature space in which the input

data become more separable compared to the original input space (Wang 2005)

Support Vector Machine operates as a group classifier by constructing a hyperplane or set of hyperplanes in a high dimensional space using training data to separate data into two groups. One of the main advantages of SVM is its effectiveness in high dimensional spaces and its ability to discriminate data which are not readily separable by simpler methods. In order to make the separation easier for non linearable separable datasets to discriminate, or for overlapping variations of features for two classes, the algorithm maps the original data space into much higher space to make the separation easier.

Tasks like classification or regression can be accomplished by the construction of hyperplanes as seen in **Fig. 10** where maximizing the minimum distance from the separating hyperplane. For example, the classifier generalization error could be minimized by an optimal separator - maximum margin, hyperplane 1 does not separates the class, hyperplane 2 does separates the classes but only with a small margin and hyperplane 3 is optimal - maximum margin, compared with the other hyperplanes.

Although SVMs are well developed and used extensively for machine learning projects and research alike, from 2000 to 2010 LIBSVM (Chang and Lin 2011 : 27) has been downloaded more than 250,000 times, in order to achieve good results first an appropriate kernel needs to be identified and then optimize its function for mapping of features and the other parameters by optimizing it with different optimization methods.

In this thesis the RBF kernel in both binary and multi-class SVMs was used and the optimization method used was a cross-validated grid-search over a parameter grid.

3.4.1 WTA - SVM

Support Vector Machine is a supervised machine learning algorithm which was designed and is fundamentally a binary classification algorithm to solve two classes - binary problems. Although effectively extending SVMs for multiclass classification is still an ongoing research problem, several methods have been proposed where a multi-class classifier is constructed by combining several binary classifiers (Hsu et.al. 2002 : 415-425). Two SVMs common methods are employed to solve the multi-class problem by reducing the single multi-class problem to into multiple binary classification problems, the one-versus-all method and the one-versus-one method.

The one-versus-all method is using the winner-takes-all strategy in which the classifier with the highest output function gets the class assignment. It constructs k classifiers, where k is the number of classes and the m^{th} classifier is trained with all of the examples (input) in the m^{th} class with positive labels, and all other examples with negative labels. Another problem with the one-versus-the-rest approach is that

the training sets are imbalanced, in this thesis before using the training sets for every class, each training set is first balanced.

The one-versus-one method trains *k(k-1)/2* different 2-class classifiers on all possible pairs of classes, and then uses a max-wins voting strategy in which the class with the highest number of votes - a voting is performed among the classifiers, in every win of the two class problem gets the class assignment.

In this paper the approach of one-versus-all method using the winner-takes-all strategy (WTA) is employed to solve the multi-class problem.

3.5 Dataset

The NSL-KDD dataset is an updated and refined version of the KDD99 dataset (Mahbod, et al. 2009 : 1-6), a widely used publicly available dataset for network-based anomaly detection systems. The KDD99 intrusion detection contest - The Third International Knowledge Discovery and Data Mining Tools Competition, dataset consists of five million connection records of TCP dump data from seven weeks of network traffic. It was prepared and managed by MIT Lincoln Labs for the 1998 DARPA Intrusion Detection Evaluation Program. The objective of the program was to survey and evaluate research in intrusion detection. Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data simulating multiple attacks against a typical U.S. Air Force LAN.

A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes (Stolfo et al. 2000 : 8).

The NSL-KDD data set has several advantages over the original KDD data set. The most important advantages are 1) the NSL-KDD does not include redundant records in the train set and as such the the classifiers will not be biased towards more frequent records; 2) there are no duplicate records in the separate test sets and therefore, the performance of the learners is not biased by the methods which have better detection rates based on the frequent - duplicated records; 3) the number of records in the train and test sets is reasonably reduced to a point where it is affordable to experiment on the complete dataset without the need to split and randomly select smaller chunks of the dataset and therefore maintain consistent and comparable evaluation results across the board (Mahbod, et al. 2009 : 1-6).

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This makes the task more realistic. The

datasets contain a total of 24 training attack types, with an additional 14 types in the test data only. The attack classes are grouped into four categories : Denial of Service, Probe, User to Root and Remote to Local attacks.

A Denial of Service - DoS attack, is the type of attack in which an attacker executes network requests in such a way where it consumes all of the resources of a networked computer - usually a web server, resulting in the failure of that computer to serve legitimate network requests and therefore drop legitimate users access.

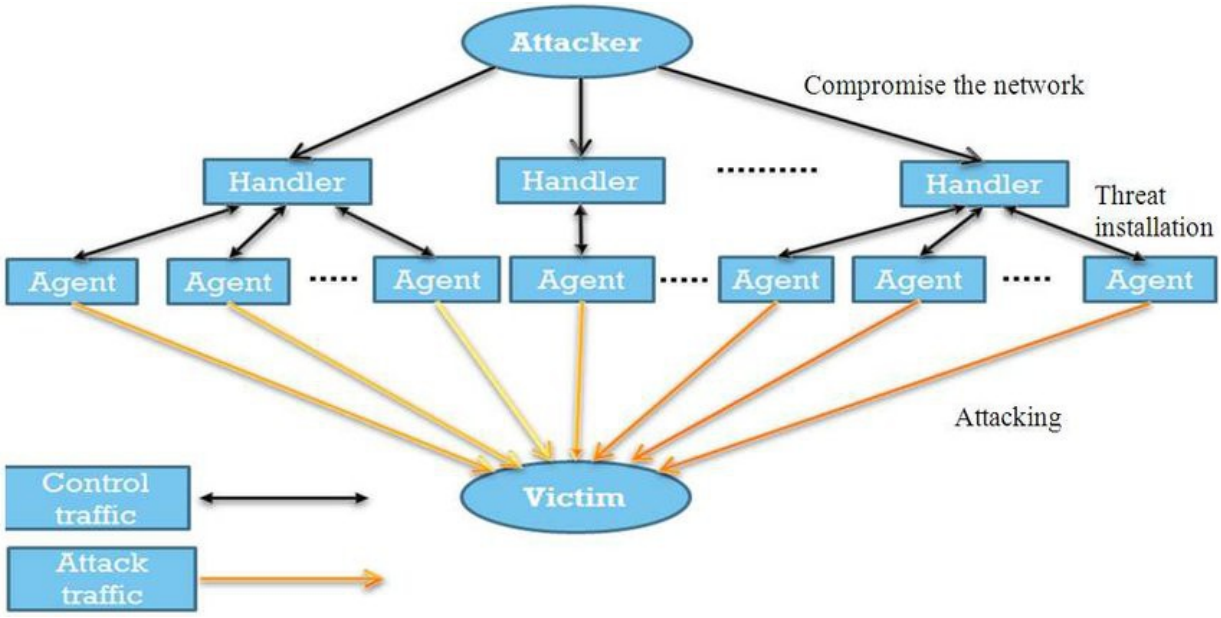


Figure 11: Denial Of Service Attack

A Remote To User (R2L) attack is an attack in which an attacker exploits a computer's vulnerability via the network to remotely gain unauthorized access to that computer and gain local machine user privileges.User to Root Attacks (U2R) attack is a privilege escalation attack where an attacker already has a certain level of privileges of an organization's computer system and attempts to escalate privileges - gain unauthorized access to a different level of privileges privileges of what the attacker had already given, usually root or administrator by exploiting a computer's vulnerability.

Probe is an active reconnaissance type of computer attack in which an attacker actively engages with the targeted system to gather information about a target systems and network, usually enumeration is involved to discover vulnerabilities.

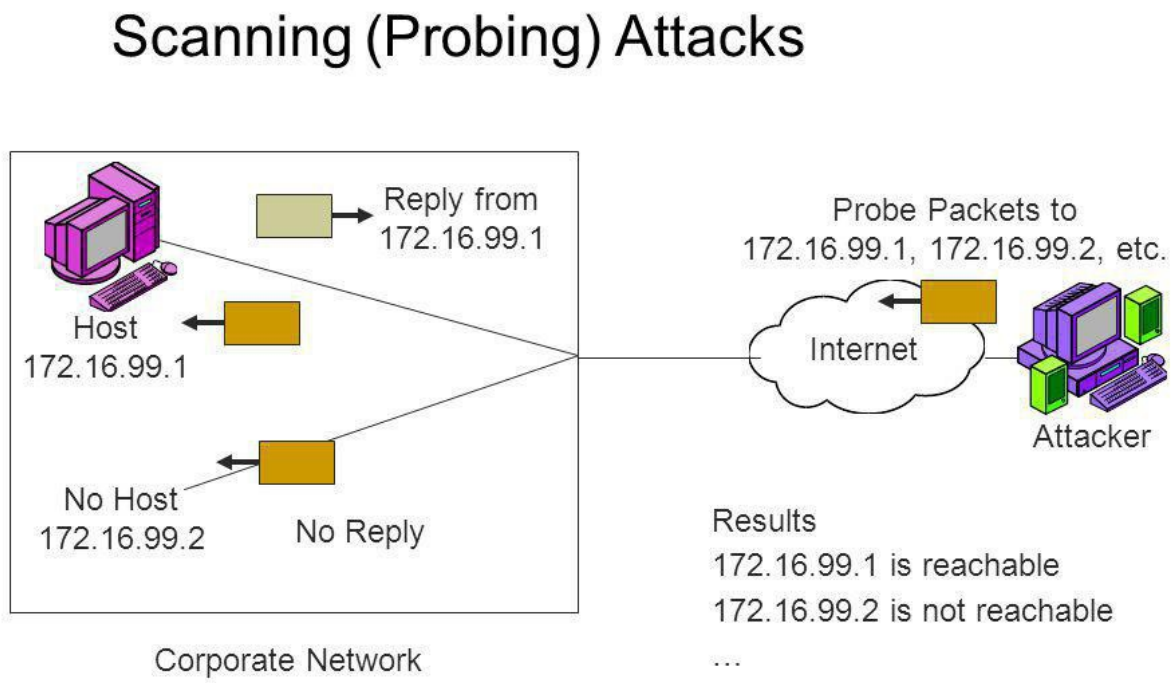


Figure 12: Probe Attack

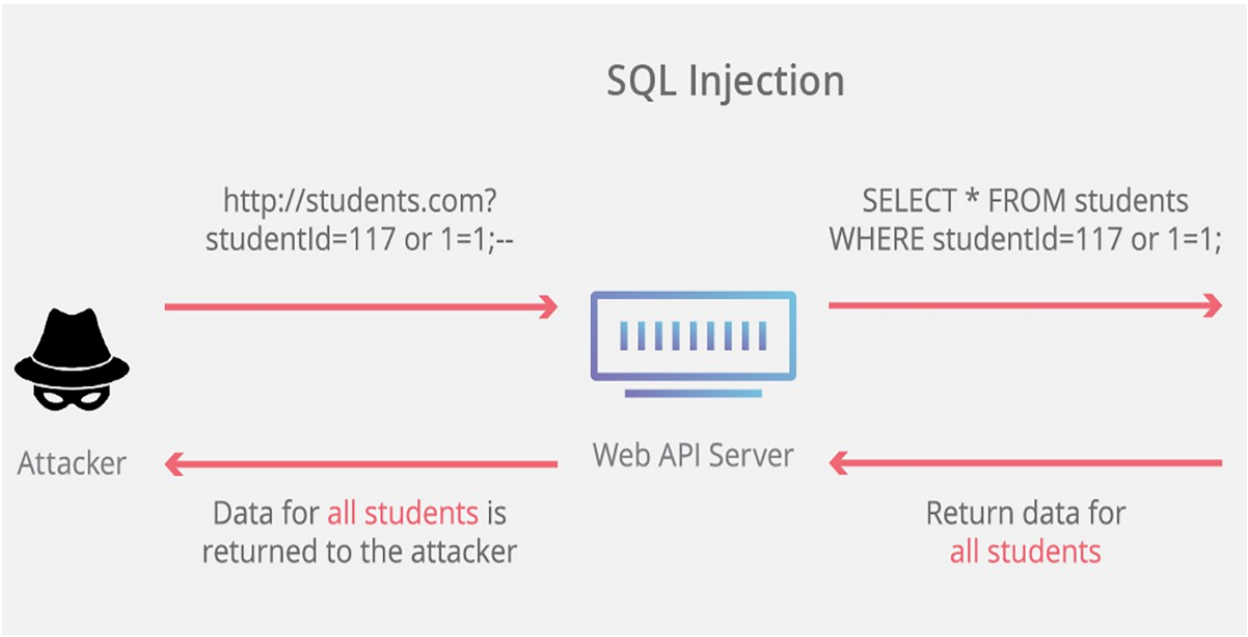


Figure 13: SQL Injection Attack

In each record there are forty one attributes describing different features of the TCP connection and a label assigned to each either as an attack type or as normal. The types of attacks are classified into four major categories; DoS, Probe, R2L and U2R as seen in Table 1.

Attack Class	Attack Type	# of Attacks
DoS	Back, Land, Neptune, Pod, Smurf, Teardrop,Apache2, Udpstorm, Processtable, Worm	10
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint	6
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named	16
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps	7

Table 1: Classification of types of attacks

In table 2 below the details of the attributes namely the attribute name, their description, sample data and type information of all the 41 attributes available in the NSL-KDD data set (Mahbod, et al. 2009 : 1-6) are listed.

Attribute No.	Attribute Name	Description	Sample Data
1	Duration	Length of time duration of the connection	0
2	Protocol_type	Protocol used in the connection	Tcp
3	Service	Destination network service used	ftp_data
4	Flag	Status of the connection – Normal or Error	SF
5	Src_bytes	Number of data bytes transferred from source to destination in single connection	491
6	Dst_bytes	Number of data bytes transferred from destination to source in single connection	0
7	Land	If source and destination IP addresses and port numbers and equal then,this variable takes value 1 else 0	0

8	Wrong_fragment	Total number of wrong fragments in this connection	0
9	Urgent	Number of urgent packets in this connection.Urgent packets are packed with the urgent bit activated.	0
10	Hot	Number of „hot‟ indicators in the content such as: entering a system directory, creating programs and executing programs	0
11	Num_failed_logins	Count of failed login attempts	0
12	Logged_in	Login Status : 1 if successfully logged in; 0 otherwise	0
13	Num_compromised	Number of ``compromised'' conditions	0
14	Root_shell	1 if root shell is obtained; 0 otherwise	0
15	Su_attemped	1 if ``su root'' command attempted or used; 0 otherwise	0
16	Num_root	Number of ``root'' accesses or number of operations performed as a root in the connection	0
17	Num_file_creations	Number of file creation operations in the connection	0

18	Num_shells	Number of shell prompts	0
19	Num_access_files	Number of operations on access control files	0
20	Num_outbound_cmds	Number of outbound commands in an ftp session	0
21	Is_hot_login	1 if the login belongs to the ``hot'' list i.e., root or admin; else 0	0
22	Is_guest_login	1 if the login is a ``guest'' login; 0 otherwise	0
23	Count	Number of connections to the same destination host as the current connection in the past two seconds	2
24	Srv_count	Number of connections to the same service (port number) as the current connection in the past two seconds	2
25	Serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)	0
26	Srv_serror_rate	The percentage	0

		of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)	
27	Srv_serror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)	0
28	Rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)	0
29	Srv_rerror rate	The percentage of connections that were to the same service, among the connections aggregated in count (23)	1
30	Diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in count (23)	0
31	Srv_diff_host_rate	The percentage of connections that were to	0

		different destination machines among the connections aggregated in srv_count (24)	
32	Dst_host_count	Number of connections having the same destination host IP address	150
33	Dst_cost_srv_count	Number of connections having the same port number	25
34	Dst_host_same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)	0,17
35	Dst_host_diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)	0,03
36	Dst_host_same_src_port_rate	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)	0,17
37	Dst_host_srv_diff_host_rate	The percentage	0

		of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count(33)	
38	Dst_host_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32)	0
39	Dst_host_srv_serror_rate	The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33)	0
40	Dst_host_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)	0,05

41	Dst_host_srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)	0

Table 2: Attributes Description

Type	Features
Nominal	Protocol_type(2), Service(3), Flag(4)
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21),, is_guest_login(22)
Numeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11), num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23) srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29) diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41)

Table 3: Attribute Value Type

Chapter 4

Proposed Framework

4.1 Incremental Learning NIPS

The proposed framework as seen in **Fig. 14**, consists of a detection engine, the core of the framework, a preprocessing module of the incoming traffic that feeds the detection engine, a validation module to evaluate the results of the detection engine and an update module where the failed results are fed back to the detection engine.

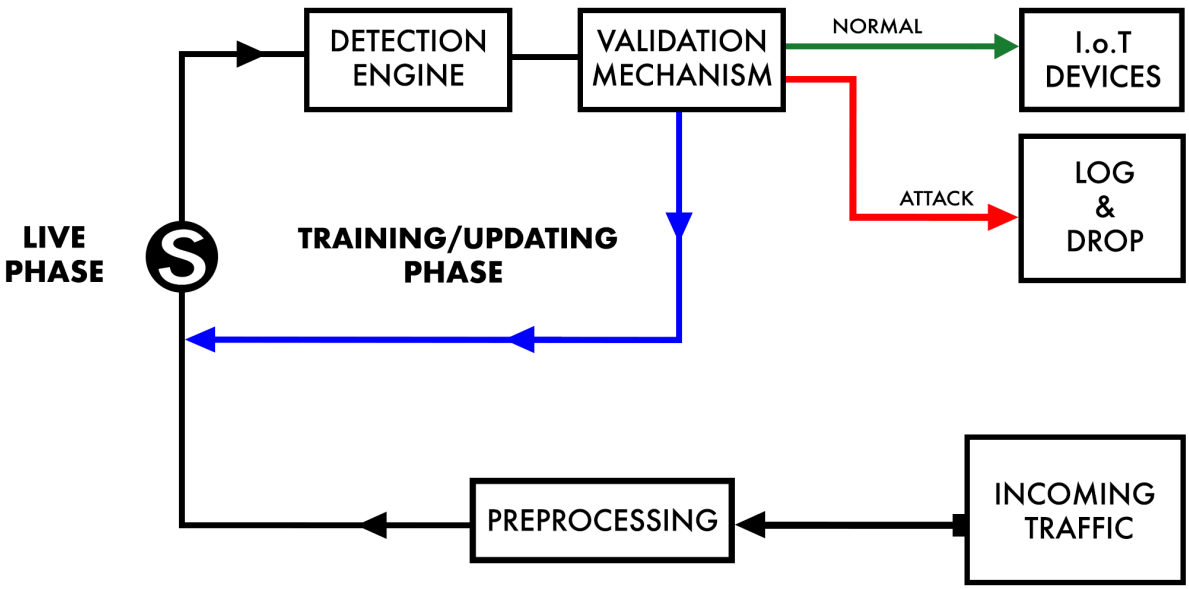


Figure 14: Incremental Learning - NIPS

The core of the framework is based on its detection engine, the mechanism that will, with a relative small sample of input data, detect adequately enough, basic forms of network attacks. Subsequently, as

more network data become available, the mechanism will be updated - incrementally, by input data of classes that it failed to detect in order to refine and advance its protective capabilities.

The preprocessing module captures, extracts and transforms the data at the TCP layer to be fed to the detection engine. The validation module validates the results of the detection engine - either by a human or by any other means in an automated way, saves the results it in order to feed the detection engine and at the update phase the detection engine is retrained with the updated module without losing its previously acquired knowledge. The update module feeds the failed classifications the detection engine predicted as a way of input back to it in the updated phase. After the initial training the system switches between two different phases, the live phase, where the input traffic is categorized between normal or attack - four different categories of attacks where used in experimentation and evaluation and the updated phase, where the system is incrementally updating its machine learning capabilities by feeding it with new input data - failed classifications.

4.1.1 Concept

The framework was conceptualized on the premise that a network protective mechanism can be build to initially learn from a relative small sample of network data, adequately enough, for basic network protection. Subsequently, as more network data become available, the mechanism will be updated incrementally, by input data of classes that it failed to detect in order to refine and advance its protective capabilities. The mechanism that decides whether a decision failed or not is decided by a validation mechanism. In order to advance its capabilities the core mechanism must be able to categorize the network data in a multi-class manner, not just if a connection is an attack or not, but what kind of attack that is and therefore provide a solution to an incremental learning multi-class problem.

The core of the framework is based on its detection engine, the mechanism that will, with a relative small sample of input data, detect adequately enough, basic forms of network attacks. Subsequently, as more network data become available, the mechanism will be updated incrementally by input data of classes that it failed to detect in order to refine and advance its protective capabilities. The detection engine and the core of the framework as seen in **Fig. 15**, is initialized with a dataset of k attack classes where x is a d dimensional TCP network feature vector of the connection and y is an attack class category label. A network connection attack category is modeled by two n-SOINNs, one with a high n-value and another with a low n-value. For every k class, a pair of two n-SOINNs and one SVM binary classifier is created. The n-SOINN with the low n-value is supposed to hold more accurate compressed information than the n-SOINN with the high n-value making it a binary problem. For every pair of n-SOINNs, the positive n-SOINN is considered to be the one with the low n-value and the negative one with the high n-value. The input vector of every SVM binary classifier is constructed by the output of the

positive k n-SOINN and all other negative n-SOINNs. The output class along with its score of every binary SVM is then compared with all other binary SVMs in order to choose the top m classes. After choosing the top m classes, the output of their respective n-SOINNs is combined as an input to a multi-class SVM to get the final class.

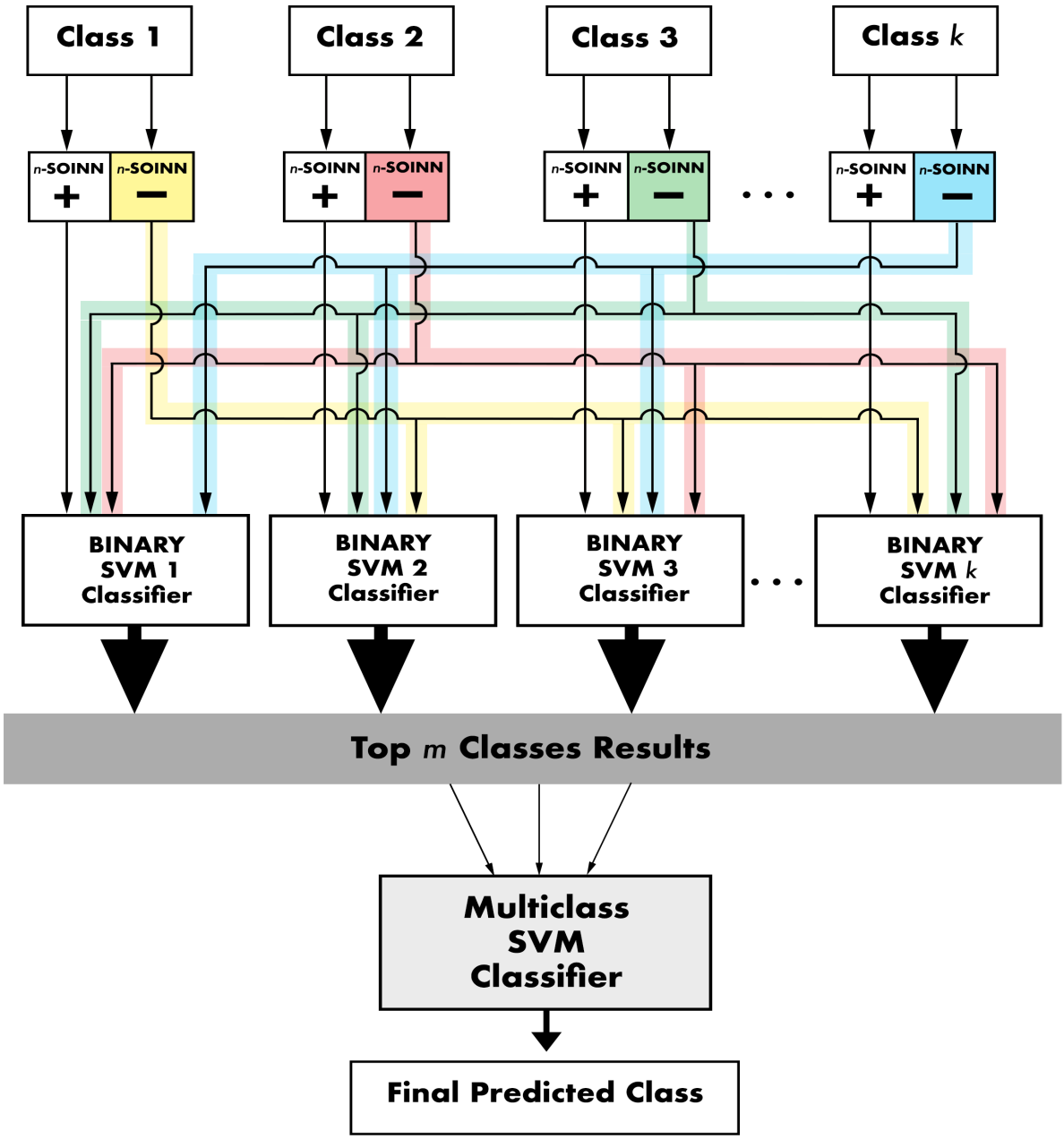


Figure 15: n-SOINN-WTA-SVM

4.2 Framework modules

The framework consists of the detection engine - n-SOINN-WTA-SVM, the core of the framework and the novelty which this thesis proposes, a preprocessing module that prepares the incoming traffic that feeds the detection engine, a validation module to evaluate the results of the detection engine and an update module where the failed results are fed back to the detection engine.

4.2.1 Preprocessing

This module captures and process incoming traffic in real time. Features of TCP connection characteristics are captured and processed in order to be made available as an input vector to the detection engine - n-SOINN-WTA-SVM. The framework is based on the premise that TCP characteristics of a connection suggest whether a connection is defined as an attack or not and evermore if it is defined as an attack what kind of an attack that is. For the purposes of this thesis this module was represented by the feature selection of the dataset to show proof-of-concept. From the 42 features of the dataset available as seen in Chapter 3, the author choose only the 32 numerical ones to use for the learning process. The feature selection was based on the preliminary results of the experimentation which showed that the model performed better and more stable with the numerical results only. The case that the use of a reduced number of features in the NSL-KDD dataset can deliver enhanced or comparable performance was shown by Sung and Mukkamala where they reduced the features with the method of elimination to identify important features for each of the 5 classes of intrusion patterns . (Sung and Mukkamala 2003 : 209-216).

The 70 attacks where then categorized into 5 classes, four major attack categories; DoS, Probe, R2L and U2R as seen in Table 1 and the normal class.

4.2.2 Detection Engine - n-SOINN-WTA-SVM

The core module of the framework consists of two main parts as seen in **Fig. 15**. The clustering part where a pair of n-SOINNs is used for each class to compress the information given from the TCP connections by the preprocessing module and achieve incremental learning. For every class a pair of n-SOINNs is created a negative and a positive, one with a high n-value and another with a low n-value. The n-SOINN with the low n-value - the positive n-SOINN, is supposed to hold more accurate compressed information than the n-SOINN with the high n-value - the negative n-SOINN, making it a binary problem and thus applicable to any binary suited algorithm solution. For every pair of n-SOINNs, the positive n-SOINN is considered to be the one with the low n-value and the negative one with the high n-value.

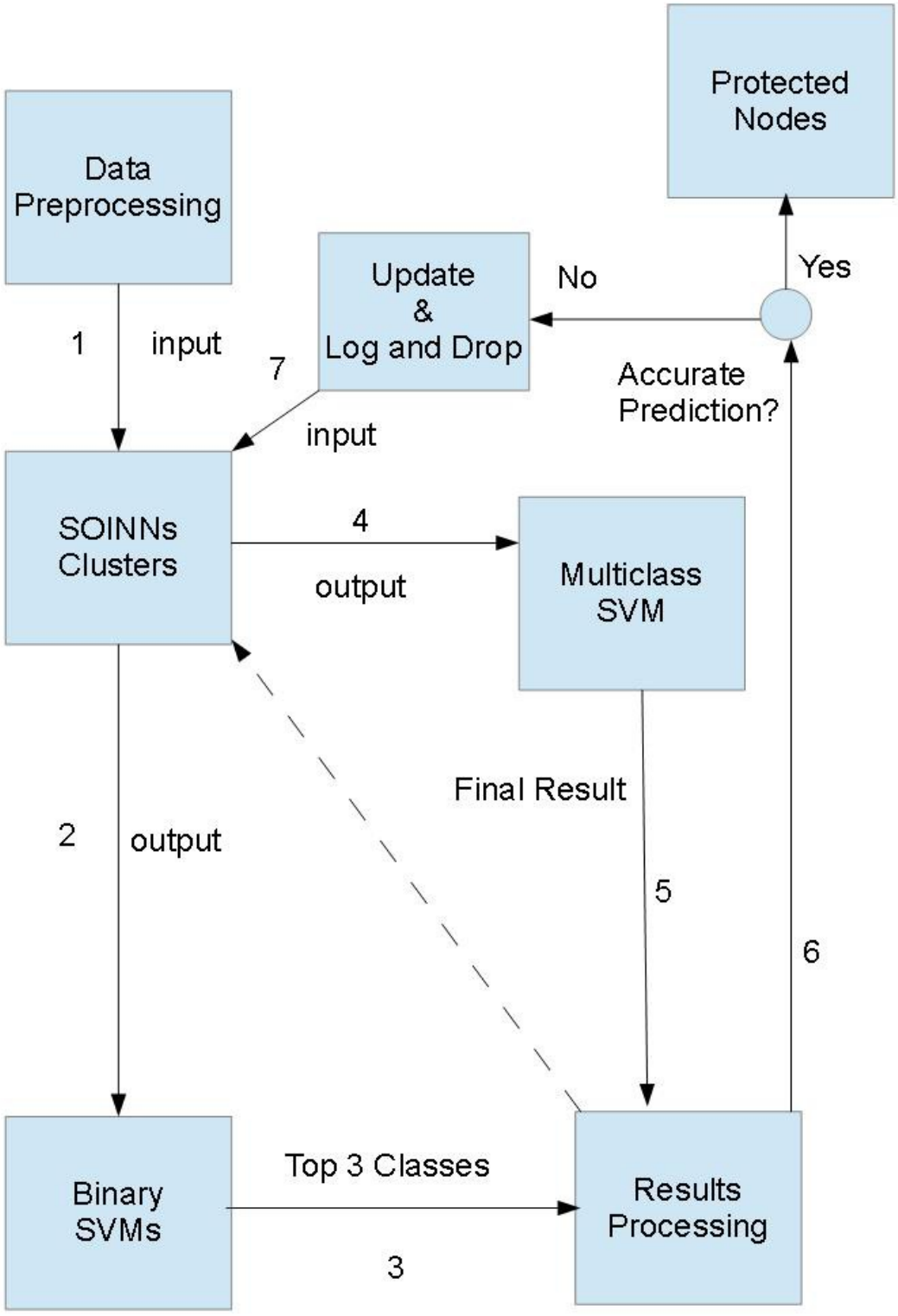


Figure 16: Algorithm Flowchart

The differentiation of negative and positive is realized by the way each n-SOINN feeds the binary SVMs for class k . Inspired by the winner-takes-all strategy applied to a multiclass SVM, the output of the positive n-SOINN feeds classifier k and the negative n-SOINN feeds all other classifiers as depicted in **Fig. 15**. The classifying part takes the compressed information from the n-SOINNs - the output of the n-SOINNs nodes, and constructs an input for an SVM classifier for each class to perform the preliminary classification.

To choose the top m classes, the predicted class along with its score of every binary SVM is then compared with all other binary SVMs prediction scores and the top m classes are then sorted by score. The top m classes are the winners for the next phase. The top m classes pairs of each classifier is then classified by a multi-class SVM for the final decision. The input-output process is exactly the same as the previous phase, meaning that the output of the top m SOINNs pairs is used as the input to the multi-class SVM instead of the binary SVMs to perform the final prediction.

The reason to perform another classification is that the discriminant function is based on the differentiation of the SOINNs compressed positive and the negative input data and thus it is possible that the first and the second best predictions might deviate from the expected and correct class. It must be noted that k is user defined according to the system needs in terms of the number of classes available and the score is calculated on the distance of the samples to the separating hyperplane.

4.2.3 Validation Module

The validation module serves the purpose of improving and advance the framework's accuracy by confirming the predicted label manually or automatically. It is the module that given a small sample of input data and with the online and incremental learning method and technique, life long learning could be achieved. Incremental learning could be achieved by a manual method where a network security expert administrator could interact in real time and or at a later time, assuming the predictions are saved, validates whether the predictions made by the detection engine are valid or not. Alternative, an automatic method with the predictions saved for later use and compared against known and confirmed predictions could be used, the module then will forward the failed predictions to the updating module. For the purpose of a proof-of-concept in this thesis, the validation module used the automatic method and the comparison was made against subsets of the NSL-KDD dataset as seen in Chapter 5. The validation module also serves the purpose of providing the information of whether a connection is an attack or not and based on the prediction the connection is then dropped or forwarded to the

destination. The information could be forwarded by the validation module to a different future module where it could log, alert and drop connections.

4.2.4 Updating Module

The framework was designed to operate in two phases, the live phase where the detection engine is making decisions based on its previously gained knowledge accumulated and an update phase where the update module is updating the system with failed predictions to improve its capabilities.

The validation module pass on the failed prediction results to the update module which updates the SOINNs with the failed prediction data in the clustering mechanism which holds the compressed data. The SOINNs cluster weights get updated with every new round of updates, which in turn hold more accurate information than before the update. Due to the online capabilities of SOINN, the previously gained knowledge is not lost with every new round of updates, instead the new information is accumulated into a more accurate knowledge clustering mechanism. This update mechanism along with the SVM classification coupling brings novelty to this framework with an incremental online learning capability.

The phases could run in parallel if needed in production or switching between the two phases according to the network traffic. For example, if there is no network traffic the update phase will trigger the update module to update the system. For the purpose of this postgraduate thesis, the phases run in series - manually to show proof-of-concept.

Chapter 5
Experimental Results

Experimentation and evaluation was performed with the NSL-KDD dataset, which is an improved version of well-known KDD'99 dataset, a very good candidate dataset to evaluate the performance of any IDS and thus our framework. Although it is stated that the NSL-KDD dataset still suffers from some of the problems discussed in Chapter 3 and may not be seen as representative for applications of today's world, the author believes it is suited for the objective of this thesis because it is an effective benchmark dataset to detect network intrusions and it serves the purpose of the framework which is to show proof-of-concept of an online incremental-learning intrusion detection system. For the data structures Pandas, an open source Python Data Analysis Library was used to perform feature selection, data manipulation and analysis.

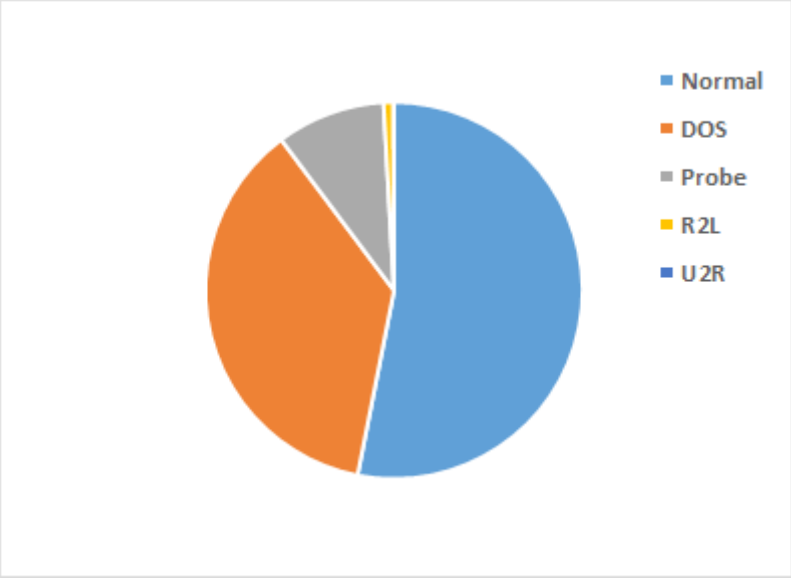


Figure 17: Class Distribution of NSL-KDD dataset

With the use of Pandas feature selection of 32 out of 42 features was performed and the 70 attacks where then categorized into 5 classes, four major attack categories; DoS, Probe, R2L and U2R and the normal class. The distribution of the classes in the datasets for normal and attack (2 classes) is 53% for the normal class and 47% for the attack class. At five classes, the distribution is 53% for the normal class, 37% for the DoS class, 9% for the Probe class, 0.7 for the R2L class and 0.3% for the U2R class.

The original SOINN is modified in two ways to utilize multiple pairs of SOINNs, for the negative SOINN we used n=2 and for the positive SOINN we used n = 100. This numbers where taken by the paper of Kawewong et.al (Kawewong et.al. 2013 : 496 - 502), which inspired this thesis. Different numbers combinations have been tested to verify the suitability for this framework and the results showed that n=2 for the negative SOINN and n = 100 for the positive SOINN yielded the best results.

The classes of the dataset were five and thus five binary classes - a pair of negative and positive SOINN for each class were created for this experiment, a normal class and four attack classes, namely - Denial of Service, Probe, R2L and U2R.

For the the top m classes we choose the smallest possible number after two m = 3 which is the best performer in a multi-class SVM out of the box in the LIBSVM (Chang and Lin 2011 : 27), the RBF kernel in both binary and multi-class SVMs was used. Both C and gamma in binary and multi-class SVM's where optimized by cross-validated grid-search over a parameter grid.

The dataset consists of two different subsets, a subset with 125973 records and a subset with 22544 records. The second subset is not from the same probability distribution as the first one and it also contains specific types of attacks not present in the first one. For the framework evaluation and in order to show that incremental learning is achieved we have divided the first subset into five smaller subsets used for the test/update rounds, subset one with 25197 records, subset two with 25196 records, subset three with 25194 records, subset four with 25193 records, subset five with 25193 records and used the second larger subset from the original with 22544 records for the initial training. It should be noted that after the initial training for every update round, the subset is tested against the trained n-SOINN-WTA-SVM and only the failed predictions are fed back to the system. The results as shown in **Table 4**, indicate that the framework can achieve incremental learning with promising prospects.

Final Prediction Results			
Rounds	**Accuracy %**	**Time in Seconds**	**# Samples**
Initial Training	78.23	986	22544
Round 1 Update	84.44	1065	28028
Round 2 Update	87.98	1647	31948
Round 3 Update	88.88	2328	34975
Round 4 Update	88.96	2801	37776
Round 5 Update	89.67	3285	40557

Table 4: Multiclass Final PRedictions Accuracy - Results table

The samples shown in the table are the accumulated samples from the previous round. For example, for the initial training 22544 records was used and for the first round of updates 5484 records were fed back to the system, the number of failed predictions against the first test subset out of five with 25197 records, accumulating the number of samples as input to 28028. After the initial training, the model predicted 78% correct classes with 5484 failed predictions out of 25197, after the first round of updates the model predicted 84,44% correct classes with 3920 failed predictions out of 25196, after the second round of updates the the model predicted 87,98% correct classes with 3027 failed predictions out of 25194, after the third round of updates the the model predicted 88,88% correct classes with 2801 failed predictions out of 25193, after the fourth round of updates the the model predicted 88,96% correct classes with 2781 failed predictions out of 25193 and finally after the fifth round of updates the the model predicted 89,67% correct classes with 2602 failed predictions out of 25193.

Top 3 Predictions Results			
Rounds	Accuracy %	Time in Seconds	# Samples
Initial Training	83.01	986	22544
Round 1 Update	87.37	1065	28028
Round 2 Update	89.22	1647	31948
Round 3 Update	90.36	2328	34975
Round 4 Update	91.14	2801	37776
Round 5 Update	95.02	3285	40557

Table 5: Binary Top 3 Predictions Accuracy - Results table

Before feeding the output of the SOINNs to the multiclass SVM to get the final prediction, the score of each prediction of each of the binary SVMs was compared to get the top 3 winning results. Then the output of each SOINN was used as an input to the multiclass SVM. The top 3 predictions for every round was as follows: After the initial training, the model predicted 83% correct classes with 4283 failed predictions out of 25197, after the first round of updates the model predicted 87,37% correct classes with 3181 failed predictions out of 25196, after the second round of updates the the model predicted 89,22% correct classes with 2715 failed predictions out of 25194, after the third round of updates the the model predicted 90,36% correct classes with 2428 failed predictions out of 25193, after the fourth round of updates the the model predicted 91,14% correct classes with 2231 failed predictions out of 25193 and finally after the fifth round of updates the the model predicted 95,02% correct classes with 1254 failed predictions out of 25193.

The chart as shown in **Fig. 18**, shows the relation between the number of records accumulating and the accuracy of the system. Clearly the accuracy trend is pointing upwards with a 89.67% prediction result.

These accuracy results are recognition results, they are not just accuracy results of a two class problem - attack and normal, the results are based on a 5 multi-class problem, recognizing not just if a record is an attack or not, but what kind of an attack is.

The framework training samples accumulated by the end of Round 5 were 27.30% of the total dataset. This property of the framework makes it suitable for scaling applications because of its efficiency. Instead of feeding the model the complete dataset, in our experimentation the SOINNs kept only a fraction of the information the whole dataset available. In contrast static models are trained with the full data available with out the ability to adapt to the dynamic nature of network data over time. The time cost in seconds for the initial training was 986 seconds and after that, each update round the time cost incrementally increased with the last round costing 3285 seconds as shown in **Fig. 19**.

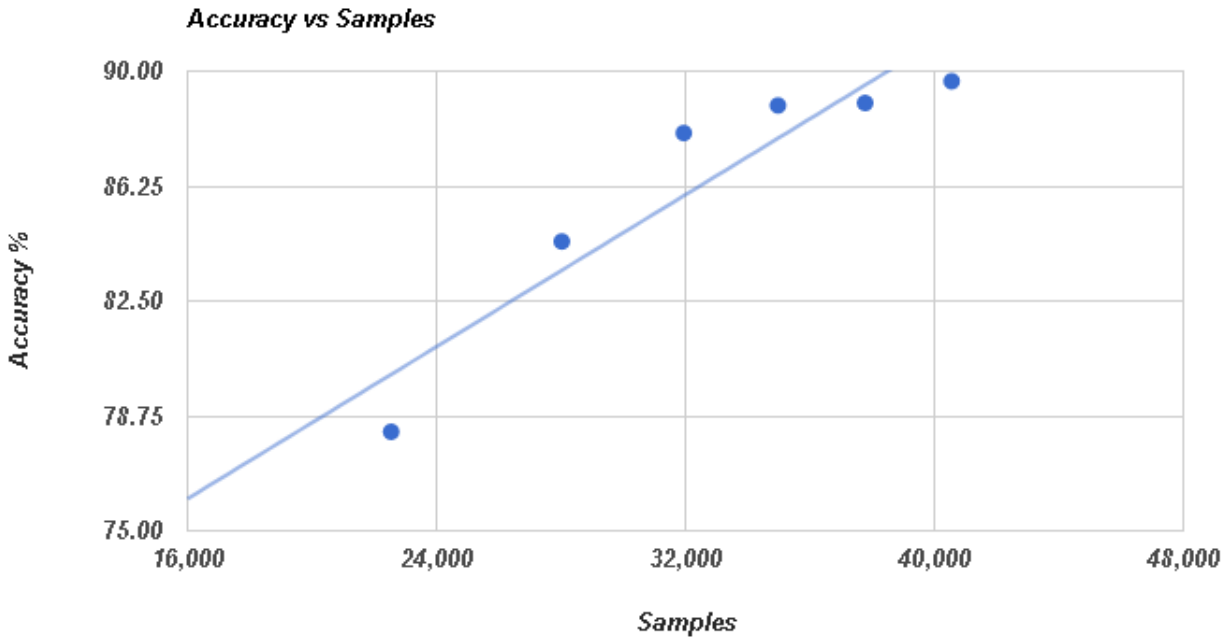


Figure 18: Increasing Accuracy vs # Samples Feedback Chart

It must be noted that the time cost is not just for the training of the model but also the combined time cost for the verification of each record, meaning that each record is tested against the dataset with correct labels to verify whether the prediction was correct or not.

Although, the time increases over input samples of data, these are in essence rounds of updates which increase the number of feedback feeding to the model.

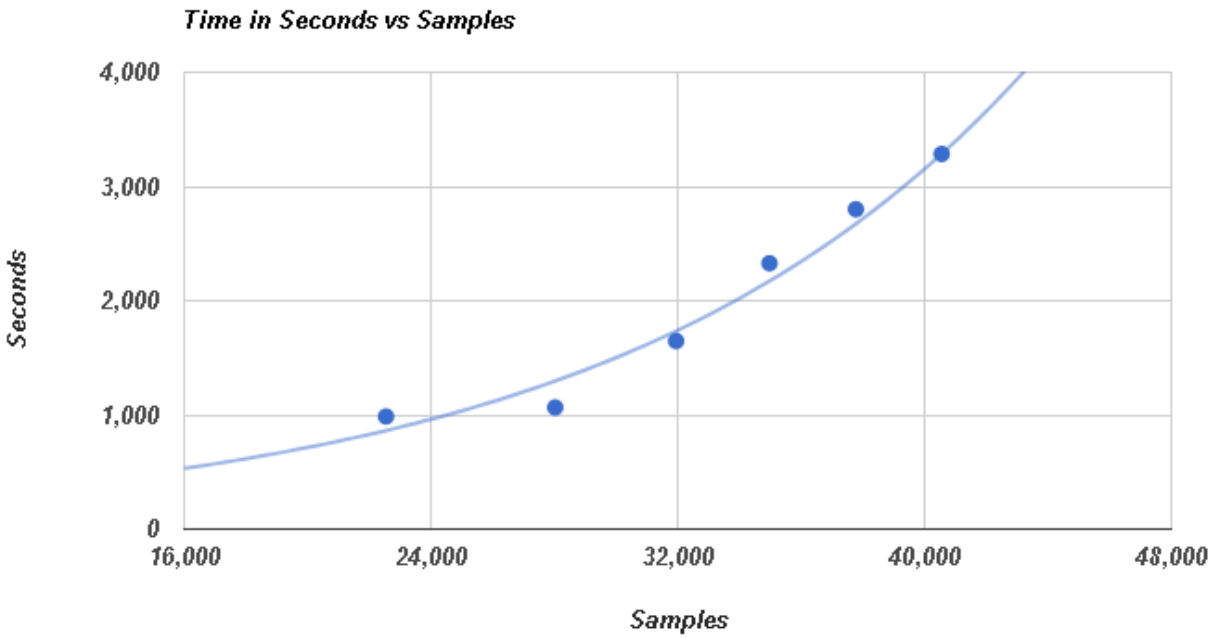


Figure 19: Time Cost vs # Samples Chart

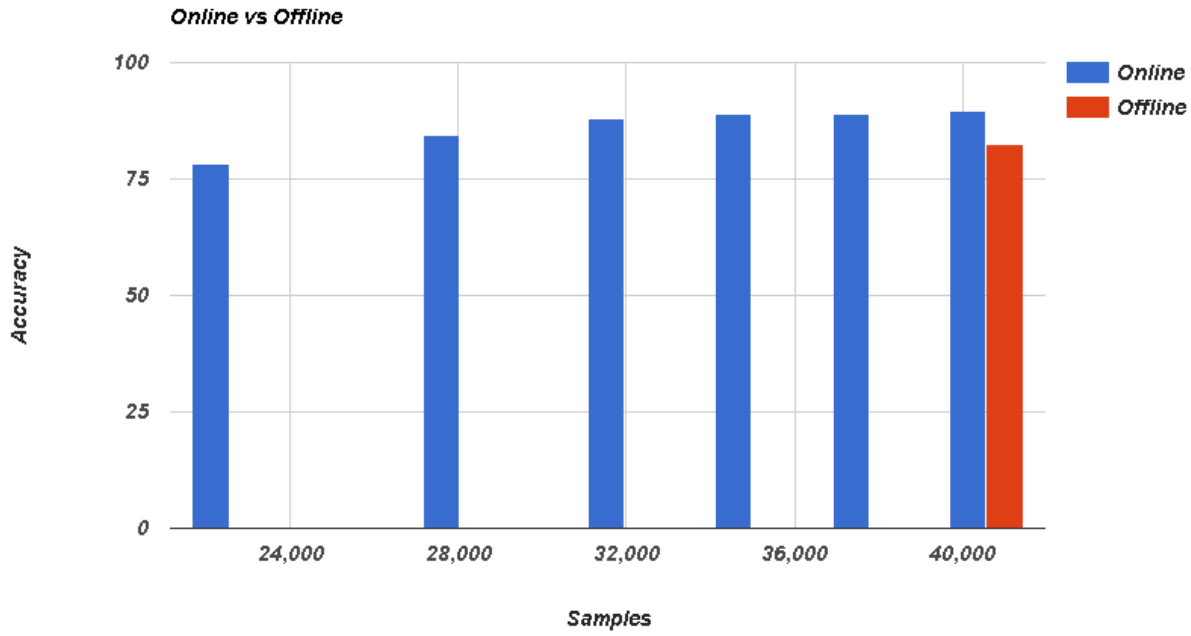


Figure 20: Online vs Offline Method Chart

To compare the online - incremental learning method with the offline method, the framework was trained once in one batch with the same amount of input sample data. The results were 82.59% accuracy classification with a time cost of 2857 seconds. The time cost, just like in the offline method was used with the intention to compare the two methods - meaning that the time cost was based on the fact that the model was trained with the same amount of samples and checked the samples one by one to validate its accuracy. In the online version, for each round the model is trained and then validates each record to check whether is accurate or not and save it for the next update cycle. Each cycle is updating the model with the failed predictions of the previous round. The fact that the offline method achieved less accurate results only make the case stronger that the incremental learning proposed framework could be a competitive candidate for an industrial application.

Chapter 6
Discussion and Feature Work

The incremental learning property of the proposed framework and the evaluation results indicate that our proposal can adapt to the dynamic profile nature of network data for both normal and attack categories. The framework utilize less resources, is faster and it has higher detection rate than the offline method. It utilizes less resources and it is faster because the system is initialized with a small sample of data and as the failed predictions are accumulating, in the updated phase n-SOINNs are fed with more data, however n-SOINN algorithm's garbage collector discards irrelevant nodes while preserving the previously gained knowledge, making the system efficient.

By only feeding the system with failed predictions not only we achieve incremental learning with promising results but the framework is also resources efficient, meaning that - instead of feeding it the complete dataset, in our experimentation the SOINNs kept only a fraction of the information the whole dataset had, making the framework a very good candidate for a scaling industrial system. To build the initial model the framework not only trains the model but it also verifies each record whether it is accurate or not in order to perform the next cycle of update when it is requested to do so. In an industrial setting, the model, after the initial learning phase will be able to predict classes immediately by separating the prediction with the verification as depicted in **Fig. 16**.

Although, the framework's updating time grows as the update data input grows, as seen in **Fig. 19**, the framework's update and live mode could either work in parallel simultaneously or the update mode could switch when the framework's live phase is idle - when no incoming data are present to predict. The next contribution to the framework will be to modify the framework in such a way where it will be capable to learn incrementally unknown attacks automatically with an added module designed just for that purpose. Additionally to the detection of unknown attacks inherently the framework has now, a new module will automatically add these new type of attacks as new classes to the framework which

will learn unseen attacks in an incremental - "online" manner as well.

Experimentation and evaluation was performed with the NSL-KDD dataset, which is an improved version of well-known KDD'99 dataset, a very good candidate dataset to evaluate the performance of any IDS and thus our framework. Although it is stated that the NSL-KDD dataset still suffers from some of the problems discussed in Chapter 3 and may not be seen as representative for applications of today's world, the author believes it is suited for the objective of this thesis because it is an effective benchmark dataset to detect network intrusions and it serves the purpose of the framework which is to show proof-of-concept of an online incremental-learning intrusion detection system. Additionally, the experimental results showed that using only the numeric types from the forty one attributes of the dataset the system is more stable and more accurate.

Another future contribution to the framework will be the use of a different more recent publicly available dataset that would fit the purpose of this thesis and a dataset that will be created in a lab to verify these thesis results.

The results of the experimentation process indicate that the framework not only can achieve incremental learning in an online setting with high accurate predictions, the results also show that as more data become available and the model process them, the more accurate the model will be. The chart in **Fig. 18** shows that the model becomes more accurate with more samples processed and in a networked environment, data are abundant and available as soon as the framework is in place, which implies that the proposed model, in time and as input data gets processed, could reach 100% accuracy.

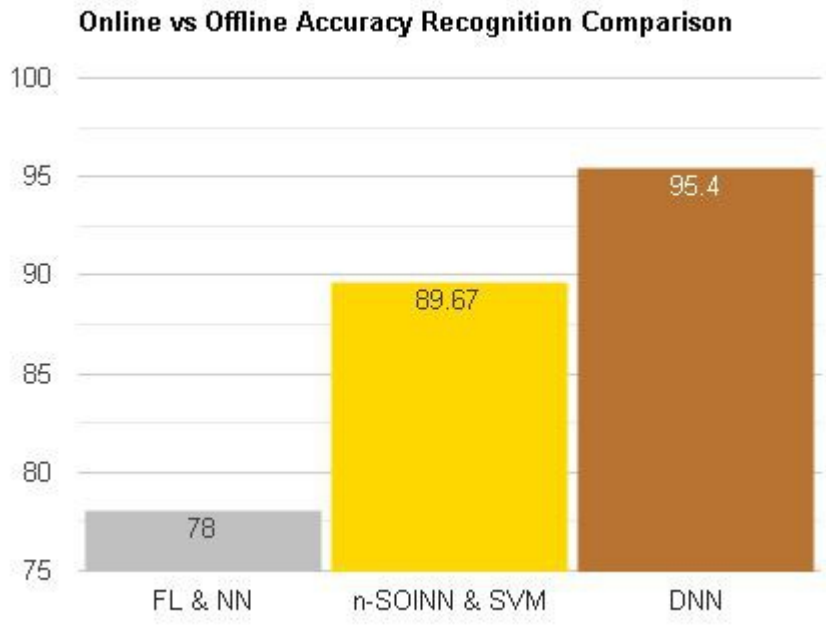


Figure 21: Online vs Offline Method Chart

To compare the accuracy results with other proposed algorithms and research work in order to evaluate this thesis proposed framework accurately, similar characteristics among the researchers work should be shared. The scope - network intrusion detection system, approach - behavioral, method - neural networks, dataset - NSL-KDD or KDD99 and operational results - accuracy - recognition on a 5 five class problem, should at least be comparable. However, beyond those shared characteristics this thesis proposed framework novelty, is based on the online incremental learning technique. According to this thesis author knowledge, there is no other model that utilizes online incremental learning methodologies against the NSL-KDD dataset for the purposes of Intrusion Detection Systems. In spite of the fact that no other model that utilizes online incremental learning for network intrusion detection system, the following comparisons was made to evaluate the accuracy - recognition results of this framework: Muna and Mehrotra proposed a network intrusion detection system based on Fuzzy logic - neural network hybrid model that evaluated on the KDD-99 dataset (Muna and Mehrotra, 2010 285 - 294), Abuadlla, Yousef, et al. proposed a network intrusion detection system based on a 2 layer neural network (Abuadlla, Yousef, et al., 2014 : 601 - 622) both of these models were evaluated on the KDD-99 or its derivative.

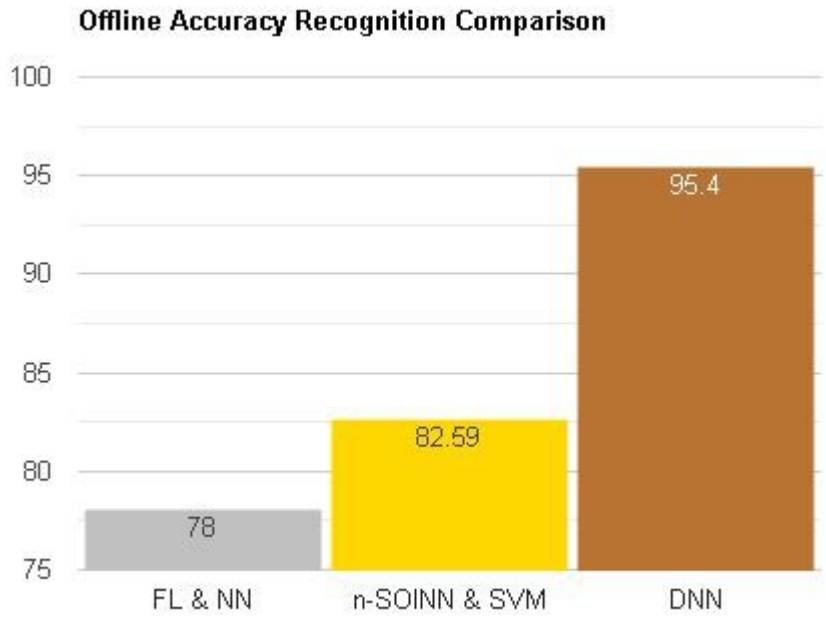


Figure 22: Online vs Offline Method Chart

The performance comparison chart as seen in **Fig. 21** compares this thesis framework online accuracy

result with the other 2 offline accuracy results in the 5 multi-class problem. The fact that n-SOINN-SVM incremental learning accuracy results performed better than Muna's and Mehrotra's proposed model and it is comparable to Abuadlla, Yousef, et al. proposed model, shows that this thesis proposed model not only proves proof-of-concept which it could incrementally improved its detection capabilities over input network data, a novelty that according to this thesis author knowledge no other model is offering, but it is also comparable and performs well against its selection of offline counterparts. Similarly, the results as seen in **Fig. 22** which compares this thesis framework offline accuracy result with the other 2 offline accuracy results in the 5 multi-class problem again it shows that n-SOINN-SVM is also comparable and performs well against its selection of offline counterparts.

Incremental learning algorithms allows the classifier to refine and improve its capabilities over time (input data) in contrast to an offline or batch learning algorithm where the classifier is assumed to be exposed to the input data once - in a batch. Network data dynamically change over time and applying static learned models degrades the detection performance significantly over time, making an offline algorithm not suitable for a network intrusion detection system.

The initial learning was performed with a relative small sample of data compared to the dataset. If for example the proposed framework was implemented in a commercial application as a cloud service, the initial learning, in time and with client applications increasing, could be a continuous updated accumulated knowledge with continuously better and more accurate initial results. The application will then update incrementally and improve its detection capabilities based on the data available to that specific network.

The example above its just one application implementation that the proposed framework could be of a valuable addition to the computer and network security industry, the proposed framework could provide the basis for an industrial intelligent network intrusion detection system.

Chapter 7
Conclusion

In summary, the proposed framework could help build an industrial Network Intrusion Protection System where it could incrementally improved its detection capabilities over input network data with the objective to protect devices - nodes connected to that network.

This paper proposes a novel Network Intrusion Prevention System based on an incremental machine learning framework where it achieves high accurate results comparable to most of the offline neural network-based NIDS.

The evaluation results shows that the proposed framework can achieve on-line updated incremental learning in a fast and efficient manner making it suitable for scaling applications.

The framework's ability to adapt to the dynamic profile nature of network data, its life-long learning capabilities and the use of minimal resources with a small sample data initialization, indicate a promising prospect for further development towards a future industrial application.

An intelligent Network Intrusion Protection System could be an appropriate solution for Internet of Things devices as well, since IoT connected devices have limited hardware resources and the rate of IoT devices connected to the internet is growing exponentially.

These are exciting times for Artificial Intelligence and its potential in all areas of human life is enormous, it could help security professionals advance the tools in their disposal to defend and protect organizations resources and technology users alike. The author believes that this postgraduate thesis will contribute to the research of artificial intelligence applied to researchers in academia and the network computer security.

Appendix A
The Code

The code was written in Python programming language.

A.1 Preparation of Data

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy import array
from numpy import isnan,isinf
import joblib
from pathlib import Path
import os
from sklearn.model_selection import train_test_split

global columns_names
global nominal_inx
global nominal_inx_labels
global binary_inx
global non_numeric_inx
global non_numeric_inx_labels
global labels
global attack_dict
global dir_path
global path


dir_path = os.path.dirname(os.path.realpath(__file__))
path = Path(dir_path)

columns_names = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
```

```
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","labels","unknown"]

nominal_inx = ['protocol_type','service','flag', 'labels']
nominal_inx_labels = ['protocol_type','service','flag']
binary_inx =
['land','logged_in','root_shell','su_attempted','is_host_login','is_guest_login','src_bytes'
, 'dst_bytes']
non_numeric_inx = nominal_inx + binary_inx
non_numeric_inx_labels = nominal_inx_labels + binary_inx
labels = ['labels']

attack_dict = {
    'normal': 'normal',

    'back': 'DoS',
    'land': 'DoS',
    'neptune': 'DoS',
    'pod': 'DoS',
    'smurf': 'DoS',
    'teardrop': 'DoS',
    'mailbomb': 'DoS',
    'apache2': 'DoS',
    'processtable': 'DoS',
    'udpstorm': 'DoS',

    'ipsweep': 'Probe',
    'nmap': 'Probe',
    'portsweep': 'Probe',
    'satan': 'Probe',
    'mscan': 'Probe',
    'saint': 'Probe',

    'ftp_write': 'R2L',
    'guess_passwd': 'R2L',
    'imap': 'R2L',
    'multihop': 'R2L',
    'phf': 'R2L',
    'spy': 'R2L',
    'warezclient': 'R2L',
    'warezmaster': 'R2L',
    'sendmail': 'R2L',
    'named': 'R2L',
    'snmpgetattack': 'R2L',
    'snmpguess': 'R2L',
    'xlock': 'R2L',
```

```
  'xsnoop': 'R2L',
  'worm': 'R2L',

  'buffer_overflow': 'U2R',
  'loadmodule': 'U2R',
  'perl': 'U2R',
  'rootkit': 'U2R',
  'httptunnel': 'U2R',
  'ps': 'U2R',
  'sqlattack': 'U2R',
  'xterm': 'U2R'
}

def train_dataset_20():
        df_train = pd.read_csv(path / 'raw_data' / 'kdd20.txt', sep=',',header=None)
        df_train.columns = columns_names
        train_item_list = set(df_train.columns)
        train_item_list = [e for e in train_item_list if e not in non_numeric_inx]
        train_item_list = list(train_item_list)
        train_df = df_train[train_item_list]
        train_df = train_df.copy()
        train_df = train_df.drop(columns='num_outbound_cmds')
        train_labels_list = set(df_train.columns)
        train_labels_list = [e for e in train_labels_list if e in labels]
        train_labels_list = list(train_labels_list)
        df_train_labels = df_train[train_labels_list]
        df_train_labels = df_train_labels.copy()
        df_train_labels['labels'] = df_train_labels['labels'].apply(lambda v:
attack_dict[v])
        df_train_labels['labels'] = df_train_labels['labels'].apply(lambda i: 0 if i
=='normal' else( 1 if i =='U2R' else( 2 if i =='R2L' else( 3 if i =='DoS' else 4))))
        return train_df, df_train_labels



def train_dataset():
        df_train = pd.read_csv(path / 'raw_data' / 'kdd_train.txt',
sep=',',header=None)
        df_train.columns = columns_names
        train_item_list = set(df_train.columns)
        train_item_list = [e for e in train_item_list if e not in non_numeric_inx]
        train_item_list = list(train_item_list)
        train_df = df_train[train_item_list]
        train_df = train_df.copy()
        train_df = train_df.drop(columns='num_outbound_cmds')
        train_labels_list = set(df_train.columns)
        train_labels_list = [e for e in train_labels_list if e in labels]
        train_labels_list = list(train_labels_list)
        df_train_labels = df_train[train_labels_list]
        df_train_labels = df_train_labels.copy()
```

```
        df_train_labels['labels'] = df_train_labels['labels'].apply(lambda v:
attack_dict[v])
        df_train_labels['labels'] = df_train_labels['labels'].apply(lambda i: 0 if i
=='normal' else( 1 if i =='U2R' else( 2 if i =='R2L' else( 3 if i =='DoS' else 4))))
        return train_df, df_train_labels

def split_full_train_dataset():
        full_train_df, full_df_train_labels = full_train_dataset()
        train_df, update_df, df_train_labels, df_update_labels =
train_test_split(full_train_df, full_df_train_labels, test_size=0.4, random_state=0)
        return train_df, df_train_labels, update_df, df_update_labels

def split_train_dataset():
        train_df, df_train_labels, update_df, df_update_labels =
split_full_train_dataset()
        return train_df, df_train_labels

def update_dataset():
        train_df, df_train_labels, update_df, df_update_labels =
split_full_train_dataset()
        return update_df, df_update_labels


def train_dataset_with_labels():
        df_train = pd.read_csv(path / 'raw_data' / 'kdd_train.txt',
sep=',',header=None)
        df_train.columns = columns_names
        train_item_list_labels = set(df_train.columns)
        train_item_list_labels = [e for e in train_item_list_labels if e not in
non_numeric_inx_labels]
        train_item_list_labels = list(train_item_list_labels)
        train_df_with_labels = df_train[train_item_list_labels]
        train_df_with_labels = train_df_with_labels.copy()
        train_df_with_labels =
train_df_with_labels.drop(columns='num_outbound_cmds')
        train_df_with_labels['labels'] = train_df_with_labels['labels'].apply(lambda v:
attack_dict[v])
        train_df_with_labels['labels'] = train_df_with_labels['labels'].apply(lambda i:
0 if i =='normal' else( 1 if i =='U2R' else( 2 if i =='R2L' else( 3 if i =='DoS' else 4))))
        return train_df_with_labels


def train_dataset_20_with_labels():
        df_train = pd.read_csv(path / 'raw_data' / 'kdd20.txt', sep=',',header=None)
        df_train.columns = columns_names
        train_item_list_labels = set(df_train.columns)
        train_item_list_labels = [e for e in train_item_list_labels if e not in
non_numeric_inx_labels]
        train_item_list_labels = list(train_item_list_labels)
        train_df_with_labels = df_train[train_item_list_labels]
```

```
        train_df_with_labels = train_df_with_labels.copy()
        train_df_with_labels =
train_df_with_labels.drop(columns='num_outbound_cmds')
        train_df_with_labels['labels'] = train_df_with_labels['labels'].apply(lambda v:
attack_dict[v])
        train_df_with_labels['labels'] = train_df_with_labels['labels'].apply(lambda i:
0 if i =='normal' else( 1 if i =='U2R' else( 2 if i =='R2L' else( 3 if i =='DoS' else 4))))
        return train_df_with_labels


def test_dataset():
        df_test = pd.read_csv(path / 'raw_data' / 'kdd_test.txt', sep=',',header=None)
        df_test.columns = columns_names
        test_item_list = set(df_test.columns)
        test_item_list = [e for e in test_item_list if e not in non_numeric_inx]
        test_item_list = list(test_item_list)
        test_df = df_test[test_item_list]
        test_df = test_df.copy()
        test_df = test_df.drop(columns='num_outbound_cmds')
        test_labels_list = set(df_test.columns)
        test_labels_list = [e for e in test_labels_list if e in labels]
        test_labels_list = list(test_labels_list)
        df_test_labels = df_test[test_labels_list]
        df_test_labels = df_test_labels.copy()
        df_test_labels['labels'] = df_test_labels['labels'].apply(lambda v:
attack_dict[v])
        df_test_labels['labels'] = df_test_labels['labels'].apply(lambda i: 0 if i
=='normal' else( 1 if i =='U2R' else( 2 if i =='R2L' else( 3 if i =='DoS' else 4))))
        return test_df, df_test_labels

def test_dataset_with_labels():
        df_test = pd.read_csv(path / 'raw_data' / 'kdd_test.txt', sep=',',header=None)
        df_test.columns = columns_names
        test_item_list_labels = set(df_test.columns)
        test_item_list_labels = [e for e in test_item_list_labels if e not in
non_numeric_inx_labels]
        test_item_list_labels = list(test_item_list_labels)
        test_df_with_labels = df_test[test_item_list_labels]
        test_df_with_labels = test_df_with_labels.copy()
        test_df_with_labels =
test_df_with_labels.drop(columns='num_outbound_cmds')
        test_df_with_labels['labels'] = test_df_with_labels['labels'].apply(lambda v:
attack_dict[v])
        test_df_with_labels['labels'] = test_df_with_labels['labels'].apply(lambda i: 0
if i =='normal' else( 1 if i =='U2R' else( 2 if i =='R2L' else( 3 if i =='DoS' else 4))))
        return test_df_with_labels


def soinn_full_train_data():
        train_df, df_train_labels = train_dataset()
```

```
        train_df = train_df.as_matrix(columns=None)
        train_df = train_df.astype('float')
        df_train_labels = df_train_labels.to_records(index=False)
        df_train_labels = list(df_train_labels.astype('float').view('float'))
        return train_df, df_train_labels


def soinn_train_data(attack):

        if attack == 'normal':
                train_df_with_labels = test_dataset_with_labels()
                #train_df, df_train_labels = train_dataset_20()
                train_df_normal =
train_df_with_labels.loc[train_df_with_labels['labels'] == 0]
                #df_train_labels_normal = df_train_labels.loc[df_train_labels['labels']
== 0]
                train_df_normal = train_df_normal.drop(columns='labels')
                train_df_normal = train_df_normal.as_matrix(columns=None)
                train_df_normal = train_df_normal.astype('float')
                #df_train_labels_normal =
df_train_labels_normal.to_records(index=False)
                #df_train_labels_normal =
list(df_train_labels_normal.astype('float').view('float'))

        elif attack == 'U2R':
                train_df_with_labels = test_dataset_with_labels()
                #train_df, df_train_labels = train_dataset_20()
                train_df_U2R = train_df_with_labels.loc[train_df_with_labels['labels']
== 1]
                #df_train_labels_U2R = df_train_labels.loc[df_train_labels['labels'] ==
1]
                train_df_U2R = train_df_U2R.drop(columns='labels')
                train_df_U2R = train_df_U2R.as_matrix(columns=None)
                train_df_U2R = train_df_U2R.astype('float')
                #df_train_labels_U2R = df_train_labels_U2R.to_records(index=False)
                #df_train_labels_U2R =
list(df_train_labels_U2R.astype('float').view('float'))

        elif attack == 'R2L':
                train_df_with_labels = test_dataset_with_labels()
                #train_df, df_train_labels = train_dataset_20()
                train_df_R2L = train_df_with_labels.loc[train_df_with_labels['labels']
== 2]
                #df_train_labels_R2L = df_train_labels.loc[df_train_labels['labels'] ==
2]
                train_df_R2L = train_df_R2L.drop(columns='labels')
                train_df_R2L = train_df_R2L.as_matrix(columns=None)
                train_df_R2L = train_df_R2L.astype('float')
                #df_train_labels_R2L = df_train_labels_R2L.to_records(index=False)
                #df_train_labels_R2L =
```

```
list(df_train_labels_R2L.astype('float').view('float'))

        elif attack == 'DoS':
                train_df_with_labels = test_dataset_with_labels()
                #train_df, df_train_labels = train_dataset_20()
                train_df_DoS = train_df_with_labels.loc[train_df_with_labels['labels']
== 3]
                #df_train_labels_DoS = df_train_labels.loc[df_train_labels['labels'] ==
3]
                train_df_DoS = train_df_DoS.drop(columns='labels')
                train_df_DoS = train_df_DoS.as_matrix(columns=None)
                train_df_DoS = train_df_DoS.astype('float')
                #df_train_labels_DoS = df_train_labels_DoS.to_records(index=False)
                #df_train_labels_DoS =
list(df_train_labels_DoS.astype('float').view('float'))

        else:
                train_df_with_labels = test_dataset_with_labels()
                #train_df, df_train_labels = train_dataset_20()
                train_df_Probe =
train_df_with_labels.loc[train_df_with_labels['labels'] == 4]
                #df_train_labels_Probe = df_train_labels.loc[df_train_labels['labels']
== 4]
                train_df_Probe = train_df_Probe.drop(columns='labels')
                train_df_Probe = train_df_Probe.as_matrix(columns=None)
                train_df_Probe = train_df_Probe.astype('float')
                #df_train_labels_Probe =
df_train_labels_Probe.to_records(index=False)
                #df_train_labels_Probe =
list(df_train_labels_Probe.astype('float').view('float'))

        train_df_data = eval('train_df_{0}'.format(attack))
        #df_train_labels_data = eval('df_train_labels_{0}'.format(attack))

        return train_df_data#, df_train_labels_data

if __name__ == '__main__':
        #train, labels = train_dataset()
        #train_u, labels_u = update_dataset()
        #train_t, labels_t = test_dataset()
        #print train.head
        #print train_u.head
        #print train_t.head
        #test_df, df_test_labels = test_dataset()
        #print test_df.head
        split_full_train_dataset()
```

A.2 n_SOINN

```
import numpy as np
from scipy.sparse import dok_matrix
import util


class Soinn(object):

    def __init__(self,n_soinn=2, delete_node_period=300, max_edge_age=50):

        self.n_soinn = n_soinn
        self.delete_node_period = delete_node_period
        self.max_edge_age = max_edge_age
        self.min_degree = 1
        self.num_signal = 0
        self.nodes = np.array([], dtype=np.float64)
        self.winning_times = []
        self.winning_times1st = []
        self.winning_times2nd = []
        self.adjacent_mat = dok_matrix((0, 0), dtype=np.float64)
        self.n2nd = 0
        self.n1st = 0

    def input_signal(self, signal, learning=True):

        self.__check_signal(signal)
        self.num_signal += 1

        if self.nodes.shape[0] < 3:
            self.__add_node(signal)
            return

        winner, dists = self.__find_nearest_nodes(2, signal)

        if not learning:
            return winner

        sim_thresholds = self.calculate_similarity_thresholds(winner)
        if (dists[0] > sim_thresholds[0] and self.n2nd > self.n_soinn) or (dists[1] >
sim_thresholds[1] and self.n2nd > self.n_soinn):
            self.__add_node(signal)
```

```
        else:
            self.__add_edge(winner)
            self.__increment_edge_ages(winner[1])
            winner[1] = self.__delete_old_edges(winner[1])
            #print ('winner_beofre = ',winner)
            self.__update_winner(winner[0], winner[1], signal)
            #print ('winner_after = ',winner)
            #print ('self.n_soinn = ',self.n_soinn)
            #print ('self.n2nd = ',self.n2nd)
            #print ('self.n1st = ',self.n1st)
            #exit()
            self.__update_adjacent_nodes(winner[1], signal)

        #if self.num_signal % self.delete_node_period == 0:
         #   self.__delete_noise_nodes()
        return winner

    def __check_signal(self, signal):

        if not(isinstance(signal, np.ndarray)):
            raise TypeError()
        if len(signal.shape) != 1:
            raise TypeError()
        if not(hasattr(self, 'dim')):
            self.dim = signal.shape[0]
        else:
            if signal.shape[0] != self.dim:
                raise TypeError()

    def __add_node(self, signal):
        n = self.nodes.shape[0]
        self.nodes.resize((n + 1, self.dim))
        self.nodes[-1, :] = signal
        self.winning_times1st.append(1)
        self.winning_times2nd.append(1)
        self.winning_times.append(1)
        self.adjacent_mat.resize((n + 1, n + 1))

    def __find_nearest_nodes(self, num, signal, mahar=True):
        #if mahar: return self.__find_nearest_nodes_by_mahar(num, signal)
        n = self.nodes.shape[0]
        indexes = [0.0] * num
        sq_dists = [0.0] * num
        D = util.calc_distance(self.nodes, np.asarray([signal] * n))
        for i in range(num):
            indexes[i] = np.nanargmin(D)
            sq_dists[i] = D[indexes[i]]
            D[indexes[i]] = float('nan')
        return indexes, sq_dists
```

```
  def __find_nearest_nodes_by_mahar(self, num, signal):
    indexes, sq_dists = util.calc_mahalanobis(self.nodes, signal, 2)
    return indexes, sq_dists

  def calculate_similarity_thresholds(self, node_indexes):
    sim_thresholds = []
    for i in node_indexes:
      pals = self.adjacent_mat[i, :]
      if len(pals) == 0:
        idx, sq_dists = self.__find_nearest_nodes(2, self.nodes[i, :])
        sim_thresholds.append(sq_dists[1])
      else:
        pal_indexes = []
        for k in pals.keys():
          pal_indexes.append(k[1])
        sq_dists = util.calc_distance(self.nodes[pal_indexes],
np.asarray([self.nodes[i]] * len(pal_indexes)))
        sim_thresholds.append(np.max(sq_dists))
    return sim_thresholds

  def __add_edge(self, node_indexes):
    self.__set_edge_weight(node_indexes, 1)

  def __increment_edge_ages(self, winner_index):
    for k, v in self.adjacent_mat[winner_index, :].items():
      self.__set_edge_weight((winner_index, k[1]), v + 1)

  def __delete_old_edges(self, winner_index):
    candidates = []
    for k, v in self.adjacent_mat[winner_index, :].items():
      if v > self.max_edge_age + 1:
        candidates.append(k[1])
        self.__set_edge_weight((winner_index, k[1]), 0)
    delete_indexes = []
    for i in candidates:
      if len(self.adjacent_mat[i, :]) == 0:
        delete_indexes.append(i)
    self.__delete_nodes(delete_indexes)
    delete_count = sum([1 if i < winner_index else 0 for i in delete_indexes])
    return winner_index - delete_count

  def __set_edge_weight(self, index, weight):
    self.adjacent_mat[index[0], index[1]] = weight
    self.adjacent_mat[index[1], index[0]] = weight

  def __update_winner(self, winner_index0, winner_index, signal):
    self.winning_times[winner_index] += 1
    w = self.nodes[winner_index]
    self.nodes[winner_index] = w + (signal - w)/self.winning_times[winner_index]
    self.winning_times1st[winner_index0] += 1
```

```
      #self.winning_times1st.append(1)
      #self.winning_times2nd[winner_index] += 1
      self.n1st = self.winning_times1st[winner_index0]
      if self.n1st > self.n_soinn:
         self.winning_times2nd[winner_index] += 1
         #self.winning_times2nd.append(1)
      self.n2nd = self.winning_times2nd[winner_index]

   def __update_adjacent_nodes(self, winner_index, signal):
      pals = self.adjacent_mat[winner_index]
      for k in pals.keys():
         i = k[1]
         w = self.nodes[i]
         self.nodes[i] = w + (signal - w)/(100 * self.winning_times[i])

   def __delete_nodes(self, indexes):
      n = len(self.winning_times)
      self.nodes = np.delete(self.nodes, indexes, 0)
      remained_indexes = list(set([i for i in range(n)]) - set(indexes))
      self.winning_times = [self.winning_times[i] for i in remained_indexes]
      #_old_ver_adjacent_mat = self.adjacent_mat[np.ix_(remained_indexes,
remained_indexes)]
      self.__update_adjacent_mat(indexes, n, len(remained_indexes))
      #assert (_old_ver_adjacent_mat.toarray() == self.adjacent_mat.toarray()).all()

   def __update_adjacent_mat(self, indexes, prev_n, next_n):
      while indexes:
         next_adjacent_mat = dok_matrix((prev_n, prev_n))
         for key1, key2 in self.adjacent_mat.keys():
            if key1 == indexes[0] or key2 == indexes[0]:
               continue
            if key1 > indexes[0]:
               new_key1 = key1 - 1
            else:
               new_key1 = key1
            if key2 > indexes[0]:
               new_key2 = key2 - 1
            else:
               new_key2 = key2

            next_adjacent_mat[new_key1, new_key2] = super(dok_matrix,
self.adjacent_mat).__getitem__((key1, key2))
         self.adjacent_mat = next_adjacent_mat.copy()
         indexes = [i-1 for i in indexes]
         indexes.pop(0)
      self.adjacent_mat.resize((next_n, next_n))

   def __delete_nodes2(self, indexes):
      n = len(self.winning_times)
      self.nodes = np.delete(self.nodes, indexes, 0)
```

```
        remained_indexes = list(set([i for i in range(n)]) - set(indexes))
        self.winning_times = [self.winning_times[i] for i in remained_indexes]
        self.adjacent_mat = self.adjacent_mat[np.ix_(remained_indexes,
remained_indexes)]

    def __delete_noise_nodes(self):
        n = len(self.winning_times)
        noise_indexes = []
        for i in range(n):
            if len(self.adjacent_mat[i, :]) < self.min_degree:
                noise_indexes.append(i)
        if noise_indexes:
            self.__delete_nodes(noise_indexes)

    def print_info(self):
        print('Total Nodes: {0}'.format(len(self.nodes)))

    def save(self, dumpfile='soinn.dump'):
        import joblib
        joblib.dump(self, dumpfile, compress=True, protocol=0)
```

A.2.1 n_SOINN's Standard Euclidean Distance

```
import numpy as np
import scipy
from scipy.spatial import distance
from scipy.spatial.distance import pdist

def calc_distance(x, y):

    V = np.var(x,axis=0)
    V[V == 0] = 1
    for z,w in zip(x,y):
        dist_list.append(distance.seuclidean(z,w,V))
    return np.array(dist_list)
```

A.3 Create n_SOINNs Pairs

```
from n_soinn import Soinn
from collections import defaultdict
from pathlib import Path
import os


def learning(soinn, x_train):
    for i, data_x in enumerate(x_train):
        if i % 1000 == 0:
            print('Processing {0}th data.'.format(i))
        soinn.input_signal(data_x)

def print_soinn_info(soinn_nodes):
    for i, node in enumerate(soinn_nodes):
        print('node[{0}] values: {1}'.format(i, node))

def create_soinns(data, n_soinn_number, category, delete_period, max_age ):
    train_data = data
    n_soinn = n_soinn_number
    delete_node_period = delete_period
    max_edge_age = max_age
    category_label = category
    dir_path = os.path.dirname(os.path.realpath(__file__))
    path = Path(dir_path)
    dumpfile = path / 'dump' / 'SOINNs' /
'soinn{0}_{1}.dump'.format(n_soinn,category_label)
    print('New SOINN is created.')
    soinn_i = Soinn(n_soinn,
delete_node_period=delete_node_period,max_edge_age=max_edge_age)
    learning(soinn_i, train_data)
    soinn_i.print_info()
    soinn_i.save(dumpfile)

if __name__ == '__main__':

            data,labels = soinn_train_data('normal')
            n_soinn_number = 2
            category = 'normal'
            delete_period = 100
            max_age = 30
            create_soinns(data, labels, n_soinn_number, category, delete_period,
max_age)
```

A.4 Prepare Binary SVMs Input Data

```
from sklearn.svm import SVC
#from sklearn.svm import LinearSVC
import pandas as pd
import joblib
from pathlib import Path
from sklearn.model_selection import GridSearchCV
import numpy as np
import os
global path
#import time
#start_time = time.time()

dir_path = os.path.dirname(os.path.realpath(__file__))
path = Path(dir_path)

global normal_negatives
global Probe_negatives
global R2L_negatives
global U2R_negatives
global DoS_negatives


normal_negatives = ['DoS', 'U2R', 'R2L', 'Probe']
Probe_negatives = ['DoS', 'U2R', 'R2L', 'normal']
R2L_negatives = ['DoS', 'U2R', 'Probe', 'normal']
U2R_negatives = ['DoS', 'R2L', 'Probe', 'normal']
DoS_negatives = ['U2R', 'R2L', 'Probe', 'normal']


def svm_train_data(category_label):

        normal_negatives_soinns = []
        Probe_negatives_soinns = []
        R2L_negatives_soinns = []
        U2R_negatives_soinns = []
        DoS_negatives_soinns = []

        lo_soinn_dumpfile = path / 'dump' / 'SOINNs' /
'soinn2_{0}.dump'.format(category_label)
        hi_soinn_dumpfile = path / 'dump' / 'SOINNs' /
'soinn100_{0}.dump'.format(category_label)
        df_pos_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_pos.dump'.format(category_label)
        df_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(category_label)
        df_svm_dumpfile = path / 'dump' / 'svms_datasets' /
```

```
'df_svm_{0}.dump'.format(category_label)
        df_svm_labels_dumpfile = path / 'dump' / 'svms_datasets' /
'df_svm_{0}_labels.dump'.format(category_label)


        if category_label == 'normal':

                try:
                        df_svm_normal = joblib.load(df_svm_dumpfile)
                        df_svm_normal_labels = joblib.load(df_svm_labels_dumpfile)
                except:

                        try:

                                df_normal_pos = joblib.load(df_pos_dumpfile)
                                for normal_negative in normal_negatives:
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(normal_negative)
                                        normal_df_neg =
joblib.load(df_soinn_neg_dumpfile)

        normal_negatives_soinns.append(normal_df_neg)

                        except:
                                lo_soinn_normal = joblib.load(lo_soinn_dumpfile)
                                pos_values_normal = []
                                for node in lo_soinn_normal.nodes:
                                        pos_values_normal.append(list(node))
                                df_normal_pos = pd.DataFrame(pos_values_normal)
                                df_normal_pos.loc[:,'labels'] = 1
                                joblib.dump(df_normal_pos, df_pos_dumpfile)

                                for normal_negative in normal_negatives:
                                        hi_neg_soinn_dumpfile = path / 'dump' /
'SOINNs' / 'soinn100_{0}.dump'.format(normal_negative)
                                        hi_soinn = joblib.load(hi_neg_soinn_dumpfile)
                                        neg_values = []
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(normal_negative)
                                        for node in hi_soinn.nodes:
                                                neg_values.append(list(node))
                                        normal_df_neg = pd.DataFrame(neg_values)
                                        normal_df_neg.loc[:,'labels'] = 0

        normal_negatives_soinns.append(normal_df_neg)
                                        joblib.dump(normal_df_neg,
df_soinn_neg_dumpfile)

                        df_svm_normal =
df_normal_pos.append(normal_negatives_soinns)
```

```
                        df_svm_normal_labels = df_svm_normal['labels']
                        df_svm_normal = df_svm_normal.drop(columns='labels')
                        joblib.dump(df_svm_normal, df_svm_dumpfile)
                        joblib.dump(df_svm_normal_labels, df_svm_labels_dumpfile)

        elif category_label == 'Probe':

                try:
                        df_svm_Probe = joblib.load(df_svm_dumpfile)
                        df_svm_Probe_labels = joblib.load(df_svm_labels_dumpfile)
                except:

                        try:

                                df_Probe_pos = joblib.load(df_pos_dumpfile)
                                for Probe_negative in Probe_negatives:
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(Probe_negative)
                                        Probe_df_neg =
joblib.load(df_soinn_neg_dumpfile)
                                        Probe_negatives_soinns.append(Probe_df_neg)

                        except:
                                lo_soinn_Probe = joblib.load(lo_soinn_dumpfile)
                                pos_values_Probe = []
                                for node in lo_soinn_Probe.nodes:
                                        pos_values_Probe.append(list(node))
                                df_Probe_pos = pd.DataFrame(pos_values_Probe)
                                df_Probe_pos.loc[:,'labels'] = 1
                                joblib.dump(df_Probe_pos, df_pos_dumpfile)

                                for Probe_negative in Probe_negatives:
                                        hi_neg_soinn_dumpfile = path / 'dump' /
'SOINNs' / 'soinn100_{0}.dump'.format(Probe_negative)
                                        hi_soinn = joblib.load(hi_neg_soinn_dumpfile)
                                        neg_values = []
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(Probe_negative)
                                        for node in hi_soinn.nodes:
                                                neg_values.append(list(node))
                                        Probe_df_neg = pd.DataFrame(neg_values)
                                        Probe_df_neg.loc[:,'labels'] = 0
                                        Probe_negatives_soinns.append(Probe_df_neg)
                                        joblib.dump(Probe_df_neg,
df_soinn_neg_dumpfile)

                        df_svm_Probe =
df_Probe_pos.append(Probe_negatives_soinns)
                        df_svm_Probe_labels = df_svm_Probe['labels']
                        df_svm_Probe = df_svm_Probe.drop(columns='labels')
```

```
                joblib.dump(df_svm_Probe, df_svm_dumpfile)
                joblib.dump(df_svm_Probe_labels, df_svm_labels_dumpfile)

        elif category_label == 'R2L':

                try:
                        df_svm_R2L = joblib.load(df_svm_dumpfile)
                        df_svm_R2L_labels = joblib.load(df_svm_labels_dumpfile)
                except:

                        try:

                                df_R2L_pos = joblib.load(df_pos_dumpfile)
                                for R2L_negative in R2L_negatives:
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(R2L_negative)
                                        R2L_df_neg =
joblib.load(df_soinn_neg_dumpfile)
                                        R2L_negatives_soinns.append(R2L_df_neg)

                        except:
                                lo_soinn_R2L = joblib.load(lo_soinn_dumpfile)
                                pos_values_R2L = []
                                for node in lo_soinn_R2L.nodes:
                                        pos_values_R2L.append(list(node))
                                df_R2L_pos = pd.DataFrame(pos_values_R2L)
                                df_R2L_pos.loc[:,'labels'] = 1
                                joblib.dump(df_R2L_pos, df_pos_dumpfile)

                                for R2L_negative in R2L_negatives:
                                        hi_neg_soinn_dumpfile = path / 'dump' /
'SOINNs' / 'soinn100_{0}.dump'.format(R2L_negative)
                                        hi_soinn = joblib.load(hi_neg_soinn_dumpfile)
                                        neg_values = []
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(R2L_negative)
                                        for node in hi_soinn.nodes:
                                                neg_values.append(list(node))
                                        R2L_df_neg = pd.DataFrame(neg_values)
                                        R2L_df_neg.loc[:,'labels'] = 0
                                        R2L_negatives_soinns.append(R2L_df_neg)
                                        joblib.dump(R2L_df_neg,
df_soinn_neg_dumpfile)

                        df_svm_R2L = df_R2L_pos.append(R2L_negatives_soinns)
                        df_svm_R2L_labels = df_svm_R2L['labels']
                        df_svm_R2L = df_svm_R2L.drop(columns='labels')
                        joblib.dump(df_svm_R2L, df_svm_dumpfile)
                        joblib.dump(df_svm_R2L_labels, df_svm_labels_dumpfile)
```

```
        elif category_label == 'U2R':

                try:
                        df_svm_U2R = joblib.load(df_svm_dumpfile)
                        df_svm_U2R_labels = joblib.load(df_svm_labels_dumpfile)
                except:

                        try:

                                df_U2R_pos = joblib.load(df_pos_dumpfile)
                                for U2R_negative in U2R_negatives:
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(U2R_negative)
                                        U2R_df_neg =
joblib.load(df_soinn_neg_dumpfile)
                                        U2R_negatives_soinns.append(U2R_df_neg)

                        except:
                                lo_soinn_U2R = joblib.load(lo_soinn_dumpfile)
                                pos_values_U2R = []
                                for node in lo_soinn_U2R.nodes:
                                        pos_values_U2R.append(list(node))
                                df_U2R_pos = pd.DataFrame(pos_values_U2R)
                                df_U2R_pos.loc[:,'labels'] = 1
                                joblib.dump(df_U2R_pos, df_pos_dumpfile)

                                for U2R_negative in U2R_negatives:
                                        hi_neg_soinn_dumpfile = path / 'dump' /
'SOINNs' / 'soinn100_{0}.dump'.format(U2R_negative)
                                        hi_soinn = joblib.load(hi_neg_soinn_dumpfile)
                                        neg_values = []
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(U2R_negative)
                                        for node in hi_soinn.nodes:
                                                neg_values.append(list(node))
                                        U2R_df_neg = pd.DataFrame(neg_values)
                                        U2R_df_neg.loc[:,'labels'] = 0
                                        U2R_negatives_soinns.append(U2R_df_neg)
                                        joblib.dump(U2R_df_neg,
df_soinn_neg_dumpfile)

                        df_svm_U2R = df_U2R_pos.append(U2R_negatives_soinns)
                        df_svm_U2R_labels = df_svm_U2R['labels']
                        df_svm_U2R = df_svm_U2R.drop(columns='labels')
                        joblib.dump(df_svm_U2R, df_svm_dumpfile)
                        joblib.dump(df_svm_U2R_labels, df_svm_labels_dumpfile)
```

```
        else:

                try:
                        df_svm_DoS = joblib.load(df_svm_dumpfile)
                        df_svm_DoS_labels = joblib.load(df_svm_labels_dumpfile)
                except:

                        try:

                                df_DoS_pos = joblib.load(df_pos_dumpfile)
                                for DoS_negative in DoS_negatives:
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(DoS_negative)
                                        DoS_df_neg =
joblib.load(df_soinn_neg_dumpfile)
                                        DoS_negatives_soinns.append(DoS_df_neg)

                        except:
                                lo_soinn_DoS = joblib.load(lo_soinn_dumpfile)
                                pos_values_DoS = []
                                for node in lo_soinn_DoS.nodes:
                                        pos_values_DoS.append(list(node))
                                df_DoS_pos = pd.DataFrame(pos_values_DoS)
                                df_DoS_pos.loc[:,'labels'] = 1
                                joblib.dump(df_DoS_pos, df_pos_dumpfile)

                                for DoS_negative in DoS_negatives:
                                        hi_neg_soinn_dumpfile = path / 'dump' /
'SOINNs' / 'soinn100_{0}.dump'.format(DoS_negative)
                                        hi_soinn = joblib.load(hi_neg_soinn_dumpfile)
                                        neg_values = []
                                        df_soinn_neg_dumpfile = path / 'dump' /
'Soinns_datasets' / 'df_{0}_neg.dump'.format(DoS_negative)
                                        for node in hi_soinn.nodes:
                                                neg_values.append(list(node))
                                        DoS_df_neg = pd.DataFrame(neg_values)
                                        DoS_df_neg.loc[:,'labels'] = 0
                                        DoS_negatives_soinns.append(DoS_df_neg)
                                        joblib.dump(DoS_df_neg,
df_soinn_neg_dumpfile)

                        df_svm_DoS = df_DoS_pos.append(DoS_negatives_soinns)
                        df_svm_DoS_labels = df_svm_DoS['labels']
                        df_svm_DoS = df_svm_DoS.drop(columns='labels')
                        joblib.dump(df_svm_DoS, df_svm_dumpfile)
                        joblib.dump(df_svm_DoS_labels, df_svm_labels_dumpfile)

        svm_train_data = eval('df_svm_{0}'.format(category_label))
```

```
        svm_train_labels = eval('df_svm_{0}_labels'.format(category_label))

        svm_train_data = svm_train_data.reset_index(drop=True)
        svm_train_labels = svm_train_labels.reset_index(drop=True)


        return svm_train_data, svm_train_labels
```

A.5 Prepare Multi-class SVMs Input Data

```
def multiclass_svm_datasets(category_label, updating=False):

        df_pos_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_pos.dump'.format(category_label)
        df_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(category_label)
        dumpfile = path / 'dump'/ 'multi_svms_datasets' /
'df_svm_{0}.dump'.format(category_label)
        dumpfile_labels = path / 'dump' / 'multi_svms_datasets' /
'df_svm_{0}_labels.dump'.format(category_label)

        if category_label == 'normal':

                if updating == False:

                        try:
                                df_svm_normal = joblib.load(dumpfile)
                                df_svm_normal_labels = joblib.load(dumpfile_labels)

                        except:
                                print('New df_svm_normal file pairs are created.')
                                df_normal_pos = joblib.load(df_pos_dumpfile)
                                df_normal_neg = joblib.load(df_neg_dumpfile)
                                df_normal_pos.loc[:,'labels'] = 0
                                df_normal_neg.loc[:,'labels'] = 0
                                df_svm_normal =
df_normal_pos.append(df_normal_neg)
                                df_svm_normal_labels = df_svm_normal['labels']
                                df_svm_normal =
df_svm_normal.drop(columns='labels')
                                joblib.dump(df_svm_normal, dumpfile)
                                joblib.dump(df_svm_normal_labels, dumpfile_labels)
                else:

                        print('Updating..New df_svm_normal file pairs are created.')
                        df_normal_pos = joblib.load(df_pos_dumpfile)
```

```
                df_normal_neg = joblib.load(df_neg_dumpfile)
                df_normal_pos.loc[:,'labels'] = 0
                df_normal_neg.loc[:,'labels'] = 0
                df_svm_normal = df_normal_pos.append(df_normal_neg)
                df_svm_normal_labels = df_svm_normal['labels']
                df_svm_normal = df_svm_normal.drop(columns='labels')
                joblib.dump(df_svm_normal, dumpfile)
                joblib.dump(df_svm_normal_labels, dumpfile_labels)

    elif category_label == 'Probe':

        if updating == False:

            try:
                df_svm_Probe = joblib.load(dumpfile)
                df_svm_Probe_labels = joblib.load(dumpfile_labels)

            except:
                print('New df_svm_Probe file pairs are created.')
                df_Probe_pos = joblib.load(df_pos_dumpfile)
                df_Probe_neg = joblib.load(df_neg_dumpfile)
                df_Probe_pos.loc[:,'labels'] = 4
                df_Probe_neg.loc[:,'labels'] = 4
                df_svm_Probe = df_Probe_pos.append(df_Probe_neg)
                df_svm_Probe_labels = df_svm_Probe['labels']
                df_svm_Probe = df_svm_Probe.drop(columns='labels')
                joblib.dump(df_svm_Probe, dumpfile)
                joblib.dump(df_svm_Probe_labels, dumpfile_labels)

        else:

            print('Updating...New df_svm_Probe file pairs are created.')
            df_Probe_pos = joblib.load(df_pos_dumpfile)
            df_Probe_neg = joblib.load(df_neg_dumpfile)
            df_Probe_pos.loc[:,'labels'] = 4
            df_Probe_neg.loc[:,'labels'] = 4
            df_svm_Probe = df_Probe_pos.append(df_Probe_neg)
            df_svm_Probe_labels = df_svm_Probe['labels']
            df_svm_Probe = df_svm_Probe.drop(columns='labels')
            joblib.dump(df_svm_Probe, dumpfile)
            joblib.dump(df_svm_Probe_labels, dumpfile_labels)


    elif category_label == 'DoS':

        if updating == False:

            try:
                df_svm_DoS = joblib.load(dumpfile)
                df_svm_DoS_labels = joblib.load(dumpfile_labels)
```

```
                except:
                        print('New df_svm_DoS file pairs are created.')
                        df_DoS_pos = joblib.load(df_pos_dumpfile)
                        df_DoS_neg = joblib.load(df_neg_dumpfile)
                        df_DoS_pos.loc[:,'labels'] = 3
                        df_DoS_neg.loc[:,'labels'] = 3
                        df_svm_DoS = df_DoS_pos.append(df_DoS_neg)
                        df_svm_DoS_labels = df_svm_DoS['labels']
                        df_svm_DoS = df_svm_DoS.drop(columns='labels')
                        joblib.dump(df_svm_DoS, dumpfile)
                        joblib.dump(df_svm_DoS_labels, dumpfile_labels)

        else:

                print('Updating..New df_svm_DoS file pairs are created.')
                df_DoS_pos = joblib.load(df_pos_dumpfile)
                df_DoS_neg = joblib.load(df_neg_dumpfile)
                df_DoS_pos.loc[:,'labels'] = 3
                df_DoS_neg.loc[:,'labels'] = 3
                df_svm_DoS = df_DoS_pos.append(df_DoS_neg)
                df_svm_DoS_labels = df_svm_DoS['labels']
                df_svm_DoS = df_svm_DoS.drop(columns='labels')
                joblib.dump(df_svm_DoS, dumpfile)
                joblib.dump(df_svm_DoS_labels, dumpfile_labels)


elif category_label == 'R2L':

        if updating == False:

                try:
                        df_svm_R2L = joblib.load(dumpfile)
                        df_svm_R2L_labels = joblib.load(dumpfile_labels)

                except:
                        print('New df_svm_R2L file pairs are created.')
                        df_R2L_pos = joblib.load(df_pos_dumpfile)
                        df_R2L_neg = joblib.load(df_neg_dumpfile)
                        df_R2L_pos.loc[:,'labels'] = 2
                        df_R2L_neg.loc[:,'labels'] = 2
                        df_svm_R2L = df_R2L_pos.append(df_R2L_neg)
                        df_svm_R2L_labels = df_svm_R2L['labels']
                        df_svm_R2L = df_svm_R2L.drop(columns='labels')
                        joblib.dump(df_svm_R2L, dumpfile)
                        joblib.dump(df_svm_R2L_labels, dumpfile_labels)


        else:
```

```
                print('Updating..New df_svm_R2L file pairs are created.')
                df_R2L_pos = joblib.load(df_pos_dumpfile)
                df_R2L_neg = joblib.load(df_neg_dumpfile)
                df_R2L_pos.loc[:,'labels'] = 2
                df_R2L_neg.loc[:,'labels'] = 2
                df_svm_R2L = df_R2L_pos.append(df_R2L_neg)
                df_svm_R2L_labels = df_svm_R2L['labels']
                df_svm_R2L = df_svm_R2L.drop(columns='labels')
                joblib.dump(df_svm_R2L, dumpfile)
                joblib.dump(df_svm_R2L_labels, dumpfile_labels)

    else:

        if updating == False:

            try:
                df_svm_U2R = joblib.load(dumpfile)
                df_svm_U2R_labels = joblib.load(dumpfile_labels)

            except:
                print('New df_svm_U2R file pairs are created.')
                df_U2R_pos = joblib.load(df_pos_dumpfile)
                df_U2R_neg = joblib.load(df_neg_dumpfile)
                df_U2R_pos.loc[:,'labels'] = 1
                df_U2R_neg.loc[:,'labels'] = 1
                df_svm_U2R = df_U2R_pos.append(df_U2R_neg)
                df_svm_U2R_labels = df_svm_U2R['labels']
                df_svm_U2R = df_svm_U2R.drop(columns='labels')
                joblib.dump(df_svm_U2R, dumpfile)
                joblib.dump(df_svm_U2R_labels, dumpfile_labels)

        else:

            print('Updating...New df_svm_U2R file pairs are created.')
            df_U2R_pos = joblib.load(df_pos_dumpfile)
            df_U2R_neg = joblib.load(df_neg_dumpfile)
            df_U2R_pos.loc[:,'labels'] = 1
            df_U2R_neg.loc[:,'labels'] = 1
            df_svm_U2R = df_U2R_pos.append(df_U2R_neg)
            df_svm_U2R_labels = df_svm_U2R['labels']
            df_svm_U2R = df_svm_U2R.drop(columns='labels')
            joblib.dump(df_svm_U2R, dumpfile)
            joblib.dump(df_svm_U2R_labels, dumpfile_labels)

    multi_svm_data = eval('df_svm_{0}'.format(category_label))
    multi_svm_labels = eval('df_svm_{0}_labels'.format(category_label))

    return multi_svm_data, multi_svm_labels
```

A.6 Run Binary SVM Predictions

```
def load_svms(category_label, updating=False):

        dumpfile = path / 'dump' / 'Bin_svms' /
'svm_{0}.dump'.format(category_label)

        if category_label == 'normal':

                if updating == False:
                        try:
                                svm_normal = joblib.load(dumpfile)
                                #print('normal SVM is loaded.')

                        except:
                                print('New normal SVM is created.')
                                df_svm_normal, df_svm_normal_labels =
svm_train_data('normal')
                                svm_normal = SVC(C=21.9428032327,
gamma=0.000132571136559)
                                #svm_normal = svm_svc()
                                svm_normal.fit(df_svm_normal,
df_svm_normal_labels)
                                #print 'clf.best_estimator_.C = ',
svm_normal.best_estimator_.C
                                #print 'clf.best_estimator_.gamma = ',
svm_normal.best_estimator_.gamma
                                joblib.dump(svm_normal, dumpfile)
                else:
                        print('New normal Updated SVM is created.')
                        df_svm_normal, df_svm_normal_labels =
svm_train_data('normal')
                        svm_normal = SVC(C=21.9428032327,
gamma=0.000132571136559)
                        #svm_normal = svm_svc()
                        svm_normal.fit(df_svm_normal, df_svm_normal_labels)
                        #print 'clf.best_estimator_.C = ', svm_normal.best_estimator_.C
                        #print 'clf.best_estimator_.gamma = ',
svm_normal.best_estimator_.gamma
                        joblib.dump(svm_normal, dumpfile)


        elif category_label == 'Probe':

                if updating == False:
```

```
                                try:
                                        svm_Probe = joblib.load(dumpfile)
                                        #print('Probe SVM is loaded.')

                                except:
                                        print('New Probe SVM is created.')
                                        df_svm_Probe, df_svm_Probe_labels =
svm_train_data('Probe')
                                        #svm_Probe = svm_svc()
                                        svm_Probe = SVC(C=26.9594844333,
gamma=0.0001)
                                        svm_Probe.fit(df_svm_Probe,
df_svm_Probe_labels)
                                        #print 'clf.best_estimator_.C = ',
svm_Probe.best_estimator_.C
                                        #print 'clf.best_estimator_.gamma = ',
svm_Probe.best_estimator_.gamma
                                        joblib.dump(svm_Probe, dumpfile)
                else:
                        print('New Probe Updated SVM is created.')
                        df_svm_Probe, df_svm_Probe_labels = svm_train_data('Probe')
                        #svm_Probe = svm_svc()
                        svm_Probe = SVC(C=26.9594844333, gamma=0.0001)
                        svm_Probe.fit(df_svm_Probe, df_svm_Probe_labels)
                        #print 'clf.best_estimator_.C = ', svm_Probe.best_estimator_.C
                        #print 'clf.best_estimator_.gamma = ',
svm_Probe.best_estimator_.gamma
                        joblib.dump(svm_Probe, dumpfile)

        elif category_label == 'DoS':

                if updating == False:
                                try:
                                        svm_DoS = joblib.load(dumpfile)
                                        #print('DoS SVM is loaded.')

                                except:
                                        print('New DoS SVM is created.')
                                        df_svm_DoS, df_svm_DoS_labels =
svm_train_data('DoS')
                                        #svm_DoS = svm_svc()
                                        svm_DoS = SVC(C=26.9594844333,
gamma=0.0001)
                                        svm_DoS.fit(df_svm_DoS, df_svm_DoS_labels)
                                        #print 'clf.best_estimator_.C = ',
svm_DoS.best_estimator_.C
                                        #print 'clf.best_estimator_.gamma = ',
svm_DoS.best_estimator_.gamma
                                        joblib.dump(svm_DoS, dumpfile)
                else:
```

```
                        print('New DoS Updated SVM is created.')
                        df_svm_DoS, df_svm_DoS_labels = svm_train_data('DoS')
                        #svm_DoS = svm_svc()
                        svm_DoS = SVC(C=26.9594844333, gamma=0.0001)
                        svm_DoS.fit(df_svm_DoS, df_svm_DoS_labels)
                        #print 'clf.best_estimator_.C = ', svm_DoS.best_estimator_.C
                        #print 'clf.best_estimator_.gamma = ',
svm_DoS.best_estimator_.gamma
                        joblib.dump(svm_DoS, dumpfile)

        elif category_label == 'R2L':

                if updating == False:
                                try:
                                        svm_R2L = joblib.load(dumpfile)
                                        #print('R2L SVM is loaded.')

                                except:
                                        print('New R2L SVM is created.')
                                        df_svm_R2L, df_svm_R2L_labels =
svm_train_data('R2L')
                                        #svm_R2L = svm_svc()
                                        svm_R2L = SVC(C=1.0,
gamma=0.000494171336132)
                                        svm_R2L.fit(df_svm_R2L, df_svm_R2L_labels)
                                        #print 'clf.best_estimator_.C = ',
svm_R2L.best_estimator_.C
                                        #print 'clf.best_estimator_.gamma = ',
svm_R2L.best_estimator_.gamma
                                        joblib.dump(svm_R2L, dumpfile)
                else:
                        print('New R2L Updated SVM is created.')
                        df_svm_R2L, df_svm_R2L_labels = svm_train_data('R2L')
                        #svm_R2L = svm_svc()
                        svm_R2L = SVC(C=1.0, gamma=0.000494171336132)
                        svm_R2L.fit(df_svm_R2L, df_svm_R2L_labels)
                        #print 'clf.best_estimator_.C = ', svm_R2L.best_estimator_.C
                        #print 'clf.best_estimator_.gamma = ',
svm_R2L.best_estimator_.gamma
                        joblib.dump(svm_R2L, dumpfile)

        else:

                if updating == False:
                                try:
                                        svm_U2R = joblib.load(dumpfile)
                                        #print('U2R SVM is loaded.')

                                except:
                                        print('New U2R SVM is created.')
```

```
                                        df_svm_U2R, df_svm_U2R_labels =
svm_train_data('U2R')
                                        #svm_U2R = svm_svc()
                                        svm_U2R = SVC(C=1.50952026539,
gamma=0.00104811313415)
                                        svm_U2R.fit(df_svm_U2R, df_svm_U2R_labels)
                                        #print 'clf.best_estimator_.C = ',
svm_U2R.best_estimator_.C
                                        #print 'clf.best_estimator_.gamma = ',
svm_U2R.best_estimator_.gamma
                                        joblib.dump(svm_U2R, dumpfile)
                else:
                        print('New U2R Updated SVM is created.')
                        df_svm_U2R, df_svm_U2R_labels = svm_train_data('U2R')
                        #svm_U2R = svm_svc()
                        svm_U2R = SVC(C=1.50952026539,
gamma=0.00104811313415)
                        svm_U2R.fit(df_svm_U2R, df_svm_U2R_labels)
                        #print 'clf.best_estimator_.C = ', svm_U2R.best_estimator_.C
                        #print 'clf.best_estimator_.gamma = ',
svm_U2R.best_estimator_.gamma
                        joblib.dump(svm_U2R, dumpfile)

        svm_trained = eval('svm_{0}'.format(category_label))

        return svm_trained
```

A.7 Sort Prediction Pairs and Run multi-class SVMs

```
import n_soinn_svm_data
from n_soinn_svm_data import *
from sklearn.svm import SVC
#from sklearn.svm import LinearSVC
from pathlib import Path
import os
import itertools
from operator import itemgetter
import heapq

global dumpfile
global category_label
global path
global bin_svm_top3_score_list_final
```

```
bin_svm_top3_score_list_final = []

dir_path = os.path.dirname(os.path.realpath(__file__))
path = Path(dir_path)



def get_all_prediction_pairs(test_data,test_labels):

        global Scores
        global Predictions
        z = test_data.reshape(1,-1)

        svm_normal = load_svms('normal')
        normal_score_list = []
        normal_pred_pair = []
        normal_score = svm_normal.decision_function(z)
        normal_score_list.append('normal')
        normal_score_list.append(abs(normal_score))
        normal_pred = svm_normal.predict(z)
        normal_score_list.append(normal_pred[0])
        normal_pred_pair.append(normal_score_list)

        svm_Probe = load_svms('Probe')
        Probe_score_list = []
        Probe_pred_pair = []
        Probe_score = svm_Probe.decision_function(z)
        Probe_score_list.append('Probe')
        Probe_score_list.append(abs(Probe_score))
        Probe_pred = svm_Probe.predict(z)
        Probe_score_list.append(Probe_pred[0])
        Probe_pred_pair.append(Probe_score_list)

        svm_DoS = load_svms('DoS')
        DoS_score_list = []
        DoS_pred_pair = []
        DoS_score = svm_DoS.decision_function(z)
        DoS_score_list.append('DoS')
        DoS_score_list.append(abs(DoS_score))
        DoS_pred = svm_DoS.predict(z)
        DoS_score_list.append(DoS_pred[0])
        DoS_pred_pair.append(DoS_score_list)

        svm_R2L = load_svms('R2L')
        R2L_score_list = []
        R2L_pred_pair = []
        R2L_score = svm_R2L.decision_function(z)
        R2L_score_list.append('R2L')
        R2L_score_list.append(abs(R2L_score))
        R2L_pred = svm_R2L.predict(z)
```

```
        R2L_score_list.append(R2L_pred[0])
        R2L_pred_pair.append(R2L_score_list)

        svm_U2R = load_svms('U2R')
        U2R_score_list = []
        U2R_pred_pair = []
        U2R_score = svm_U2R.decision_function(z)
        U2R_score_list.append('U2R')
        U2R_score_list.append(abs(U2R_score))
        U2R_pred = svm_U2R.predict(z)
        U2R_score_list.append(U2R_pred[0])
        U2R_pred_pair.append(U2R_score_list)

        Scores = [normal_score_list, Probe_score_list, DoS_score_list, R2L_score_list,
U2R_score_list]
        Predictions = [normal_pred, Probe_pred, DoS_pred, R2L_pred, U2R_pred]
        Pred_Pairs = zip(normal_pred_pair, Probe_pred_pair, DoS_pred_pair,
R2L_pred_pair, U2R_pred_pair)

        return Scores, Predictions, Pred_Pairs




def get_top_3_score_predictions(Pred_Pairs):
        top_3_list = []
        for cat_labels in Pred_Pairs:
                h = []
                for value in cat_labels:
                        heapq.heappush(h, value)
                top_3_list.append(heapq.nlargest(3, h, key=lambda x: x[1]))
        return top_3_list[0]

#print get_top_3_score_predictions(Pred_Pairs)

def get_svm_multiclass_data_input(single_pred,updating=False):
        Soinn_Data1 = None
        Soinn_Data2 = None
        Soinn_Data3 = None
        i = 0
        for pred in single_pred:
                i = i + 1
                if pred[0] == 'normal':
                        if updating == False:
                                df_svm_normal, df_svm_normal_labels =
multiclass_svm_datasets('normal')
                        else:
                                df_svm_normal, df_svm_normal_labels =
multiclass_svm_datasets('normal',True)
```

```
                        df_svm_normal.loc[:,'attack'] = 'normal'
                        if i == 1:
                                Soinn_Data1 = df_svm_normal, df_svm_normal_labels
                        elif i == 2:
                                Soinn_Data2 = df_svm_normal, df_svm_normal_labels
                        else:
                                 Soinn_Data3 = df_svm_normal, df_svm_normal_labels
                elif pred[0] == 'DoS':
                        if updating == False:
                                df_svm_DoS, df_svm_DoS_labels =
multiclass_svm_datasets('DoS')
                        else:
                                df_svm_DoS, df_svm_DoS_labels =
multiclass_svm_datasets('DoS',True)
                        df_svm_DoS.loc[:,'attack'] = 'DoS'
                        if i == 1:
                                Soinn_Data1 = df_svm_DoS, df_svm_DoS_labels
                        elif i == 2:
                                Soinn_Data2 = df_svm_DoS, df_svm_DoS_labels
                        else:
                                 Soinn_Data3 = df_svm_DoS, df_svm_DoS_labels
                elif pred[0] == 'Probe':
                        if updating == False:
                                df_svm_Probe, df_svm_Probe_labels =
multiclass_svm_datasets('Probe')
                        else:
                                df_svm_Probe, df_svm_Probe_labels =
multiclass_svm_datasets('Probe',True)
                        df_svm_Probe.loc[:,'attack'] = 'Probe'
                        if i == 1:
                                Soinn_Data1 = df_svm_Probe, df_svm_Probe_labels
                        elif i == 2:
                                Soinn_Data2 = df_svm_Probe, df_svm_Probe_labels
                        else:
                                 Soinn_Data3 = df_svm_Probe, df_svm_Probe_labels
                elif pred[0] == 'U2R':
                        if updating == False:
                                df_svm_U2R, df_svm_U2R_labels =
multiclass_svm_datasets('U2R')
                        else:
                                df_svm_U2R, df_svm_U2R_labels =
multiclass_svm_datasets('U2R',True)
                        df_svm_U2R.loc[:,'attack'] = 'U2R'
                        if i == 1:
                                Soinn_Data1 = df_svm_U2R, df_svm_U2R_labels
                        elif i == 2:
                                Soinn_Data2 = df_svm_U2R, df_svm_U2R_labels
                        else:
                                 Soinn_Data3 = df_svm_U2R, df_svm_U2R_labels
                else:
```

```
                        if updating == False:
                                df_svm_R2L, df_svm_R2L_labels =
multiclass_svm_datasets('R2L')
                        else:
                                df_svm_R2L, df_svm_R2L_labels =
multiclass_svm_datasets('R2L',True)
                        df_svm_R2L.loc[:,'attack'] = 'R2L'
                        if i == 1:
                                Soinn_Data1 = df_svm_R2L, df_svm_R2L_labels
                        elif i == 2:
                                Soinn_Data2 = df_svm_R2L, df_svm_R2L_labels
                        else:
                                 Soinn_Data3 = df_svm_R2L, df_svm_R2L_labels

        return Soinn_Data1, Soinn_Data2, Soinn_Data3


def train_multiclass_SVM(SVM_Data1, SVM_Data2,
SVM_Data3,updating=False,attack_string=False):
        df_svm1, df_svm_labels1 = SVM_Data1
        df_svm2, df_svm_labels2 = SVM_Data2
        df_svm3, df_svm_labels3 = SVM_Data3
        df_svm1_cat = df_svm1.iloc[0]['attack']
        df_svm2_cat = df_svm2.iloc[0]['attack']
        df_svm3_cat = df_svm3.iloc[0]['attack']
        df_svm1 = df_svm1.drop(columns='attack')
        df_svm2 = df_svm2.drop(columns='attack')
        df_svm3 = df_svm3.drop(columns='attack')
        df_svm = df_svm1.append([df_svm2, df_svm3])
        df_svm = df_svm.as_matrix(columns=None)
        df_svm= df_svm.astype('float')
        df_svm_labels = df_svm_labels1.append([df_svm_labels2, df_svm_labels3])
        df_svm_labels = df_svm_labels.as_matrix(columns=None)
        df_svm_labels = df_svm_labels.astype('float')

        if df_svm1_cat == 'normal':
                if df_svm2_cat == 'DoS':
                        if df_svm3_cat == 'Probe':
                                attack_name = 'NPD'
                        elif df_svm3_cat == 'U2R':
                                attack_name = 'NDU'
                        else:
                                attack_name = 'NDR'
                elif df_svm2_cat == 'Probe':
                        if df_svm3_cat == 'DoS':
                                attack_name = 'NPD'
                        elif df_svm3_cat == 'U2R':
                                attack_name = 'NPU'
                        else:
                                attack_name = 'NPR'
```

```
            elif df_svm2_cat == 'U2R':
                    if df_svm3_cat == 'DoS':
                            attack_name = 'NDU'
                    elif df_svm3_cat == 'Probe':
                            attack_name = 'NPU'
                    else:
                            attack_name = 'NPR'
            else:
                    if df_svm3_cat == 'DoS':
                            attack_name = 'NDR'
                    elif df_svm3_cat == 'Probe':
                            attack_name = 'NPR'
                    else:
                            attack_name = 'NUR'
    elif df_svm1_cat == 'DoS':
            if df_svm2_cat == 'normal':
                    if df_svm3_cat == 'Probe':
                            attack_name = 'NPD'
                    elif df_svm3_cat == 'U2R':
                            attack_name = 'NDU'
                    else:
                            attack_name = 'NDR'
            elif df_svm2_cat == 'Probe':
                    if df_svm3_cat == 'normal':
                            attack_name = 'NPD'
                    elif df_svm3_cat == 'U2R':
                            attack_name = 'PDU'
                    else:
                            attack_name = 'PDR'
            elif df_svm2_cat == 'U2R':
                    if df_svm3_cat == 'normal':
                            attack_name = 'NDU'
                    elif df_svm3_cat == 'Probe':
                            attack_name = 'PDU'
                    else:
                            attack_name = 'DUR'
            else:
                    if df_svm3_cat == 'normal':
                            attack_name = 'NDR'
                    elif df_svm3_cat == 'Probe':
                            attack_name = 'PDR'
                    else:
                            attack_name = 'DUR'
    elif df_svm1_cat == 'Probe':
            if df_svm2_cat == 'normal':
                    if df_svm3_cat == 'DoS':
                            attack_name = 'NPD'
                    elif df_svm3_cat == 'U2R':
                            attack_name = 'NPU'
                    else:
```

```
                        attack_name = 'NPR'
            elif df_svm2_cat == 'DoS':
                    if df_svm3_cat == 'normal':
                        attack_name = 'NPD'
                    elif df_svm3_cat == 'U2R':
                        attack_name = 'PDU'
                    else:
                        attack_name = 'PDR'
            elif df_svm2_cat == 'U2R':
                    if df_svm3_cat == 'normal':
                        attack_name = 'NPU'
                    elif df_svm3_cat == 'DoS':
                        attack_name = 'PDU'
                    else:
                        attack_name = 'PUR'
            else:
                    if df_svm3_cat == 'normal':
                        attack_name = 'NDR'
                    elif df_svm3_cat == 'DoS':
                        attack_name = 'PDR'
                    else:
                        attack_name = 'PUR'
    elif df_svm1_cat == 'U2R':
            if df_svm2_cat == 'normal':
                    if df_svm3_cat == 'DoS':
                        attack_name = 'NDU'
                    elif df_svm3_cat == 'Probe':
                        attack_name = 'NPU'
                    else:
                        attack_name = 'NUR'
            elif df_svm2_cat == 'DoS':
                    if df_svm3_cat == 'normal':
                        attack_name = 'NDU'
                    elif df_svm3_cat == 'Probe':
                        attack_name = 'PDU'
                    else:
                        attack_name = 'DUR'
            elif df_svm2_cat == 'Probe':
                    if df_svm3_cat == 'normal':
                        attack_name = 'NPU'
                    elif df_svm3_cat == 'DoS':
                        attack_name = 'PDU'
                    else:
                        attack_name = 'PUR'
            else:
                    if df_svm3_cat == 'normal':
                        attack_name = 'NUR'
                    elif df_svm3_cat == 'DoS':
                        attack_name = 'DUR'
                    else:
```

```
                                attack_name = 'PUR'

        else:
                if df_svm2_cat == 'normal':
                        if df_svm3_cat == 'DoS':
                                attack_name = 'NDR'
                        elif df_svm3_cat == 'Probe':
                                attack_name = 'NPR'
                        else:
                                attack_name = 'NUR'
                elif df_svm2_cat == 'DoS':
                        if df_svm3_cat == 'normal':
                                attack_name = 'NDR'
                        elif df_svm3_cat == 'Probe':
                                attack_name = 'PDR'
                        else:
                                attack_name = 'DUR'
                elif df_svm2_cat == 'Probe':
                        if df_svm3_cat == 'normal':
                                attack_name = 'NPR'
                        elif df_svm3_cat == 'DoS':
                                attack_name = 'PDR'
                        else:
                                attack_name = 'PUR'
                else:
                        if df_svm3_cat == 'normal':
                                attack_name = 'NUR'
                        elif df_svm3_cat == 'DoS':
                                attack_name = 'DUR'
                        else:
                                attack_name = 'PUR'

        dumpfile = path / 'dump' / 'Multi_SVM' /
'MSVM_{0}.dump'.format(attack_name)

        if attack_name == 'NPD':
                Cv = 50.0
                gammaV = 0.000868511373751
        elif attack_name == 'NDR':
                Cv = 50.0
                gammaV = 0.000281176869797
        elif attack_name == 'NPR':
                Cv = 3.4396705138
                gammaV = 0.00115139539933
        elif attack_name == 'NDU':
                Cv = 50.0
                gammaV = 0.0001
        elif attack_name == 'PUR':
                Cv = 9.62973227417
                gammaV = 0.000175751062485
```

```
        elif attack_name == 'PDU':
                Cv = 50.0
                gammaV = 0.00014563484775
        elif attack_name == 'NUR':
                Cv = 3.4396705138
                gammaV = 0.00115139539933
        elif attack_name == 'PDR':
                Cv = 50.0
                gammaV = 0.000159985871961
        elif attack_name
== 'NPU':
                Cv = 50.0
                gammaV = 0.00115139539933
        else:
                Cv = 7.83781014057
                gammaV = 0.0001



        if updating == False:
                try:
                        train = joblib.load(dumpfile)

                except:
                        print 'New {0} Multiclass SVM is created'.format(attack_name)
                        train = SVC(C=Cv, gamma=gammaV)
                        #parameters = {'C':[1, 10]}
                        #svc = SVC()
                        #clf = GridSearchCV(svc, parameters)
                        #Cs = np.geomspace(1, 50,num=20)
                        #gammas = np.geomspace(0.0001, 0.01,num=50)
                        #train = GridSearchCV(estimator=svc,
param_grid=dict(C=Cs,gamma=gammas),n_jobs=-1)
                        train.fit(df_svm, df_svm_labels)
                        #print 'clf.best_estimator_.C = ', train.best_estimator_.C
                        #print 'clf.best_estimator_.gamma = ',
train.best_estimator_.gamma
                        joblib.dump(train, dumpfile)
        else:

                print 'New {0} Updated Multiclass SVM is
created'.format(attack_name)
                #train = SVC(C=24, gamma='auto',probability=False)
                train = train = SVC(C=Cv, gamma=gammaV)
                #parameters = {'C':[1, 10]}
                #svc = SVC()
                #clf = GridSearchCV(svc, parameters)
                #Cs = np.geomspace(1, 50,num=20)
                #gammas = np.geomspace(0.0001, 0.01,num=50)
                #train = GridSearchCV(estimator=svc,
```

```
param_grid=dict(C=Cs,gamma=gammas),n_jobs=-1)
                train.fit(df_svm, df_svm_labels)
                #print 'clf.best_estimator_.C = ', train.best_estimator_.C
                #print 'clf.best_estimator_.gamma = ', train.best_estimator_.gamma
                joblib.dump(train, dumpfile)

        return train, attack_name

def get_scores(data,labels,keep_track,troubleshoot=False):

        i = keep_track
        score_attack_name_list = []


        if troubleshoot == False:
                Scores, Predictions, Pred_Pairs = get_all_prediction_pairs(data,labels)
                top_3_pred_list = get_top_3_score_predictions(Pred_Pairs)
                SVM_Data1, SVM_Data2, SVM_Data3 =
get_svm_multiclass_data_input(top_3_pred_list)
                final_SVM, attack_name = train_multiclass_SVM(SVM_Data1,
SVM_Data2, SVM_Data3)
                z = data.reshape(1,-1)
                single_score = final_SVM.score(z,labels)
                single_prediction = final_SVM.predict(z)
                score_attack_name = []
                score_attack_name.append(attack_name)
                score_attack_name.append(single_score)
                score_attack_name_list.append(score_attack_name)



        else:

                Scores, Predictions, Pred_Pairs = get_all_prediction_pairs(data,labels)
                top_3_pred_list = get_top_3_score_predictions(Pred_Pairs)
                SVM_Data1, SVM_Data2, SVM_Data3 =
get_svm_multiclass_data_input(top_3_pred_list)
                final_SVM, attack_name = train_multiclass_SVM(SVM_Data1,
SVM_Data2, SVM_Data3)
                z = data.reshape(1,-1)
                single_score = final_SVM.score(z,labels)
                single_prediction = final_SVM.predict(z)
                #print 'single_prediction = ',single_prediction
                #print 'attack_name',attack_name
                #quit()
                score_attack_name = []
                score_attack_name.append(attack_name)
                score_attack_name.append(single_score)
                score_attack_name_list.append(score_attack_name)
                #score_attack_name_list_for_trouble_shoot.append(attack_name)
```

```
                if single_prediction == 0:
                        single_prediction_name = 'normal'
                elif single_prediction == 1:
                        single_prediction_name = 'U2R'
                elif single_prediction == 2:
                        single_prediction_name = 'R2L'
                elif single_prediction == 3:
                        single_prediction_name = 'DoS'
                else:
                        single_prediction_name = 'Probe'

                if labels[0] == 0:
                        True_Label_name = 'normal'
                elif labels[0] == 1:
                        True_Label_name = 'U2R'
                elif labels[0] == 2:
                        True_Label_name = 'R2L'
                elif labels[0] == 3:
                        True_Label_name = 'DoS'
                else:
                        True_Label_name = 'Probe'



                top3_test_prediction_score_def(top_3_pred_list,True_Label_name)
                #if single_prediction_name != True_Label_name:
                        #print '{0}_th Failed Prediction - Pred_Pairs = '.format(i),
Pred_Pairs
                        #print '{0}_th Failed Prediction - top_3_pred_list = '.format(i),
top_3_pred_list
                        #print '{0}_th Failed Prediction - Top 3 = '.format(i),
attack_name
                        #print '{0}_th Failed Prediction - single_prediction  =
'.format(i), single_prediction_name
                        #print '{0}_th Failed Prediction - True Label  = '.format(i),
True_Label_name
                #else:
                        #print '{0}_th Correct Prediction Pred_Pairs = '.format(i),
Pred_Pairs
                        #print '{0}_th Correct Prediction top_3_pred_list = '.format(i),
top_3_pred_list
                        #print '{0}_th Correct Prediction Top 3 = '.format(i),
attack_name
                        #print '{0}_th Correct Prediction single_prediction =
'.format(i), single_prediction_name

        return score_attack_name_list, single_score, single_prediction

def top3_test_prediction_score_def(top_3_pred_list,True_Label_name):
```

```
        global top3_test_prediction_score
        top3_test_prediction_score = 0
        bin_svm_top3_score_list = []
        for single_top_3_pred_list in top_3_pred_list:
                if single_top_3_pred_list[0] != True_Label_name:
                        bin_svm_top3_score_list.append(0)
                else:
                        bin_svm_top3_score_list.append(1)
        #print 'bin_svm_top3_score_list', bin_svm_top3_score_list
        if bin_svm_top3_score_list[0] == 1 or bin_svm_top3_score_list[1] == 1 or
bin_svm_top3_score_list[-1] == 1:
                bin_svm_top3_score_list_final.append(1)
        else:
                bin_svm_top3_score_list_final.append(0)
        #print 'bin_svm_top3_score_list_final',bin_svm_top3_score_list_final
        top3_test_prediction_score = reduce(lambda x, y: x + y,
bin_svm_top3_score_list_final) / float(len(bin_svm_top3_score_list_final))



def top3_test_prediction_score_def_to_main():
        top3_test_prediction_score_m = top3_test_prediction_score
        return top3_test_prediction_score_m


def final_update_of_all_BSVMs():

        load_svms('normal',True)
        load_svms('Probe',True)
        load_svms('DoS',True)
        load_svms('R2L',True)
        load_svms('U2R',True)
        final_update_of_all_MSVMs_df_files()

def final_update_of_all_MSVMs_df_files():
        multiclass_svm_datasets('normal',True)
        multiclass_svm_datasets('DoS',True)
        multiclass_svm_datasets('Probe',True)
        multiclass_svm_datasets('U2R',True)
        multiclass_svm_datasets('R2L',True)
        final_update_of_all_MSVMs()


def final_update_of_all_MSVMs():
        from itertools import combinations
        msvms_comb = list(combinations(["N", "P", "D", "R", "U"], 3))
        full_msvms_comb = ["".join(a) for a in msvms_comb]
        for one_let_pred in full_msvms_comb:
                top_3_pred_list = []
                if one_let_pred[0] == 'N':
```

```
                        one_let_pred_name1 = 'normal'
                elif one_let_pred[0] == 'U':
                        one_let_pred_name1 = 'U2R'
                elif one_let_pred[0] == 'R':
                        one_let_pred_name1 = 'R2L'
                elif one_let_pred[0] == 'D':
                        one_let_pred_name1 = 'DoS'
                else:
                        one_let_pred_name1 = 'Probe'
                #print one_let_pred_name1

                if one_let_pred[1] == 'N':
                        one_let_pred_name2 = 'normal'
                elif one_let_pred[1] == 'U':
                        one_let_pred_name2 = 'U2R'
                elif one_let_pred[1] == 'R':
                        one_let_pred_name2 = 'R2L'
                elif one_let_pred[1] == 'D':
                        one_let_pred_name2 = 'DoS'
                else:
                        one_let_pred_name2 = 'Probe'
                #print one_let_pred_name2

                if one_let_pred[-1] == 'N':
                        one_let_pred_name3 = 'normal'
                elif one_let_pred[-1] == 'U':
                        one_let_pred_name3 = 'U2R'
                elif one_let_pred[-1] == 'R':
                        one_let_pred_name3 = 'R2L'
                elif one_let_pred[-1] == 'D':
                        one_let_pred_name3 = 'DoS'
                else:
                        one_let_pred_name3 = 'Probe'
                #print one_let_pred_name3
                top_3_pred_list.append([one_let_pred_name1,0,0])
                top_3_pred_list.append([one_let_pred_name2,0,0])
                top_3_pred_list.append([one_let_pred_name3,0,0])
                SVM_Data1, SVM_Data2, SVM_Data3 =
get_svm_multiclass_data_input(top_3_pred_list,True)
                #print SVM_Data1, SVM_Data2, SVM_Data3
                train_multiclass_SVM(SVM_Data1, SVM_Data2, SVM_Data3,True)

if __name__ == '__main__':
        final_update_of_all_MSVMs()
```

A.8 Update Function

```
def update_svm_train_data(updated_soinn):

        normal_negatives_soinns_update = []
        Probe_negatives_soinns_update = []
        R2L_negatives_soinns_update = []
        U2R_negatives_soinns_update = []
        DoS_negatives_soinns_update = []

        df_pos_dumpfile_updated = path / 'dump' / 'Soinns_datasets' /
'df_{0}_pos.dump'.format(updated_soinn)
        df_neg_dumpfile_updated = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(updated_soinn)

        lo_soinn_dumpfile_updated = path / 'dump' / 'SOINNs' /
'soinn2_{0}.dump'.format(updated_soinn)
        hi_soinn_dumpfile_updated = path / 'dump' / 'SOINNs' /
'soinn100_{0}.dump'.format(updated_soinn)

        if updated_soinn == 'normal':
                lo_soinn_normal = joblib.load(lo_soinn_dumpfile_updated)
                hi_soinn_normal = joblib.load(hi_soinn_dumpfile_updated)
                try:
                        os.remove(str(df_pos_dumpfile_updated))
                        os.remove(str(df_neg_dumpfile_updated))
                except:
                        pass
                pos_values_normal = []
                for pos_node in lo_soinn_normal.nodes:
                        pos_values_normal.append(list(pos_node))
                df_normal_pos = pd.DataFrame(pos_values_normal)
                df_normal_pos.loc[:,'labels'] = 1
                neg_values_normal = []
                for neg_node in hi_soinn_normal.nodes:
                        neg_values_normal.append(list(neg_node))
                df_normal_neg = pd.DataFrame(neg_values_normal)
                df_normal_neg.loc[:,'labels'] = 0
                joblib.dump(df_normal_pos, df_pos_dumpfile_updated)
                joblib.dump(df_normal_neg, df_neg_dumpfile_updated)


        if updated_soinn == 'Probe':
                lo_soinn_Probe = joblib.load(lo_soinn_dumpfile_updated)
                hi_soinn_Probe = joblib.load(hi_soinn_dumpfile_updated)
                try:
                        os.remove(str(df_pos_dumpfile_updated))
                        os.remove(str(df_neg_dumpfile_updated))
```

```
            except:
                pass
            pos_values_Probe = []
            for pos_node in lo_soinn_Probe.nodes:
                pos_values_Probe.append(list(pos_node))
            df_Probe_pos = pd.DataFrame(pos_values_Probe)
            df_Probe_pos.loc[:,'labels'] = 1
            neg_values_Probe = []
            for neg_node in hi_soinn_Probe.nodes:
                neg_values_Probe.append(list(neg_node))
            df_Probe_neg = pd.DataFrame(neg_values_Probe)
            df_Probe_neg.loc[:,'labels'] = 0
            joblib.dump(df_Probe_pos, df_pos_dumpfile_updated)
            joblib.dump(df_Probe_neg, df_neg_dumpfile_updated)


    if updated_soinn == 'R2L':
            lo_soinn_R2L = joblib.load(lo_soinn_dumpfile_updated)
            hi_soinn_R2L = joblib.load(hi_soinn_dumpfile_updated)
            try:
                os.remove(str(df_pos_dumpfile_updated))
                os.remove(str(df_neg_dumpfile_updated))
            except:
                pass
            pos_values_R2L = []
            for pos_node in lo_soinn_R2L.nodes:
                pos_values_R2L.append(list(pos_node))
            df_R2L_pos = pd.DataFrame(pos_values_R2L)
            df_R2L_pos.loc[:,'labels'] = 1
            neg_values_R2L = []
            for neg_node in hi_soinn_R2L.nodes:
                neg_values_R2L.append(list(neg_node))
            df_R2L_neg = pd.DataFrame(neg_values_R2L)
            df_R2L_neg.loc[:,'labels'] = 0
            joblib.dump(df_R2L_pos, df_pos_dumpfile_updated)
            joblib.dump(df_R2L_neg, df_neg_dumpfile_updated)


    if updated_soinn == 'U2R':
            lo_soinn_U2R = joblib.load(lo_soinn_dumpfile_updated)
            hi_soinn_U2R = joblib.load(hi_soinn_dumpfile_updated)
            try:
                os.remove(str(df_pos_dumpfile_updated))
                os.remove(str(df_neg_dumpfile_updated))
            except:
                pass
            pos_values_U2R = []
            for pos_node in lo_soinn_U2R.nodes:
                pos_values_U2R.append(list(pos_node))
            df_U2R_pos = pd.DataFrame(pos_values_U2R)
```

```
                df_U2R_pos.loc[:,'labels'] = 1
                neg_values_U2R = []
                for neg_node in hi_soinn_U2R.nodes:
                        neg_values_U2R.append(list(neg_node))
                df_U2R_neg = pd.DataFrame(neg_values_U2R)
                df_U2R_neg.loc[:,'labels'] = 0
                joblib.dump(df_U2R_pos, df_pos_dumpfile_updated)
                joblib.dump(df_U2R_neg, df_neg_dumpfile_updated)

        if updated_soinn == 'DoS':
                lo_soinn_DoS = joblib.load(lo_soinn_dumpfile_updated)
                hi_soinn_DoS = joblib.load(hi_soinn_dumpfile_updated)
                try:
                        os.remove(str(df_pos_dumpfile_updated))
                        os.remove(str(df_neg_dumpfile_updated))
                except:
                        pass
                pos_values_DoS = []
                for pos_node in lo_soinn_DoS.nodes:
                        pos_values_DoS.append(list(pos_node))
                df_DoS_pos = pd.DataFrame(pos_values_DoS)
                df_DoS_pos.loc[:,'labels'] = 1
                neg_values_DoS = []
                for neg_node in hi_soinn_DoS.nodes:
                        neg_values_DoS.append(list(neg_node))
                df_DoS_neg = pd.DataFrame(neg_values_DoS)
                df_DoS_neg.loc[:,'labels'] = 0
                joblib.dump(df_DoS_pos, df_pos_dumpfile_updated)
                joblib.dump(df_DoS_neg, df_neg_dumpfile_updated)



        for normal_negative in normal_negatives:
                df_soinn_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(normal_negative)
                normal_df_neg = joblib.load(df_soinn_neg_dumpfile)
                normal_negatives_soinns_update.append(normal_df_neg)

        df_soinn_pos_normal_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_normal_pos.dump'
        df_normal_pos = joblib.load(df_soinn_pos_normal_dumpfile)
        df_svm_normal = df_normal_pos.append(normal_negatives_soinns_update)
        df_svm_normal_labels = df_svm_normal['labels']
        df_svm_normal = df_svm_normal.drop(columns='labels')
        df_svm_normal = df_svm_normal.reset_index(drop=True)
        df_svm_normal_labels = df_svm_normal_labels.reset_index(drop=True)
        df_svm_dumpfile_normal = path / 'dump' / 'svms_datasets' /
'df_svm_normal.dump'
        df_svm_labels_dumpfile_normal = path / 'dump' / 'svms_datasets' /
'df_svm_normal_labels.dump'
```

```
        os.remove(str(df_svm_dumpfile_normal))
        os.remove(str(df_svm_labels_dumpfile_normal))
        joblib.dump(df_svm_normal, df_svm_dumpfile_normal)
        joblib.dump(df_svm_normal_labels, df_svm_labels_dumpfile_normal)


        for Probe_negative in Probe_negatives:
                df_soinn_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(Probe_negative)
                Probe_df_neg = joblib.load(df_soinn_neg_dumpfile)
                Probe_negatives_soinns_update.append(Probe_df_neg)

        df_soinn_pos_Probe_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_Probe_pos.dump'
        df_Probe_pos = joblib.load(df_soinn_pos_Probe_dumpfile)
        df_svm_Probe = df_Probe_pos.append(Probe_negatives_soinns_update)
        df_svm_Probe_labels = df_svm_Probe['labels']
        df_svm_Probe = df_svm_Probe.drop(columns='labels')
        df_svm_Probe = df_svm_Probe.reset_index(drop=True)
        df_svm_Probe_labels = df_svm_Probe_labels.reset_index(drop=True)
        df_svm_dumpfile_Probe = path / 'dump' / 'svms_datasets' /
'df_svm_Probe.dump'
        df_svm_labels_dumpfile_Probe = path / 'dump' / 'svms_datasets' /
'df_svm_Probe_labels.dump'
        os.remove(str(df_svm_dumpfile_Probe))
        os.remove(str(df_svm_labels_dumpfile_Probe))
        joblib.dump(df_svm_Probe, df_svm_dumpfile_Probe)
        joblib.dump(df_svm_Probe_labels, df_svm_labels_dumpfile_Probe)


        for R2L_negative in R2L_negatives:
                df_soinn_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(R2L_negative)
                R2L_df_neg = joblib.load(df_soinn_neg_dumpfile)
                R2L_negatives_soinns_update.append(R2L_df_neg)

        df_soinn_pos_R2L_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_R2L_pos.dump'
        df_R2L_pos = joblib.load(df_soinn_pos_R2L_dumpfile)
        df_svm_R2L = df_R2L_pos.append(R2L_negatives_soinns_update)
        df_svm_R2L_labels = df_svm_R2L['labels']
        df_svm_R2L = df_svm_R2L.drop(columns='labels')
        df_svm_R2L = df_svm_R2L.reset_index(drop=True)
        df_svm_R2L_labels = df_svm_R2L_labels.reset_index(drop=True)
        df_svm_dumpfile_R2L = path / 'dump' / 'svms_datasets' /
'df_svm_R2L.dump'
        df_svm_labels_dumpfile_R2L = path / 'dump' / 'svms_datasets' /
'df_svm_R2L_labels.dump'
        os.remove(str(df_svm_dumpfile_R2L))
        os.remove(str(df_svm_labels_dumpfile_R2L))
```

```
        joblib.dump(df_svm_R2L, df_svm_dumpfile_R2L)
        joblib.dump(df_svm_R2L_labels, df_svm_labels_dumpfile_R2L)


        for U2R_negative in U2R_negatives:
                df_soinn_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(U2R_negative)
                U2R_df_neg = joblib.load(df_soinn_neg_dumpfile)
                U2R_negatives_soinns_update.append(U2R_df_neg)

        df_soinn_pos_U2R_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_U2R_pos.dump'
        df_U2R_pos = joblib.load(df_soinn_pos_U2R_dumpfile)
        df_svm_U2R = df_U2R_pos.append(U2R_negatives_soinns_update)
        df_svm_U2R_labels = df_svm_U2R['labels']
        df_svm_U2R = df_svm_U2R.drop(columns='labels')
        df_svm_U2R = df_svm_U2R.reset_index(drop=True)
        df_svm_U2R_labels = df_svm_U2R_labels.reset_index(drop=True)
        df_svm_dumpfile_U2R = path / 'dump' / 'svms_datasets' /
'df_svm_U2R.dump'
        df_svm_labels_dumpfile_U2R = path / 'dump' / 'svms_datasets' /
'df_svm_U2R_labels.dump'
        os.remove(str(df_svm_dumpfile_U2R))
        os.remove(str(df_svm_labels_dumpfile_U2R))
        joblib.dump(df_svm_U2R, df_svm_dumpfile_U2R)
        joblib.dump(df_svm_U2R_labels, df_svm_labels_dumpfile_U2R)



        for DoS_negative in DoS_negatives:
                df_soinn_neg_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_{0}_neg.dump'.format(DoS_negative)
                DoS_df_neg = joblib.load(df_soinn_neg_dumpfile)
                DoS_negatives_soinns_update.append(DoS_df_neg)

        df_soinn_pos_DoS_dumpfile = path / 'dump' / 'Soinns_datasets' /
'df_DoS_pos.dump'
        df_DoS_pos = joblib.load(df_soinn_pos_DoS_dumpfile)
        df_svm_DoS = df_DoS_pos.append(DoS_negatives_soinns_update)
        df_svm_DoS_labels = df_svm_DoS['labels']
        df_svm_DoS = df_svm_DoS.drop(columns='labels')
        df_svm_DoS = df_svm_DoS.reset_index(drop=True)
        df_svm_DoS_labels = df_svm_DoS_labels.reset_index(drop=True)
        df_svm_dumpfile_DoS = path / 'dump' / 'svms_datasets' /
'df_svm_DoS.dump'
        df_svm_labels_dumpfile_DoS = path / 'dump' / 'svms_datasets' /
'df_svm_DoS_labels.dump'
        os.remove(str(df_svm_dumpfile_DoS))
        os.remove(str(df_svm_labels_dumpfile_DoS))
        joblib.dump(df_svm_DoS, df_svm_dumpfile_DoS)
```

```
        joblib.dump(df_svm_DoS_labels, df_svm_labels_dumpfile_DoS)
```

A.9 Main Function - Initialization

```
from svm_n_soinn import get_scores, final_update_of_all_BSVMs,
top3_test_prediction_score_def_to_main
import joblib
from pathlib import Path
import os
import os.path
global path
from data import soinn_train_data, test_dataset, train_dataset
from ui_n_soinn import create_soinns
import time
import itertools
from n_soinn import Soinn
from n_soinn_svm_data import load_svms,
multiclass_svm_datasets,update_svm_train_data
from sklearn.linear_model import Perceptron
start_time = time.time()

dir_path = os.path.dirname(os.path.realpath(__file__))
path = Path(dir_path)
attacks = ['normal','R2L','U2R','Probe','DoS']


def update_soinns(category_label, train_rows):

        n_soinn_list = [2,100]

        for n_number in n_soinn_list:
                n_soinn = n_number
                dumpfile = path / 'dump' / 'SOINNs' /
'soinn{0}_{1}.dump'.format(n_soinn,category_label)
                soinn_i = joblib.load(dumpfile)
                print 'Category Label - {0} Total Nodes before update:
{1}'.format(category_label,len(soinn_i.nodes))
                for train_row in train_rows:
                        soinn_i.input_signal(train_row)
                print 'Category Label - {0} Total Nodes after update:
{1}'.format(category_label,len(soinn_i.nodes))
                soinn_i.save(dumpfile)

def operate_machine(test_data,test_labels,updating=False,check_soinns=False):
```

```
	if check_soinns == True:
		for attack in attacks:
			n_soinn_list = [2,100]
			for soinn_number in n_soinn_list:
				n_soinn = soinn_number
				dumpfile = path / 'dump' / 'SOINNs' /
'soinn{0}_{1}.dump'.format(n_soinn,attack)
				try:
					dumpfile.resolve()
					#print 'soinn{0}_{1} exist'.format(x,attack)
				except:
					data = soinn_train_data(attack)
					n_soinn_number = soinn_number
					category = attack
					delete_period = 100
					max_age = 30
					create_soinns(data, n_soinn_number, category,
delete_period, max_age)


	if updating == True:
		get_predictions(test_data,test_labels,True)
	else:
		get_predictions(test_data,test_labels,False)


def get_predictions(data,labels,updating=False):

	zipped_data_labels = zip(data,labels)
	score_list = []
	failed_predict_list = []
	correct_predict_list = []
	keep_track_list = []
	count = 0

	for zipped_data_row in zipped_data_labels:
		data_row = zipped_data_row[0]
		label_row = zipped_data_row[-1]
		count += 1
		score_attack_name_list, single_score, single_prediction =
get_scores(data_row,label_row,count,True)
		score_list.append(single_score)
		if single_score != 1:
			#print 'label_row', label_row
			failed_predict_list.append(zipped_data_row)
		else:
			correct_predict_list.append(zipped_data_row)

	if updating == True:
```

```
                normal_failed_pred = []
                Probe_failed_pred = []
                DoS_failed_pred = []
                U2R_failed_pred = []
                R2L_failed_pred = []

                for x in failed_predict_list:
                        update_data_row = x[0]
                        real_category_label_list = x[-1]
                        real_category_label = real_category_label_list[0]
                        if real_category_label == 0:
                                category_label = 'normal'
                                normal_failed_pred.append(update_data_row)
                        elif real_category_label == 1:
                                category_label = 'U2R'
                                U2R_failed_pred.append(update_data_row)
                        elif real_category_label == 2:
                                category_label = 'R2L'
                                R2L_failed_pred.append(update_data_row)
                        elif real_category_label == 3:
                                category_label = 'DoS'
                                DoS_failed_pred.append(update_data_row)
                        else:
                                category_label = 'Probe'
                                Probe_failed_pred.append(update_data_row)

                        #print '{0}_th real_category_label = '.format(keep_track),
category_label

                for attack in attacks:
                        attack_pred_list = eval('{0}_failed_pred'.format(attack))
                        if attack_pred_list:
                                update_soinns(attack, attack_pred_list)

                for attack in attacks:
                        attack_pred_list = eval('{0}_failed_pred'.format(attack))
                        if attack_pred_list:
                                update_svm_train_data(attack)


                final_update_of_all_BSVMs()


        top3_test_prediction_score = top3_test_prediction_score_def_to_main()
        top3_test_prediction_score_perc = top3_test_prediction_score * 100
        print 'Top 3 Achievement Percentage = ',
top3_test_prediction_score_perc ,'%'
        num_achieved_top_3 = int(top3_test_prediction_score * count)
        num_failed_top_3 = count - num_achieved_top_3
```

```
        print 'Top 3 Achievement missed {0} labels out of
{1}'.format(num_failed_top_3,count)


        num_failed_labels = len(failed_predict_list)
        num_labels = len(zipped_data_labels)
        #for i, score_num in enumerate(score_list):
        #       if score_num == 0:
        #               print '{0}th Prediction failed '.format(i+1)
        num_labels = len(zipped_data_labels)
        cum_prediction_score = reduce(lambda x, y: x + y, score_list) /
float(len(score_list))
        cum_prediction_score_perc = cum_prediction_score * 100
        print 'MSVMs Prediction = ', cum_prediction_score_perc ,'%'
        print 'MSVMs missed {0} labels out of
{1}'.format(num_failed_labels,num_labels)
        print 'Prediction from a dataset of {0} rows'.format(num_labels)
        print("time cost =  %s seconds" % (time.time() - start_time))


def sub_sample_data():

        from sklearn.model_selection import StratifiedKFold
        train_df, df_train_labels = train_dataset()
        train_df = train_df.as_matrix()
        train_df = train_df.astype('float')
        df_train_labels = df_train_labels.as_matrix()
        df_train_labels = df_train_labels.astype('float')
        #df_train_labels = df_train_labels.tolist()

        X = train_df
        y = df_train_labels # subsamples will be stratified according to y
        n = 5
        skf = StratifiedKFold(n, shuffle = True)

        ith = 0

        for _, batch in skf.split(X, y):
                #train_df_sliced = pd.DataFrame(X[batch])
                #df_train_labels_sliced = pd.DataFrame(y[batch])
                #print train_df_sliced.describe()
                #print df_train_labels_sliced.describe()
                test_data = X[batch]
                test_labels = y[batch]
                operate_machine(test_data,test_labels,True,True)
                #ith += 1
                #print ith
                #if ith == 5:
                #       operate_machine(test_data,test_labels,False,False)
```

```
if __name__ == '__main__':

        #sub_sample_data()
        test_df, df_test_labels = train_dataset()
        #test_data = test_df.loc[8000:9029]
        #test_labels = df_test_labels.loc[8000:9029]
        test_data = test_df
        test_labels = df_test_labels
        test_data = test_data.as_matrix()
        test_data = test_data.astype('float')
        test_labels = test_labels.as_matrix()
        test_labels = test_labels.astype('float')
        test_labels = test_labels.tolist()
        operate_machine(test_data,test_labels,False,False)
```

Bibliography

A. Kawewong, S. Tangruamsub, P. Kankuekul, and O. Hasegawa. "Fast online incremental transfer learning for unseen object classification using self-organizing incremental neural networks." Neural Networks (IJCNN), The 2011 International Joint Conference on. IEEE, 2011.

Abuadlla, Yousef, et al. "Flow-based anomaly intrusion detection system using two neural network stages." Computer Science and Information Systems 11.2 (2014): 601-622.

Bace, Rebecca, and Peter Mell. NIST special publication on intrusion detection systems. BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, 2001.

Bhuyan MH, Bhattacharyya DK, Kalita JK. "A multi-step outlier-based anomaly detection approach to network-wide traffic" Information Sciences 2016; 348: 243–271

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers." Proceedings of the fifth annual workshop on Computational learning theory. ACM, 1992.

Bouzida, Y. & Cuppens, F., 2006. Neural netowrks vs. decision trees for intrusion detection. Proc. IEEE/IST Workshop Monitoring, Attack Detection Mitigation, pp. 28-29.

Carpenter, Gail A., and Stephen Grossberg. "The ART of adaptive pattern recognition by a self-organizing neural network." Computer 21.3 (1988): 77-88.

Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: a library for support vector machines." ACM transactions on intelligent systems and technology (TIST) 2.3 (2011): 27.

Choudhury, S. & Bhowal, A., 2015. Comparative Analysis of Machine Learning Algorithms along with Classifiers for Network Intrusion Detection. International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), 06 May.pp. 89-95

Dhanabal, L., and S. P. Shantharajah. "A study on NSL-KDD dataset for intrusion detection system based on classification algorithms." International Journal of Advanced Research in Computer and Communication Engineering 4.6 (2015): 446-452.

E. Eaton, Active Task Selection for Lifelong Machine Learning, AAAI Spring Symposium, volume SS-13-05 of AAAI Technical Report. AAAI, 2013.

Furao Shen and Osamu Hasegawa. "An incremental network for on-line unsupervised classification and

topology learning." Neural networks 19.1 (2006): 90-106.

Furao Shen, Tomotaka Ogura, and Osamu Hasegawa. "An enhanced self-organizing incremental neural network for online unsupervised learning." Neural Networks 20.8 (2007): 893-903.

F. Shen H. Yu K. Sakurai and O. Hasegawa "An incremental online semi-supervised active learning algorithm based on self-organizing incremental neural network " Neural Comput. Appl. vol.20 no.7 pp.1061-1074 Oct. 2011.

Gepperth, Alexander, and Barbara Hammer. "Incremental learning algorithms and applications." European Symposium on Artificial Neural Networks (ESANN). 2016.

G. Kim S . Lee and S. Kim "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection" Expert Syst. with Appl. vol. 41 no. 4 pp. 1690-1700 2014.

H Debar M Becke D Siboni "A Neural Network Component for an Intrusion Detection System" Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy 1992:240-250

Hsu, Chih-Wei, and Chih-Jen Lin. "A comparison of methods for multiclass support vector machines." IEEE transactions on Neural Networks 13.2 (2002): 415-425.

Ibrahim LM. Anomaly network intrusion detection system based on Distributed Time-Delay Neural Network (DTDNN). J EngSciTechnol 2010; 5: 457-471.

Jawhar, Muna Mhammad T., and Monica Mehrotra. "Design network intrusion detection system using hybrid fuzzy-neural network." International Journal of Computer Science and Security 4.3 (2010): 285-294.

J Ryan M-J Lin R Miikkulainen "Intrusion Detection with Neural Networks" in Advances in Neural Information Processing Systems 10 Cambridge MA:MIT Press 1998 : 943-949.

Kawewong, Aram, Rapeeporn Pimup, and Osamu Hasegawa. "Incremental Learning Framework for Indoor Scene Recognition." Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence. 2013.

Kim, J., Shin, N., Jo, S. Y. & Kim, S. H., 2017. Method of Intrusion Detection using Deep Neural Network. s.l., IEEE, pp. 313-316.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43, 59–69.

Lei Li, De-Zhang Yang, Fang-Cheng Shen, "A Novel Rule-based Intrusion detection System Using Data

Mining". In the Proc. Of 3rd IEEE International Conference on Computer Science and Information Technology, pp. 169-172, 2010.

Liao, H.J., Lin, C.H.R., Lin, Y.C., Tung, K.Y. "Intrusion detection system: A comprehensive review" Journal of Network and Computer Applications 2013; 36 (1):16–24.

Li Y, Xia J, Zhang S, Yan J, Ai X, Dai K: "An efficient intrusion detection system based on support vector machines and gradually feature removal method".Expert Syst. Appl 2012, 39: 424-430. 10.1016/j.eswa.2011.07.032

Marcel van Gerven and Sander Bohte. "Artificial neural networks as models of neural information processing". Frontiers Media SA, 2018

Martinetz, T., & Schulten, K. (1994). Topology representing networks. Neural Networks, 7(3), 507–522.

M. Cova D. Balzarotti V. Felmetsger G. Vigna "Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications" Proc. Int'l Symp. Recent Advances in Intrusion Detection 2007.

M. Moradi and M. Zulkernine. "A Neural Network Based System for Intrusion Detection and Classification of Attacks " Proeeding of 2004 IEEE International Conference on Advances in Intelligent Systems Luxembourg 2004 15-18.

Nejad, Majid Bakhshandeh, and Hamid Reza Shahriary. "A Novel Approach for the Detection of Anomalous User Behavior in Web Applications Processes" 2016 International Academic Journal of Science and Engineering Vol. 3, No. 2, 2016, pp. 56-67. ISSN 2454-3896

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 vol.29 pp. 713-722 2005.

Panda, M. & Patra, M. R., 2009. Evaluating Machine Learning Algorithms for Detecting Network Intrusions. International Journal of Recent Trends in Engineering, 1(1), pp. 472-477.

Robinson, R. R. & Thomas, C., 2015. Ranking of Machine learning Algorithms Based on the Performance in Classifying DDoS Attacks. IEEE Recent Advances in Intelligent Computational Systems (RAICS), 10 December .pp. 185-190.

Russell, Stuart J., and Peter Norvig. "Artificial Intelligence (A Modern Approach)." (2010).

S. M. Bridges and R. B. Vaughn "Fuzzy data mining and genetic algorithms applied to intrusion detection " 23rd National Information Systems Security Conference 2000 : 109-122.

S Mukkamala G Janowski A H. Sung "Intrusion Detection Using Neural Networks and Support Vector

Machines" Proceedings of IEEE International Joint Conference on Neural Networks 2002, pp. 1702-1707 May 2002.

Stolfo, J., et al. "Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection." Results from the JAM Project by Salvatore (2000): 1-15.

Sung, Andrew H., and Srinivas Mukkamala. "Identifying important features for intrusion detection using support vector machines and neural networks." Applications and the Internet, 2003. Proceedings. 2003 Symposium on. IEEE, 2003.

Sutton, Richard S. "Reinforcement learning: Past, present and future." Asia-Pacific Conference on Simulated Evolution and Learning. Springer, Berlin, Heidelberg, 1998.

T. Crothers, Implementing Intrusion Detection Systems: A Hands-On Guide for Securing the Network, Wiley, 2003

Tammi, W. M. et al., 2015. Artificial Neural Network based System for Intrusion Detection using Clustering on Different Feature Selection. International Journal of Computer Applications, September, 126(12), pp. 21-28.

Tavallaee, Mahbod, et al. "A detailed analysis of the KDD CUP 99 data set." Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE, 2009.

Wang, Lipo, ed. Support vector machines: theory and applications. Vol. 177. Springer Science & Business Media, 2005.

Yu, Wei-Yi, and Hahn-Ming Lee. "An incremental-learning method for supervised anomaly detection by cascading service classifier and ITI decision tree methods." Pacific-Asia Workshop on Intelligence and Security Informatics. Springer, Berlin, Heidelberg, (2009): 155 -160 .

Zhu, Xiaojin. "Semi-supervised learning literature survey." Computer Science, University of Wisconsin-Madison 2.3 (2006): 4.

Z. Xiang, Z. Xiao, D. Wang, H.M. Georges "Incremental semi-supervised kernel construction with self-organizing incremental neural network and application in intrusion detection" J. Intell. Fuzzy Syst., 31 (2) 2016, pp. 815-823

Z. Zhang "HIDE: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification" Proc. 2nd An. IEEE Sys. Man Cyber. Info. Assurance Wksp. 2001 : 85 -90.