Cybersecurity for Industry 4.0 and Advanced Manufacturing Environments with Ensemble Intelligence

Lane Thames and Dirk Schaefer

Abstract Traditional cybersecurity architectures incorporate security mechanisms that provide services such as confidentiality, authenticity, integrity, access control, and non-repudiation. These mechanisms are used extensively to prevent computer and network intrusions and attacks. For instance, access control services prevent unauthorized access to cyber resources such as computers, networks, and data. However, the modern Internet security landscape is characterized by attacks that are voluminous, constantly evolving, extremely fast, persistent, and highly sophisticated Schnackenberg et al. (2000), Anuar et al. (2010). These characteristics impose significant challenges on preventive security services. Consequently, methodologies that enable autonomic detection and response to cyberattacks should be employed synergistically with prevention techniques in order to achieve effective defense-in-depth strategies and robust cybersecurity systems. This is especially true for the critical systems belonging to Industry 4.0 systems. In this chapter, we describe how we have integrated cyberattack detection and response mechanisms into our Software-Defined Cloud Manufacturing architecture. The cyberattack detection algorithm described in this chapter is based on ensemble intelligence with neural networks whose outputs are fed into a neuro-evolved neural network oracle. The oracle produces an optimized classification output that is used to provide feedback to active attack response mechanisms within our software-defined cloud manufacturing system. The underlying goal of this chapter is to show how computational intelligence approaches can be used to defend critical Industry 4.0 systems as well as other Internet-driven systems.

L. Thames (✉)
Tripwire Inc., Atlanta, GA, USA
e-mail: lthames@tripwire.com

D. Schaefer
University of Bath, Bath, UK
e-mail: d.schaefer@bath.ac.uk

L. Thames and D. Schaefer (eds.), *Cybersecurity for Industry 4.0*,
Springer Series in Advanced Manufacturing, DOI 10.1007/978-3-319-50660-9_10

1 Cyberattack Detection: Methodologies and Algorithms

Traditional cybersecurity architectures incorporate security mechanisms that provide services such as confidentiality, authenticity, integrity, access control, and non-repudiation. These services are used extensively to prevent computer and network intrusions and attacks. For instance, access control services prevent unauthorized access to cyber resources such as computers, networks, and data. However, the modern Internet security landscape is characterized by attacks that are voluminous, constantly evolving, extremely fast, persistent, and highly sophisticated Schnackenberg et al. (2000), Anuar et al. (2010). These characteristics impose significant challenges on preventive security services. Consequently, methodologies that enable autonomic detection and response to cyberattacks should be employed synergistically with prevention techniques in order to achieve effective defense-in-depth strategies and robust cybersecurity systems Iheagwara et al. (2006), Kabiri and Ghorbani (2005), Ruighaver (2008).

Cyberattack detection systems require algorithms that collect and analyze data generated by various events occurring within a cyber environment. The objective of a detection algorithm is to accurately discover suspicious activities based on the analysis of event data. This objective is fundamentally important as it forms the core of any attack detection system. However, the objective is hard to achieve, especially in terms of ***accuracy***. A detection algorithm that generates inaccurate results can negatively impact the performance of the entire system. Axelsson (2000) claims that the performance of an intrusion detection system, in terms of *effectiveness*, is limited by its false alarm rate. This performance limit is a consequence of the *base-rate fallacy*. For example, inaccurate detection algorithms generate large volumes of false alarms, which can lead to issues such as collateral damage, unnoticed detection of live attacks or intrusions, and unmanageable numbers of alarm notifications that overwhelm security administrators. Consequently, research has explored new algorithms and methodologies aiming to increase the performance and accuracy of detection systems Axelsson (2000), Ghorbani et al. (2010), Anderson (1980), Zhang et al. (2008), Khor et al. (2009).

The study of computational intelligence systems (CIS) is concerned with the theory and design of evolutionary and adaptive systems that possess emergent behavior and intelligent decision making capabilities and that operate within complex and dynamic environments Venayagamoorthy (2011). These systems are generally designed to cope with high dimensional and noisy data during their decision making processes. Since cyberattack detection systems are faced with large volumes of high dimensional data along with continuously evolving attack characteristics, computational intelligence systems have become logical choices to consider when designing new classification algorithms for detection systems.

A computational intelligence algorithm based on new hybridization and ensemble methodologies is presented in this chapter. The algorithms are constructed as generalized systems with no underlying domain-specific assumptions influencing the design. However, the systems are evaluated as classification frameworks for cyberattack and intrusion detection. Before introducing the core detection algorithm, we will first describe how we have integrated the detection system into our software-defined cloud manufacturing architecture. Then, we will provide the reader with a brief overview of neural networks and genetic algorithms. Next, we will describe our ensemble intelligence algorithm and provide details of how we evaluated its performance.

2 Cyberattack Detection and Response Within the Software-Defined Cloud Manufacturing Architecture

We introduced our software-defined cloud manufacturing (SDCM) architecture in Chap. 1. The goal of Chap. 1 was to introduce the reader to a broad array of technologies and paradigms based on the Industry 4.0 vision. The reader should refer back to Chap. 1 for specific details underlying SDCM. In this chapter, we show at a high level how we've incorporated real-time cyberattack detection and response to SDCM. We illustrate the system with Fig. 1. Recall from Chap. 1 that within the SDCM architecture all communications are managed by control elements (CE). SDCM control elements are highly distributed controllers deployed within the cloud. CEs are responsible for interconnecting SDCM entities, whether it is a cloud consumer using an application at the virtual layer that must interact with a hardware device or even if it

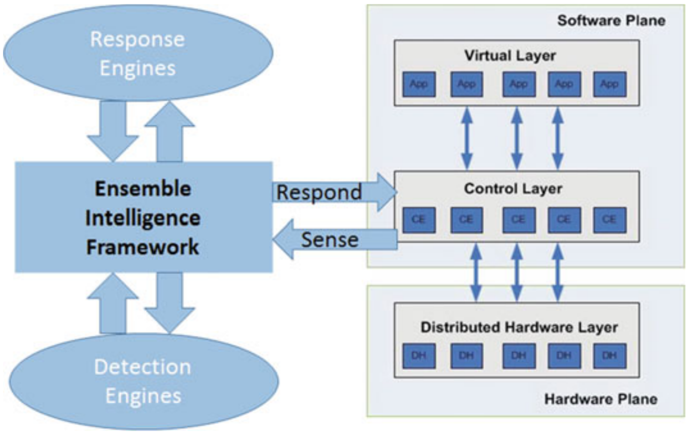

Fig. 1 Software-defined cloud manufacturing architecture with an ensemble intelligence framework for cyberattack detection and response

is multiple hardware devices needing to communicate with each other. Control elements can also communicate with each other to perform their tasks. Essentially, the intelligence of a SDCM system lies within the control layer. The elements of the virtual layer and distributed hardware layer are responsible for getting real design and manufacturing work accomplished. Since the control elements have deep insight into the activities and communications that are taking place, it is logical to use the control elements as data tap points. In our architecture, control elements act as sensors that feed streaming data into the Ensemble Intelligence Framework (EIF). The EIF is responsible for analyzing the sensed data and, when anomalies are detected, it is responsible for responding to the detected anomalies. We use the word anomaly here to reflect that the EIF can be used for other intelligence activities other than cyberattack detection. For example, this could be an intelligent system dedicated to predictive maintenance tasks. However, we are using it here for the purpose of cyberattack detection and response. As the figure illustrates, any number of detection engines can be employed as well as any number of response engines. Response mechanisms can include real-time communication connection termination, installation of new rules in perimeter firewalls, etc. Essentially, response in this context is anything the SDCM system employs for protecting its critical assets from an observed cyberattack.

We have described above at a high level how we have integrated real-time cyberattack detection and response to SDCM via the incorporation of an ensemble intelligence framework. The goal of this chapter is not to go into an in-depth discussion of the framework but instead to describe one possible algorithm that can be used as a detection engine in the framework. In particular, we will describe in the next few sections a neural network ensemble system that utilizes a neuro-evolved network oracle that can be used for analyzing streaming inputs collected by SDCM control elements for the purpose of detecting cyberattacks.

3 Neural Networks and Genetic Algorithms

In this section, we will provide an overview of neural networks and genetic algorithms and how we have used genetic algorithms to produced neuro-evolved neural networks that are capable of very good classification results.

3.1 Neural Networks

The underlying theory of Artificial Neural Networks (ANN or just network if no confusion arises) was originally inspired by biological processes. Specifically, ANNs are modeled after the human central nervous system, which consists of a very sophisticated interconnection of neurons and their associated axons, dendrites, and synapses.

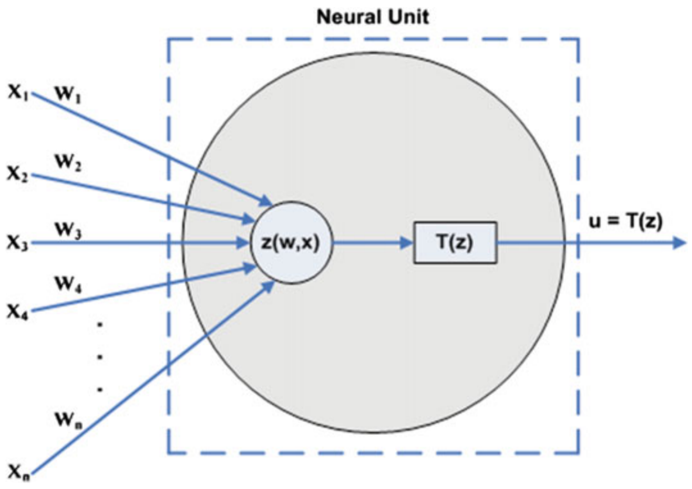

Fig. 2 A conceptual diagram of the neural unit

At the core of an ANN is the neural unit (NU) as shown in Fig. 2. The ANN is created by interconnecting many neural units across several layers to form a highly connected neural network. An NU takes as its input a vector $x = (x_1, x_2, \ldots, x_n)$. Associated with each input connection x_i is a synaptic weight w_i, and these weights form the weight vector w. The output of an NU represents its activation level for a particular set of inputs where the output is denoted by $u = T(z)$. $T(z)$ is the transfer function of the NU (sometimes T is referred to as the activation function). Several forms exist for the transfer function. In this chapter, will only consider three types, which are given by Eqs. 1, 2, and 3. Equation 1 is the logistic sigmoidal function or the logsig function, Eq. 2 is the hyperbolic tangent sigmoidal function or the tansig function, and Eq. 3 is the linear function or purelin function.

$$T(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

$$T(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{2}$$

$$T(z) = z \tag{3}$$

The transfer function's input, z, is the dot product of the input vector with the weight vector as shown by Eq. 4.

$$z = \sum_{i=1}^{n} w_i x_i \tag{4}$$

Networks are created by interconnecting neural units to other neural units formed by one or more hidden layers, where each layer has some prescribed number of units. Networks learn how to map values in the input space to values in the output space via training, and training is provided by a learning algorithm, of which many different forms exist. Common types of ANN learning algorithms are based on the gradient decent algorithm. The basic idea is as follows. Training data is provided to the ANN in the form of $(x,f(x))$ tuples where x is the input data and $f(x)$ is the target function. The learning stage takes the training tuple and sends the input value(s) into the ANN. Then, the output $f_a(x)$ is compared to the target value and an error is calculated. This error is used to evolve the weights such that over time (training epochs) the error of the ANN is minimized to some preferably global but possibly local minimum error. For our work, we utilized networks based on the back-propagation algorithm, whose weight update rule is given by Eq. 5.

$$w_i(t+1) = w_i(t) + \alpha[f(x_i) - f_a(x_i)]x_i \tag{5}$$

During training, each weight vector component for each NU in the ANN is updated similar to Eq. 5. As seen in Eq. 5, as the approximation function approaches the actual function over increasing training epochs, the change in weight value w_i approaches zero such that at convergence $w_i(t+1) \approx w_i(t)$. The value α represents the learning rate, and this value determines how fast the weights evolve. Figure 3 will be used to further illustrate the weight training process. In this example, a single neuron is simulated that is initiated with a weight vector consisting of a randomly generated set of 1000 values. A single target vector with 1000 elements is used as the training data. The algorithm employs a constant learning rate of $\alpha = 0.5$. Figure 3 plots the target vector versus the weight vector as the training epoch is increased from $t = 0$ to $t = 80$. The emergence of a straight line indicates how the weight vector successfully converges towards the target vector with increasing training epochs.

The performance of an ANN is sensitive to the selection of parameters that define its overall configuration. These parameters include, just to name a few, the type of transfer function to use in each layer, the total number of layers, the total number of units per layer, the learning rate's value, the type of training algorithm to use, and the number of training epochs to use. Furthermore, these parameters are not generalized to any given network, and in many cases, they depend on the underlying data's input-output space. If an experienced network designer has a good understanding of the input-output space, then the designer's domain knowledge and expertise allows her to select respectable parameter values. However, this is normally a trial-and-error process even for experienced designers. Further, the problem is more challenging when working with high-dimensional input-output spaces where underlying patterns that drive the selection of parameters are not known. Hence, methods for automated selection of optimized parameters using other computational optimizations sounds promising. In particular, we will investigate the use of genetic algorithms for selecting optimal network parameters.

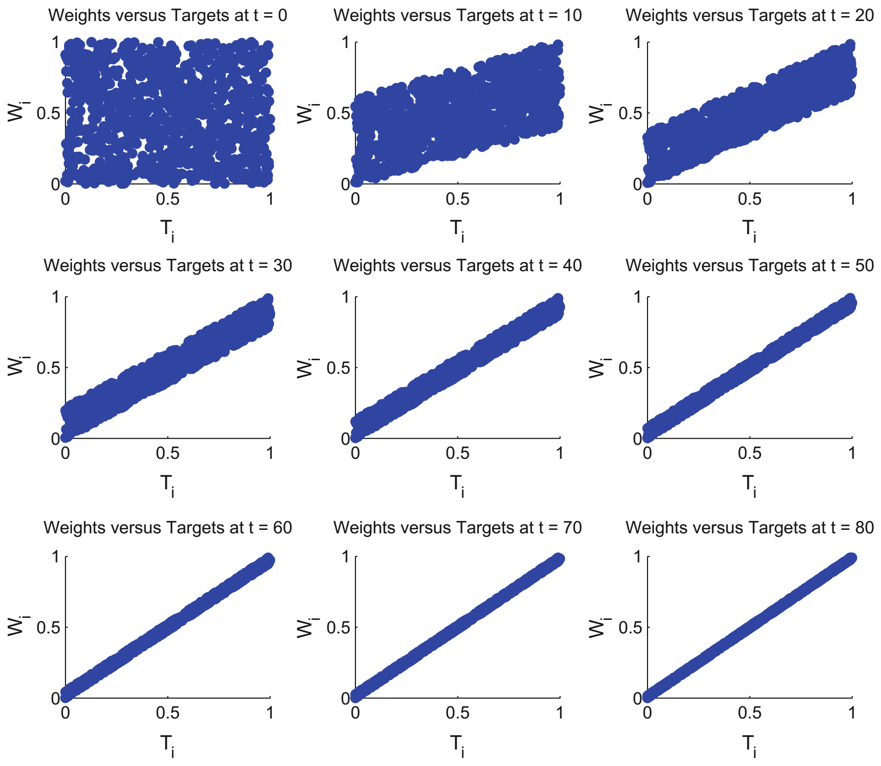

Fig. 3 Learning weights

3.2 Genetic Algorithms

The creation of genetic algorithms (GAs) was inspired by the biological evolutionary process. The primary inspiration is due to the fact that biological systems can adapt over time (evolve) within changing environments. Further, this adaptation can propagate through successor generations within the biological system. This adaptation-propagation scheme leads to the idea of survival of the fittest individuals that can adapt well to changing environments have a higher probability of survival.

The primary operations performed by GA include chromosome representation, genetic selection, genetic crossover, genetic mutation, and population fitness evaluation. In GAs, problem domains are encoded via chromosomes in a population $P(t)$. This chromosome encoding is usually in the form of a bit string or some numerical representation, i.e., one is required to map population members to a binary or numerical form. The population represents a particular state space of hypotheses at evolution time epoch t, where a hypothesis is a possible solution to a given problem. At each time epoch, the fitness of each individual of the population is evaluated. The fitness is evaluated with a fitness function $F(h_i)$ where h_i is the hypothesis represented by the ith member (chromosome) of the population and the fitness F represents how well a particular hypothesis represents the solution of the given problem.

In general, the GA's fitness function must be an increasing function with respect to a candidate hypothesis's response to the problem such that good solutions have higher fitness and poor solutions have low fitness. F is computed for each member, and the next population $P(t + 1)$ is created by probabilistically selecting the most fit members of the current population. Some of the members will be part of $P(t + 1)$ in their current form, and some are selected for genetic modifications such as crossover and mutation. Crossover produces offspring from two parents whereas mutation is the act of randomly modifying the encoding features of a selected set of individuals. There are two important design issues when using a GA. First, one must define a mapping from the input-output space of the problem into an encoding that can be used by the GA, i.e., a binary or numerical mapping. Second, one must design a fitness function for the problem domain. The power of the GA is in its ability to encode a very large set of possible solution spaces for a given problem. They are often used successfully for optimization problems, but they have also been used for function approximation, complex circuit layout, and scheduling. They are also used in neuroevolution to evolve neural networks. In this work, the GA will be used to optimize a certain set of ANN design parameters.

4 Cyberattack Detection with Ensembles of Computational Intelligence Systems

The performance of an artificial neural network is sensitive to the selection of parameters that define its overall configuration. Some of these parameters include the transfer function used within each layer, the total number of layers, the total number of units per layer, the learning rate, the training algorithm, and the number of training epochs to use. Furthermore, these parameters are not generalized to any given network and depend on characteristics of the classification data. Appropriate selection of neural network parameters is typically a trial-and-error process whereby the designer seeks the set of parameters that minimize classification error produced by the network. Optimization algorithms that autonomically tune the parameters of artificial neural networks can alleviate the trial-and-error parameter selection process and can lead to neural networks with better classification accuracy.

In this section, a classification algorithm for attack detection based on ensembles of neural networks is described. The novelty of the algorithm stems from the methodology employed for combining outputs of neural network ensembles. Particularly, a neural network oracle is utilized to combine the ensemble outputs. The neural network oracle is constructed with a genetic algorithm that finds a set of configuration parameters that produce high classification accuracy and low classification error.

4.1 The NNO Classification Algorithm

The proposed algorithm referred to as NNO employs a genetic algorithm to find an optimal selection of configuration parameters for a neural network oracle, which is responsible for combining the outputs of an ensemble of neural networks that classify features belonging to audit data for the cyberattack detection problem. The overall idea is illustrated by Fig. 4 and described as follows.

A set of artificial neural networks $\eta = \{\eta_i\}$ is assigned to features of the collection of labeled audit data that describe a classification domain. The collection of ANNs are trained with standard procedures. Once the collection has been trained, the training data is used to generate a secondary set of training data. The secondary set of training data contains the output of each ANN along with the actual output defined by the baseline training data. The secondary training data is then used to train the neural network oracle. However, the oracle uses a genetic algorithm to find the set of configuration parameters that minimizes its error.

The algorithm consists of two primary phases. During phase 1, a GA is constructed that contains a population of chromosomes that are numerical representations of ANN configuration parameters. At each evolution time epoch, t, the chromosome for each population member is submitted to the ANN. The ANN maps the chromosome's numerical values to their respective parameter types, implements a self-configuration based on these values, and then learns from a training set. Once the

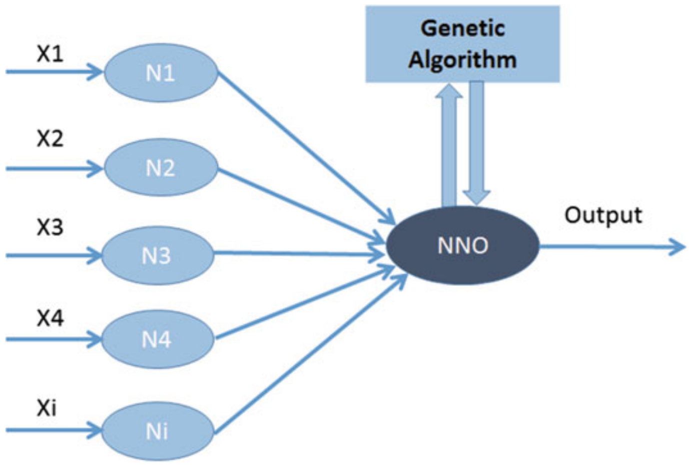

Fig. 4 Architectural illustration of the NNO algorithm

ANN has been trained, a set of labeled validation data from the input-output space is used to evaluate the ANN's post-training error response. This error response is then used to evaluate the fitness of the population member whose chromosome was submitted to the ANN for configuration and training. Since the goal is to find an ANN with minimal error, the error response, which is given by Eq. 6, is used as input to the fitness function of the genetic algorithm.

$$E_i = \sqrt{\sum_{j=1}^{N} \left(f(x_j) - f_a(x_j)\right)^2} \tag{6}$$

In Eq. 6, E_i is the error of the ANN configured and trained based on the chromosome h_i of the ith population member. $f(x)$ is the value of the target function for input x, whereas $f_a(x)$ is the approximation of $f(x)$ produced by the ANN. The error is calculated over a total of N evaluations from a validation dataset. The fitness function for the system is given by Eq. 7.

$$F(h_i; E_i) = \frac{1}{E_i} \tag{7}$$

The fitness function is inversely proportional to the error of the ANN configured by parameters represented by the ith chromosome (i.e., the ith hypothesis h_i) of the GA's population. With the fitness function of Eq. 7, a decrease in error produced by an ANN configured via h_i produces an increase in fitness, which is the underlying objective of the algorithm.

The steps described above are performed for each member of the GA's population. Once each member in the population has been evaluated for fitness, the GA performs selection, crossover, and mutation operations and then proceeds to the next evolution epoch, $t + 1$. This entire process proceeds until the evolution process terminates.

During phase 2, which proceeds after the simulated evolution process terminates, the GA submits the chromosome from the terminal population's best fit individual to the ANN. The ANN uses this chromosome to configure its parameters and then trains from a set of phase-2 training data. Once this training is complete, the system is ready to be deployed for its target application.

5 Datasets and Performance Metrics for Evaluating Cyberattack Detection Systems

Appropriate datasets for training and testing classification algorithms along with reliable metrics to evaluate classification performance are needed for the design of effective cyberattack detection systems. This section discusses the datasets and performance metrics used to evaluate the classification algorithms proposed in this chapter.

5.1 Datasets

Datasets containing relevant features that characterize cyberattacks are needed for the design, evaluation, and comparative analysis of new classification algorithms for attack detection systems. However, datasets and testing environments for evaluating attack detection systems are rare Athanasiades et al. (2003). Of the few datasets publicly available, the ones most frequently used by researchers were produced by the DARPA intrusion detection evaluation program Tavallaee et al. (2010). The objective of the DARPA intrusion detection evaluation program was to produce a collection of standardized datasets that could be used to formally evaluate and objectively compare the performance of intrusion detection systems Lippmann et al. (2000). The datasets have played a critical role in the advancement of intrusion detection systems along with the development of new attack detection and classification algorithms. The DARPA datasets were collected at MIT Lincoln Laboratories during the years 1998, 1999, and 2000. These datasets contain various types of *audit* data with features representing normal and attack traffic.

The KDD CUP99 dataset, which was used as a benchmark for the Third International Knowledge Discovery and Data (KDD) Mining Tools Competition, is frequently used to evaluate intrusion and attack detection algorithms. The CUP99 dataset is a derivative of the 1998 DARPA dataset (DARPA98). DARPA98 contains audit data generated by simulated background traffic representing *normal* packet flows between a military network and the Internet along with traffic representing *attack* packet flows.

The CUP99 data are viewed as sequences of connection records representing unique packet flows. A flow can be defined, similar to the packet filters of a firewall, by specifying a matching criteria over some set of header fields. The canonical flow is specified by a 5-tuple containing source IP address, destination IP address, source port, destination port, and protocol type. The records are classified by two types of flows: attack flows or normal flows. The attack flows are further categorized by 24 unique attack types.

Each record of the CUP99 dataset contains 42 fields. One field provides a label specifying the record's flow type, i.e., normal or attack type. The remaining 41 fields are comprised of features representing data extracted from the flow. The features are categorized as basic TCP features, content features, network-based traffic features, and host-based traffic features. The features are encoded numerically or symbolically. The goal of the KDD competition was to use the CUP99 dataset as a benchmark for evaluating attack classification algorithms produced by the various competitors of the KDD mining tools competition.

The DARPA and CUP99 datasets are out of date. Further, these datasets have been criticized by several whose investigations have discovered various limitations and deficiencies of the data McHugh (2000), Tavallaee et al. (2009). However, the datasets remain widely used for testing and evaluation of attack detection and classification algorithms because there are no suitable alternatives currently available Perdisci et al. (2009), Engen (2010).

In this chapter, classification algorithms for the cyberattack detection problem are introduced. Each algorithm was evaluated with the CUP99 dataset. Particularly, the well-known 10% CUP99 dataset was employed.

5.2 Performance Metrics

The ***accuracy*** of a classification algorithm is a key performance indicator that determines the algorithm's suitability for solving a particular problem. However, other performance indicators are commonly measured in conjunction with accuracy. For example, true positive rates and true negative rates measure a classifier's capability to correctly distinguish *positive* cases from *negative* cases. Classification problems based on two-class decision spaces can use positive and negative rates as performance measures. However, the positive and negative classes must be defined when designing the classifier. Many researchers who design algorithms for intrusion and attack detection problems define attack instances as the positive class and normal instances as the negative class. This is especially true for classifiers implementing anomaly detection. Anomaly detection is based on audit data that represents normal instances and instances deviating from the characteristics underlying the audit data are assumed to be anomalous and, by definition, indicative of an attack. Deviations indicate 'positively' that the instance is an anomaly. Hence, classification of an anomaly is defined to be positive whereas classification of a normal instance is defined to be negative, i.e., not anomaly.

The new classification algorithms proposed in this chapter for the attack detection problem were trained and tested with the CUP99 dataset. CUP99 contains records representing audit data that characterize both normal instances (normal traffic flows) and attack instances (attack traffic flows). Moreover, the dataset is comprised of a multiplicity of attack types. Consequently, CUP99 can be used to evaluate detection systems based on two-class or multi-class classification algorithms. The algorithms proposed in this chapter are designed as two-class classification systems.

Since CUP99 contains audit data characterizing normal and attack instances, several design scenarios can be considered. The data can be partitioned into normal classes for the design of anomaly detection, into attack classes for misuse detection, or the data can remain un-partitioned for mixed detection. The classification algorithms presented in this chapter were trained and tested as two-class mixed detection (non-partitioned) methodologies.

The mixed detection approach with two-class (binary) classification enables two perspectives for defining the positive and negative classes. These perspectives are illustrated by Table 1. The perspective defining the normal class to be positive and the attack class to be negative was chosen for the work described in this chapter.

Table 1 Perspectives of the positive and negative classes for mixed detection binary classification

Positive	Negative
Normal	Attack (NOT normal)
Attack	Normal (NOT attack)

Table 2 Basis parameters of the performance metrics defined in Table 3

Symbol	Description
TP	Total number of true positives
TN	Total number of true negatives
FP	Total number of false positives
FN	Total number of false negatives

Table 3 Definitions and nomenclature of the performance metrics used to evaluate the proposed classification algorithms

Name	Symbol	Calculation
Accuracy	ACC	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$
Positive prediction rate	PPR	$PPR = \frac{TP}{TP+FP}$
Negative prediction rate	NPR	$NPR = \frac{TN}{TN+FN}$
False discovery rate	FDR	$FDR = \frac{FP}{TP+FP}$
False positive rate	FPR	$FPR = \frac{FP}{TN+FP}$
False negative rate	FNR	$FNR = \frac{FN}{TP+FN}$

Several metrics are commonly used when evaluating the performance of classification algorithms. Classification accuracy is a key performance indicator of classification systems. However, accuracy measurements alone do not provide complete information for comparative analysis and optimization purposes. Other key performance indicators include metrics such as error rates, true/false positive/negative rates, and predictive rates.

Evaluations of the new algorithms introduced in this chapter where made with the basis parameters shown in Table 2 and the performance metrics shown in Table 3.

5.3 NNO Ensemble Intelligence: Simulation Results

The NNO ensemble methodology investigated for the cyberattack detection problem used a neural network oracle parametrically optimized with genetic algorithms as described above. Although several parameters can be considered for the optimal response of the NNO, the evaluations described in this section were based on an

NNO configured with two hidden layers of nodes. The optimal parameters that were determined with the genetic algorithm included the number of neural units (nodes) to use for the first and second hidden layers, the type of transfer functions to use in the hidden layers, and the type of transfer function to use for the output layer.

The CUP99 dataset was used to evaluate the performance of the proposed system. An ensemble of 41 neural networks was created. Each feature of the CUP99 dataset was assigned to a single member of the ensemble. A subset of the CUP99 dataset was extracted for training. Each neural network was trained with respect to its corresponding feature. After the ensemble components were trained, a secondary training set was produced by collecting the output of each neural network in the ensemble over the phase-1 testing data and augmenting these outputs with the target class associated with each corresponding training vector. This secondary set of training data was then used to train the NNO. The NNO was parametrically optimized with a genetic algorithm as described above. Once the genetic algorithm converged to a best fit candidate representing the configuration parameters that minimized the NNO's error response with respect to the training data, the NNO was configured and trained via these parameters.

The simulation methodology is described as follows. The CUP99 dataset was partitioned into two disjoint sets comprising training data and testing data. The training procedure was described above. For evaluation, testing proceeded as follows. A total of 500 trials was performed. For each trial, a random selection of records were

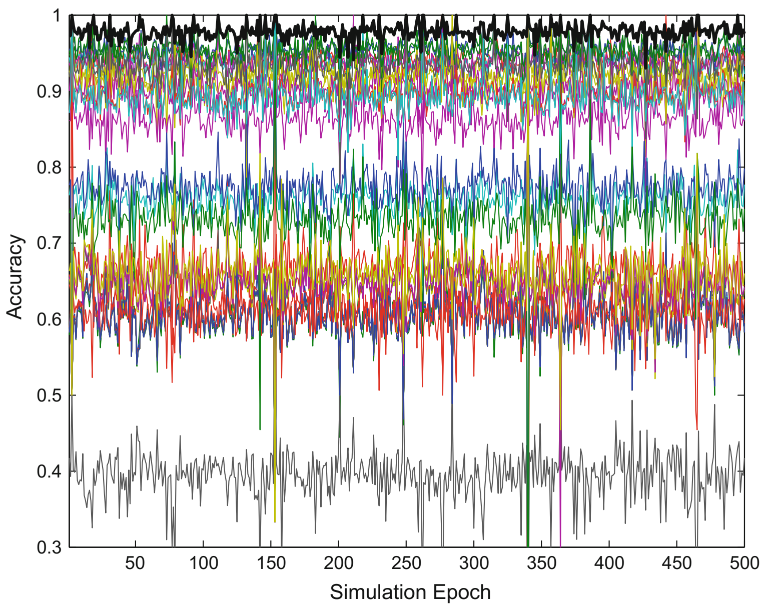

Fig. 5 Accuracies of the individual ensemble members and the neural network oracle over the 500 trials

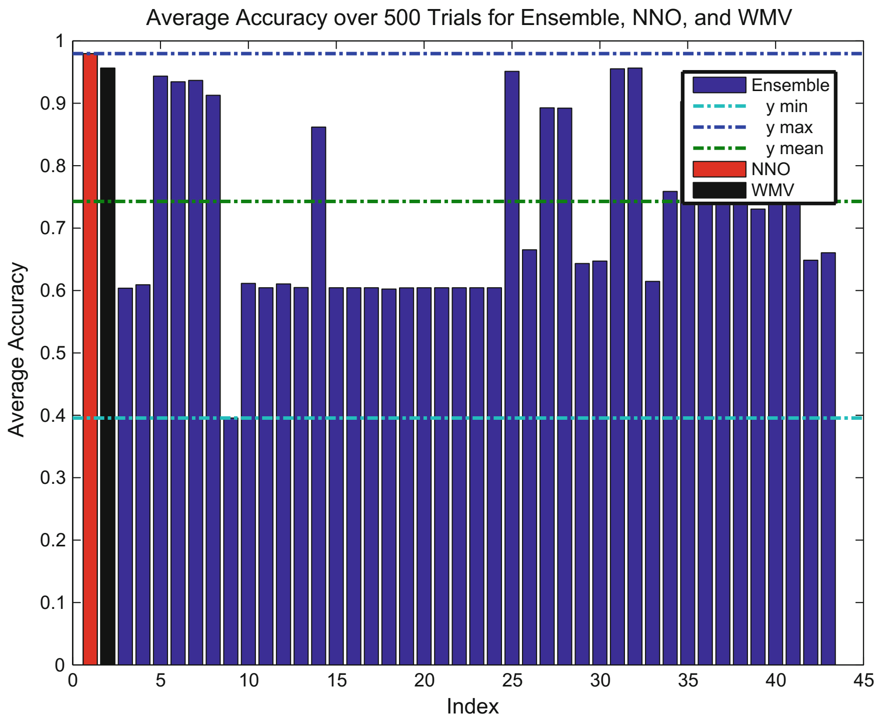

Fig. 6 Bar chart revealing the average accuracies produced over the 500 trials

selected from the test data, and performance metrics of the algorithm were computed and stored as average values. Moreover, the performance of the proposed algorithm was compared to that of the weighted majority vote (WMV) algorithm.

Figure 5 plots the average accuracy over 500 trials for each of the 41 neural networks of the ensemble along with the accuracy of the NNO. The accuracy of the NNO is highlighted by the thick black line towards the top of the figure. Two things should be noted from Fig. 5. First, the range of accuracies reveals that the system has diversity, which is a fundamental requirement for the design of ensemble systems. The accuracies indicate that some of the networks are poor classifiers, some are decent classifiers, and some are good classifiers. Second, the NNO performs consistently better than any given member of the ensemble.

Figure 6 provides a bar chart showing the average of the accuracies produced over the 500 trials for each neural network of the ensemble along with the NNO and WMV. As seen from the figure, the NNO outperforms each of the ensemble members as well as the WMV algorithm.

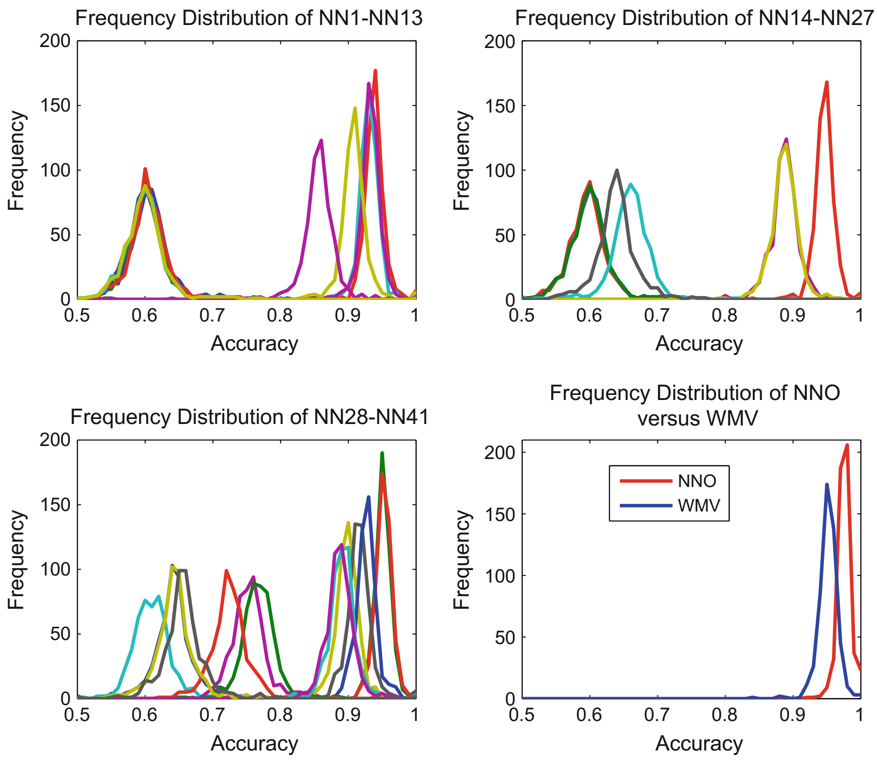

Fig. 7 Frequency distribution of the accuracies for each system over 500 simulation trials

The frequency distributions of the various metrics calculated over the 500 trials for the ensemble components, the NNO method, and the WMV method were generated in order to provide a finer-grained perspective of the performance of each system. A bin width of 0.01 was used to calculate the frequency distributions. Each of the following figures provide metrics for ensemble members, NNO, and WMV. The ensemble results are provided for completeness and for illustrating diversity of the system. However, comparing the performance of NNO versus WMV is the main objective.

Figure 7 plots the frequency distributions for classification accuracies. As seen by the lower right plot, NNO has better accuracy than WMV. Figure 8 plots the frequency distributions for positive prediction rates (PPR) for each classification algorithm. PPR measures how well a classification algorithm predicts the positive class correctly. As seen in the figure, NNO has a better PPR than WMV.

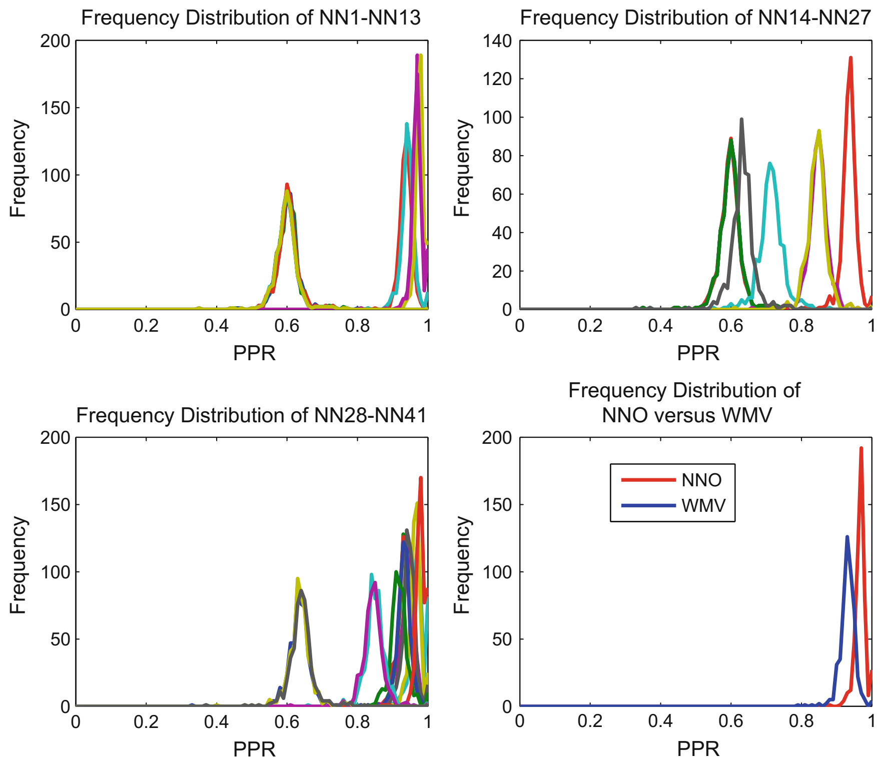

Fig. 8 Frequency distribution of the PPR for each system over 500 simulation trials

Figure 9 plots the negative prediction rate (NPR) of the algorithms. NPR measures how well a classification algorithm correctly classifies negative instances. For NPR, NNO and WMV both perform well and similarly. Figure 10 provides the false discovery rates (FDR) produced by the simulations for the classification algorithms. FDR should be small for good classification algorithms. As seen in the figure, NNO has better FDR performance than WMV.

Figures 11 and 12 plot the frequency distributions for the false positive rates (FPR) and false negative rates (FNR), respectively. Similar to FDR, a good classification algorithm should have small FPR and FNR. As seen in Fig. 11, NNO has better FPR performance than WMV. However, NNO and WMV perform comparably for FNR.

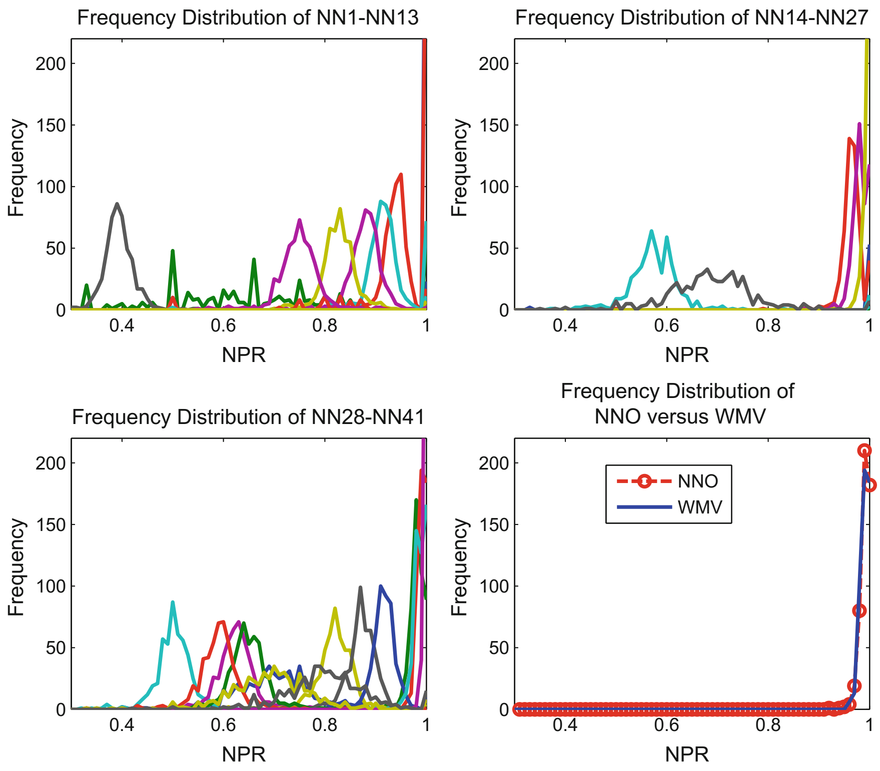

Fig. 9 Frequency distribution of the NPR for each system over 500 simulation trials

Based on the results obtained from simulations along with analysis of its performance metrics, the proposed ensemble methodology using a parametrically optimized neural network oracle provides good performance as a classification system for the cyberattack detection problem.

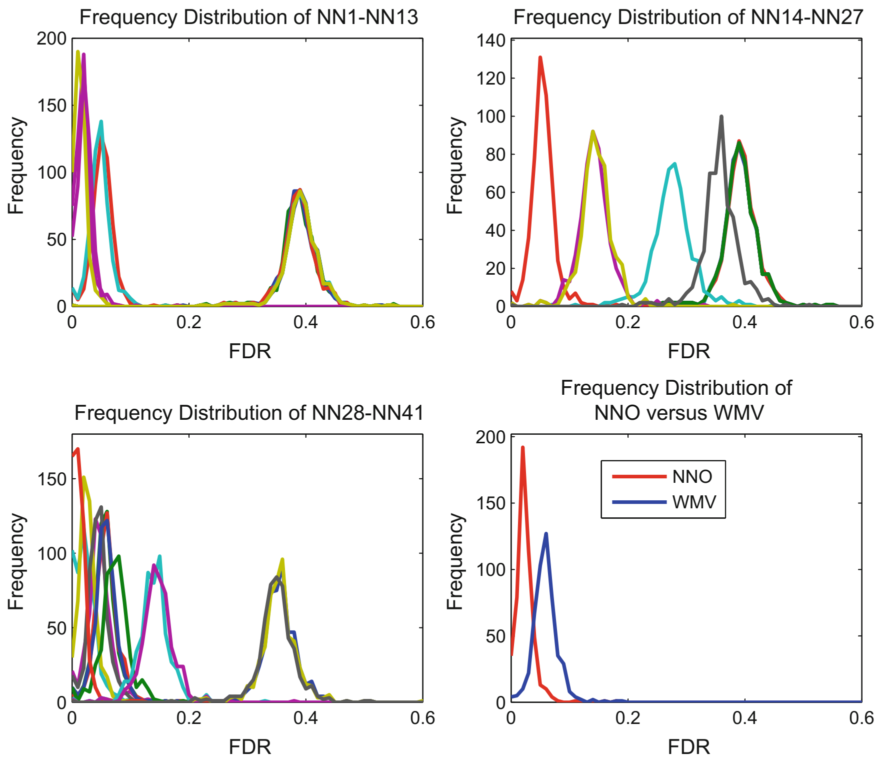

Fig. 10 Frequency distribution of the FDR for each system over 500 simulation trials

6 Summary

In this chapter, an algorithm for cyberattack detection for Internet-based systems such as Industry 4.0 systems was introduced. The algorithm, referred to as NNO, is based on an ensemble of neural networks along with a neural network oracle that has its configuration parameters optimized by genetic algorithms using a fitness function evaluated with neural network error responses. The performance evaluation of the

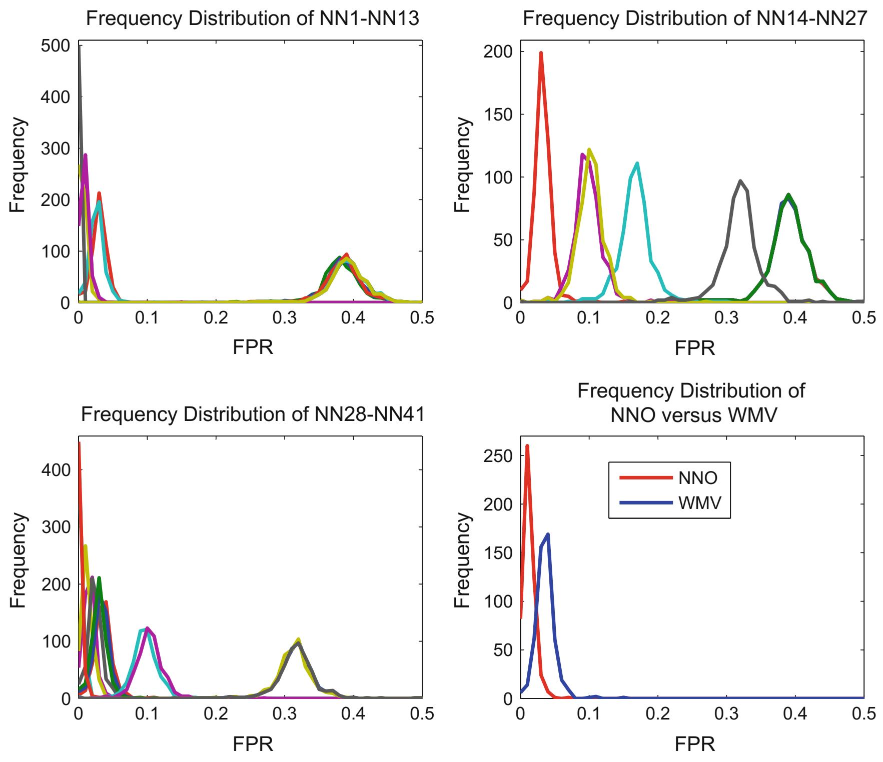

Fig. 11 Frequency distribution of the FPR for each system over 500 simulation trials

proposed algorithm was based on the CUP99 intrusion detection dataset. According to the simulation results obtained, the algorithm was shown to provide good classification performance when trained and tested with the CUP99 intrusion detection dataset. The algorithm can be used to successfully detect cyberattacks targeting Industry 4.0 systems and can be coupled with active response mechanisms in order to stop real cyberattacks.

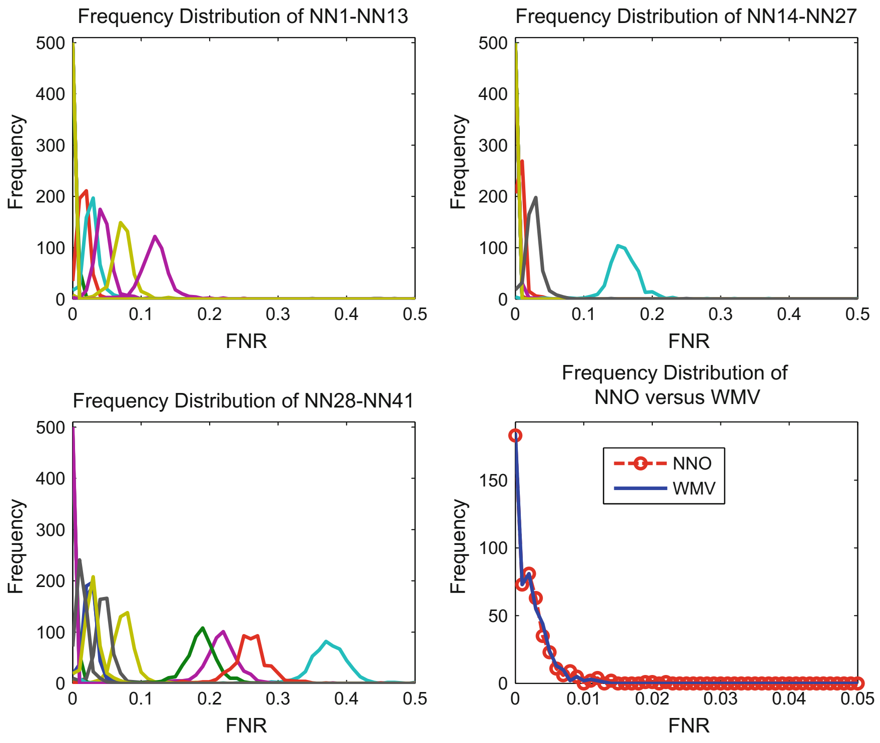

Fig. 12 Frequency distribution of the FNR for each system over 500 simulation trials

References

Anderson J (1980) Computer security threat monitoring and surveillance

Anuar NB, Papadaki M, Furnell S, Clarke N (2010) An investigation and survey of response options for intrusion response systems (IRSs). In: Information security for south africa (ISSA)

Athanasiades N, Abler R, Levine J, Owen H, Riley G (2003) Intrusion detection testing and benchmarking methodologies. In: Proceedings of the first IEEE international workshop on information assurance (IWIA'03)

Axelsson S (2000) Intrusion detection systems: a survey and taxonomy. Technical Report, Department of Computer Engineering, Chalmers University of Technology

Axelsson S (2000) The base-rate fallacy and the difficulty of intrusion detection. ACM Trans Inf Syst Secur 3(3):186–205

Engen V (2010) Machine learning for network based intrusion detection. PhD Thesis, Bournemouth University

Ghorbani AA, Lu W, Tavallaee M (2010) Detection approaches. Springer, J Network Intrusion Detection and Prevention

Hatch M (2014) The maker movement manifesto, McGraw-Hill Education. ISBN 10:0071821120

Iheagwara C, Awan F, Acar Y, Miller C (2006) Maximizing the benefits of intrusion prevention systems: effective deployment strategies. In: Proceedings of the 18th annual forum of incident response and security teams (FIRST) conference

Kabiri P, Ghorbani A (2005) Research on intrusion detection and response: a survey. Int J Netw Secur 1(2):84–102

Khor KC, Ting CY, Amnuaisuk SP (2009) From feature selection to building of bayesian classifiers: a network intrusion detection perspective. Am J Appl Sci 6(11):1949–1960

Knapp E, Langill J (2015) Industrial network security: securing critical infrastructure networks for smart grid, SCADA, and other industriaal control systems, 2nd edn. ISBN 978-0-12-420114-9

Li BH, Zhang L, Wang SL, Tao F, Cao JW, Jiang XD et al. (2010) Cloud manufacturing: a new service-oriented networked manufacturing model. Comput Integr Manuf Syst 16(1):1–7

Lippmann R, Haines J, Fried D, Korba J, Das K (2000) The 1999 DARPA off-line intrusion detection evaluation. Comput Netw 34(4):579–595

McHugh J (2000) Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory. ACM Trans Inf Syst Secur 3(4):262–294

NIST Special Publication 800-82 (2011) Guide to industrial control systems (ICS) security. http://csrc.nist.gov/publications/nistpubs/800-82/SP800-82-final.pdf

Open Networking Foundation (ONF) (2012) Software-defined networking: the new form for networks

Paul Brody (2013) Get ready for software-defined supply chain. Web: http://www.supplychainquarterly.com/topics/Manufacturing/20140110-get-ready-for-the-software-defined-supply-chain/

Perdisci R, Ariu D, Fogla P, Giacinto G, Lee W (2009) McPAD: A multiple classifier system for accurate payload-based anomaly detection. Int J Comput Telecommun Netw 53(6):864–881

Peterson A, Schaefer D (2016) Social product development: introduction, overview, and current status, In: Schaefer D (ed) Product development in the socio-sphere: game changing paradigms for 21st century breakthrough product development and innovation. Springer pp 63–98. ISBN 978-3-319-07403-0

Ruighaver A (2008) Organisational security requirements: an agile approach to ubiquitous information security. In: Proceedings of the sixth australian information security management conference

Schaefer D, Thames JL, Wellman R, Wu D, Yim S, Rosen D (2012) Distributed collaborative design and manufacture in the cloud motivation, infrastructure, and education. ASEE 2012 annual conference and exposition, San Antonio, Texas, June pp 10–13

Schnackenberg D, Djahandari K, Sterne D (2000) Infrastructure for intrusion detection and response. In: Proceedings of the 2000 DARPA information survivability conference and exposition

Tavallaee M, Stakhanova N, Ghorbani A (2010) Toward credible evaluation of anomaly-based intrusion-detection methods. IEEE Trans Syst Man Cybern Part C: Appl 40(5):516–524

Tavallaee M, Bagheri E, Lu W, Ghorbani A (2009) A detailed analysis of the KDD CUP 99 data set. In: Proceedings of the second IEEE international conference on Computational intelligence for security and defense applications, IEEE Press

Thames JL, Abler R, Hyder A, Wellman R, Schaefer D (2011) Architectures and design methodologies for scalable and sustainable remote laboratory infrastructures. In: Azad A, Judson (ed) Internet accessible remote laboratories: scalable e-learning tools for engineering and science disciplines. IGI Global Publishing, ISBN 978-1-61350-186-3, Chapter 13, pp 254–275

Thames JL (2014) Distributed, collaborative, and automated cyber security infrastructures for cloud-based design and manufacturing systems. In: Schaefer D (ed) Cloud-based design and manufacturing (CBDM): a service-oriented product development paradigm for the 21st century. Springer, pp 207–229. ISBN 978-3-319-07398-9. doi:10.1007/978-3-319-07398-9_8

Venayagamoorthy G (2011) Dynamic, stochastic, computational, and scalable technologies for smart grids. IEEE Comput Intell Mag 6(3):22–35

Wu D, Greer MJ, Rosen DW, Schaefer D (2013) Cloud manufacturing: strategic vision and state-of-the-art. J Manuf Syst

Wu D, Thames JL, Rosen D, Schaefer D (2012) Towards a cloud-based design and manufacturing paradigm: looking backward, looking forward. ASME 2012 international design engineering technical conference and computers and information in engineering conference (IDETC/CIE), Chicago, Illinois, August pp 12–15

Wu D, Thames JL, Rosen D, Schaefer D (2013) Enhancing the product realization process with cloud-based design and manufacturing systems. ASME J Comput Inf Sci Eng (JCISE) 13(4)

Xu X (2012) From cloud computing to cloud manufacturing. Rob Comput Integr Manuf 28(1):75–86

Zhang J, Porras P, Ullrich J (2008) Gaussian process learning for cyber-attack early warning. In: Proceedings of the SIAM international conference on data mining