Smart Intrusion Detection with Expert Systems

Flora Amato[1(✉)], Francesco Moscato[2], Fatos Xhafa[3], and Emilio Vivenzio[1]

[1] Department of Electrical Engineering and Information Technology, University of Naples Federico II, Naples, Italy
{flora.amato,emilio.vivenzio}@unina.it
[2] Department of Scienze Politiche, University of Campania "Luigi Vanvitelli", Caserta, Italy
francesco.moscato@unicampania.it
[3] Department of Computer Science, Universitat Politècnica de Catalunya, Barcelona, Spain
fatos@cs.upc.edu

Abstract. Nowadays security concerns of computing devices are growing significantly. This is due to ever increasing number of devices connected to the network. In this context, optimising the performance of intrusion detection systems (IDS) is a key research issue to meet demanding requirements on security of complex and large scale networks. Within the IDS systems, attack classification plays an important role. In this work we propose and evaluate the use the generalizing power of neural networks to classify attacks. More precisely, we use multilayer perceptron (MLP) with the back-propagation algorithm and the sigmoidal activation function. The proposed attack classification system is validated and its performance studied through a subset of the DARPA dataset, known as KDD99, which is a public dataset labelled for an IDS and previously processed. We analysed the results corresponding to different configurations, by varying the number of hidden layers and the number of training epochs to obtain a low number of false results. We observed that it is required a large number of training epochs and that by using the entire data set consisting of 31 features the best classification is carried out for the type of Denial-Of-Service and Probe attacks.

1 Introduction

Today computer networks, from IoT devices to servers and large data centers, have a widespread distribution and services, which has inevitably led to multiple security problems for the computing devices connected to a network, data storage devices, etc. Many security problems arise when computer network systems are attacked to get illegal access. The IDS systems aim to protect the accessibility to the system, its integrity and confidentiality of data. In order to detect attacks against information systems, and therefore act against them to protect the systems, there have been done considerable research efforts and have been

F. Xhafa et al. (Eds.): 3PGCIC 2018, LNDECT 24, pp. 148–159, 2019.
https://doi.org/10.1007/978-3-030-02607-3_14

developed tools (hardware and/or software), among which the IDS (*Intrusion Detection System*) [4,11,17] are very important.

There are basically two types of IDS systems:

- **NIDS** (Network IDS): this type of IDS analyse packets passing through the network and look for the "*signatures*" (set of conditions) in the network traffic, comparing them with those in a database [16] for alert classification.
- **HIDS** (Host-Based IDS): this type of IDS operate directly on a machine by monitoring the operating system through its log file system and hard drives, to detect intrusions, malicious activity or policy violations.

On the other hand, for NIDS case, there have been proposed various techniques, which can be grouped into main ones [16]:

- **Pattern Matching Based**: Techniques in this group determine intrusions by comparing the activity with known signatures [2,3]. These techniques enable to achieve *low* false positive rate [13,18] but do not contribute to detecting new attacks.
- **Anomaly Detection Based**: Techniques in this group (see e.g. [23,24]) determine intrusions by looking for anomalies in network traffic. They achieve a *high* rate of false positives but are able to identify new attacks, not yet in the database.

There exist, in the literature, many interesting artificial intelligence and machine learning approaches for solving problems related to the intrusion detection of general purpose or limited to a single class of anomalies (with decision trees, Bayesian classifiers, multilayer perceptron). There are numerous references, the reader is referred to [1,5,7,10,15] for some relevant approaches.

On the other hand, existing IDS, like Snort[1], an open-source software, show limitations with regard to the understanding of the attacks, closely related to the "*signatures*". In fact, they are not able to acquire new knowledge, unless the system administrator updates the definitions, just as conceptually happens with anti-viruses. So, if an unknown attack occurs, although it may differ slightly from a present in definitions, the IDS will not be able to spot that.

In order to overcome such limitations, in this paper we propose to use a multilayer perceptron (MLP) with the back-propagation algorithm and the sigmoidal activation function for attack classification. An MLP network is able to identify intrusion into the system, in both known and unknown forms, and to reduce false alarms, if properly trained with a series of examples. The proposed attack classification system based on MLP is validated and its performance studied through a subset of the DARPA dataset, known as KDD99, which is a public dataset labelled for an IDS and previously processed. We analysed the results corresponding to different configurations, by varying the number of hidden layers and the number of training epochs to obtain a low number of false results.

The rest of the paper is structured as follows. In present an overview of an MPL network in Sect. 3. In Sect. 4, we describe the KDD99 data set used for

[1] https://www.snort.org/.

validation as well as processing, feature selection and training. Some computational results based on the obtained training configurations are summarised in Sect. 5. We end the paper in Sect. 6 with some conclusions and indications for future work.

2 Intrusion Detection for IoT and Edge Computing

Internet is of course a mean to rapidly retrieve needed information and data, but something is changing in classical management of information. Recent analyses have estimated that the traffic generated by mobile devices and smart sensors, will overcome traffic by Personal Computers and "old" nodes on Internet. The new scenario, known as Internet of Things (IoT), mainly include many "things" (i.e. sensors, actuators, mobile smart devices, smart phones etc.) that communicate on proper networks, with proper protocols, both providing services to (human) users and to other "things". When dealing with IoT, consider that recent studies showed, in healthcare use case with more than 30 millions of users, a data throughput with more than 25.000 records per second. The throughput reaches easily millions of records per seconds in other hi-tech scenarios like smart cities.

One potential way to face the problem of managing this huge amount of data, is to bring communications (and networks), storage and processing units closer to devices. This solution was commonly addressed as Edge Computing. The solution is the integration of concepts both from Edge and Cloud computing. The result is Fog Computing and Networking. Fog Computing is a distributed paradigm that provides Cloud capabilities at network edges.

Besides sensors property of providing low traffic rate, large scale scenarios involve thousands and thousands sensors and smart devices. Big Data management requires both connections among smart devices and the use of proper Fog resources in order to store and manage (possibly at real-time) the huge volume of data generated by sources. In this case, Fog has to provide complex networking functionalities to link clusters of smart devices each other and to connect them to (virtual) computing and storage nodes in the "classical Cloud" parts of Fog infrastructures. Fog networks not only provide connectivity, but many functionalities and Quality of Service too, like security, throughput, limited latency etc.

In this scenario, the introduction of an Intrusion Detection System in IoT is appealing as Fog services. We should provide a mean to collect networking information and to propagate it into the Cloud, in order to perform a fast and effective analysis of network traffic. The intrusion detection system we want to realize, must be fast enough to provide a (quasi) real-time management of traffic. The introduction of Machine Learning and NN at Cloud level in an IoT/Edge architecture allows for proper analyses within acceptable deadlines.

The process of discovering insights and hidden and not trivial evidences of intrusion allows for predictions on the base of data we are handling. Anyway, all mining should be executed in a Fog environment, moving and scheduling computational load closer to data. This is a problem since data may be distributed and

can even be migrated as consequence of mobility. Data access and routing have to be defined properly in IoT where data storage is distributed and where analyses should execute at real time. In our approach, we collect useful information (i.e.: IDS features) from data. The used approach exploits Big Data Cleaning framework to take into account only of needed features. Then, Machine Learning services in the Cloud use extracted features for detection purposes.

In this sense, our approach enables IoT devices (and smart devices too) to benefit from CLoud resources and services in order to exploit services they will never be able to execute (like and IDS service).

3 The Multilayer Perceptron Networks

In this section we start by recalling some basic concepts about MPL networks. Multilayer perceptrons networks have been introduced to cope with the limitations of single-layer perceptrons. An eraly results by Minsky and Papert [14], demonstrated how a simple exclusive OR (XOR), which is a classification problem but not linearly separable, could not be solved by that network. Therefore more levels (also called *layers*) of neurons connected in cascade can be considered.

The multi-layer networks are composed of the following three layers:

- **Input layer**: Composed of n nodes, without any processing capacity, which send the inputs to subsequent layers.
- **Hidden layer** (one or more): Composed of neural elements whose calculations are input to subsequent neural units.
- **Output layer**: Composed of m nodes, whose calculations are the actual outputs of the neural network.

In case of a competitive learning, the output is selected on the basis of a computational principle that takes the name of *Winner-Takes-All*, in other words only the neuron with the greatest "activity" will remain active, while the other neurons will be inactive.

Graphically, we can view the input data on a plane and each layer draws a straight line inside. The intersections between the various lines, generate decision regions. This is a limit to be considered, because the inclusion of too many layers can create too many regions, which means that the perceptron loses the ability to generalize, but it specializes on the set of training data samples. This phenomenon is called **overfitting**. Dually, the inverse problem exists as well, where the network has a number of neurons unable to learn and it is called **underfitting**.

Avoiding similar "excesses" can be achieved by preventive mechanisms such as cross-validation and the early stopping. Contrary to the single perceptron, assuming some hypotheses about activation functions of the individual element, it is possible to approximate any continuous function on a compact set and then to solve the problems of classification of not linearly separable sets (Kolmogorov theorem (1957)). According to that theorem, with three layers it was possible to

implement any continuous function in $[0,1]^n$, where in the first layer we place the n input elements, in the intermediate layer $(2n+1)$ elements, and in last layer we place the m elements (equal to the number of elements of the co-domain space $\mathbb{R}^m$).

3.1 Training a Neural Network

The modelling of a multilayer neural network leads to two main problems, namely, we have to:

- **Select the architecture**: the number of layers and neurons that each layer should possess, and
- **Train the network**: determine the appropriate weights of each neuron and its threshold.

Typically, if we fix an architecture, the training problem can be seen as the ability of the system to produce the outputs as similar to the desired ones, which is equivalent to minimizing the error –usually the most used is the squared error:

$$E_i(w) = \frac{1}{2}||D_i - O_i||^2 \tag{1}$$

where we indicate for simplicity as D_i the desired output of the generic i-th neuron (in place of y_i), and with O_i the obtained output (in place of $y(x_i, w)$, depending on the weight and input).

It is commonplace to usually follow heuristic methods such as structural stabilization, regularization as well as search methods Simulated annealing and Genetic algorithms (see e.g. [8,19]), because the statistical theories are often not adequate.

The **structural stabilization** heuristic consists in gradually growing, during the training, the number of neural elements (whose set is called *training set*). Initially, the error of this network is estimated on the training set and on a different set, called *validation set*. Then, we can select the network that produces the minimum error on the latter. Once trained, the network will be evaluated using a third set called *test set*.

The **regularization** heuristic, on the other hand, consists of adding penalty to the error, with the effect of restricting the choice set of weights w (see Eq. 1).

4 KDD99 Dataset and Features Description

We aim to train a neural network to enable it to predict and distinguish between malicious connections (see next a list of various kinds of attacks from four main categories) and not malicious (normal connections). To train our network we will use a publicly available dataset, which is labelled for IDS, namely the KDD99, a subset of DARPA dataset[2]. This dataset was created by acquiring nine weeks

[2] https://www.ll.mit.edu/ideval/data/.

of raw TCP dump data from a LAN, simulating a typical U.S. Air Force LAN, that is, attacks on a military environment. The connections are a sequence of TCP packets and each record consists of about 100 bytes.

The attacks fall into four main categories:

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

Being the dataset very large, about 500.000 records, we used only a tenth part for the training of MPL network. The features originally are not organized in a tabular file but, through pre-processing, we have changed the format in ARFF, which is useful to process it in KNIME environment with the components of Weka package [12]. According to the category, we can observe the following attacks:

Category	Attack
DOS	Back, Land, Neptune, Pod, Smurf, Teardrop
U2R	Ipsweep, Nmap, Portsweep, Satan
R2L	Bueroverow, Perl, Loadmodule, Rootkit
Probe	Ftpwrite, Imap, GuessPasswd, Phf, Multihop Warezmaster, Warezclient

Each sample have the following features, classified into three main categories:

Basic features of individual TCP connections:

1. *Duration*: length (number of seconds) of the connection.
2. *Procotol_type*: type of the protocol, e.g. tcp, udp, etc.
3. *Service*: network service at the destination, e.g., http, telnet, etc.
4. *Src_bytes*: number of data bytes from source to destination.
5. *Dst_bytes*: number of data bytes from destination to source.
6. *Flag*: normal or error status of the connection.
7. *Land*: 1, if connection is from/to the same host/port; 0, otherwise.
8. *Wrong_fragment*: number of "wrong" fragments.
9. *Urgent*: number of urgent packets.

Content features within a connection suggested by domain knowledge:

10. *Hot*: number of ",hot" indicators.
11. *Num_failed_logins*: number of failed login attempts.
12. *Logged_in*: 1 if successfully logged in; 0, otherwise.
13. *Num_compromised*: number of "compromised" conditions.
14. *Root_shell*: 1, if root shell is obtained; 0, otherwise.
15. *Su_attempted*: 1, if "su root" command attempted; 0, otherwise.
16. *Num_root*: number of "root" accesses.
17. *Num_file_creations*: number of file creation operations.

18. *Num_shells*: number of shell prompts.
19. *Num_access_files*: number of operations on access control files.
20. *Num_outbound_cmds*: number of outbound commands in an ftp session.
21. *Is_hot_login*: 1, if the login belongs to the "hot" list; 0, otherwise.
22. *Is_guest_login*: 1, if the login is a "guest" login; 0, otherwise.

Traffic features computed using a two-second time window:

23. *Count*: number of connections to the same host as the current connection in the past two seconds.
24. *Serror_rate*: % of connections that have "SYN" errors.
25. *Rerror_rate*: % of connections that have "REJ" errors.
26. *Same_srv_rate*: % of connections to the same service.
27. *Diff_srv_rate*: % of connections to different services.
28. *Srv_count*: number of connections to the same service as the current connection in the past two seconds.

    The following features refer to these same-service connections.
29. *Srv_serror_rate*: % of connections that have "SYN" errors.
30. *Srv_rerror_rate*: % of connections that have "REJ" errors.
31. *Srv_diff_host_rate*: % of connections to different hosts.

In addition to traffic features computed using a two-second time window, we added nine other similar attributes but relative to the destination and marked them with "dst". There is also a "class" attribute that identifies the type of attack, which is our target attribute.

In the following subsections, we detail the various components used in our system. For the data analysis, we used KNIME software, which enables data pre-processing (extractions, transformation and loading, modelling [20–22], analysing, and displaying data.)

4.1 Preprocessing and Features Selection

Given the huge number of features, we are compelled to make a selection of the essential attributes. Also, the attributes have different types: continuous, discrete and symbolic, each with its own resolution and range of variation. We can convert symbolic attributes into numerical (attributes like *protocol_type*, *service*, *flag*) and normalize the other attributes between 0 and 1.

We can observe the following blocks by using KNIME model [9]:

- **ARFF Reader**: It reads the file containing the samples.
- **Partitioning**: It partitions the table considering only 10% of the dataset.
- **Category to Number**: It takes symbolic attributes and converts them to numeric values.
- **Row Filter**: It filters rows of non-malicious connections by marking them with a 0 and rows of malicious connections by marking them with 1.
- **Concatenate**: It combines the changed tables.
- **Normalizer**: It normalizes between 0 and 1 the attribute values, dividing the value of each attribute by its maximum.

- **Color Manager**: It assigns a color to the Normal class (0) and one to Attack class (1).
- **AttributeSelectedClassifier (v3.7)**: It carries out the selection of the most discriminating attributes, based on various algorithms that we show next.

To select the discriminating attributes, we tested different configurations for the *AttributeSelectedClassifier* block, also making use of an external tool, Weka Explorer, to better display the outcomes. No substantial differences were noted: both search algorithms on 31 attributes will select 11: *protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, root_shell, count, diff_srv_rate, dst_host_same_src_port_rate*.

We have considered the two following search algorithms:

- **GreadyStepwise**: This basically uses a Hill Climbing (HC) algorithm to return a number of essential features equal to 11. The choice of HC algorithm is preferred due to its lower computational time (about 28 s in our experiments).
- **BestFirst**: This is similar to GreadyStepwise but with the use of backtracking. It returns the same number of features (11) but with a higher computational time (32 s), reason for which it was discarded.

We applied, in addition, the ranker in order to obtain a consistency on features choices. From a list of attributes classified by an evaluator, it sorts them in descending vote and still get a consistent choice. All these blocks, for practical reasons, are encapsulated in a single meta-node, called *Pre-processing*.

4.2 Training of the Multilevel Perceptron

In this phase, we present examples to the network and then we calculate the resulting error, with respect to the desired outputs. Based on this value, it is decided whether to train the network again or not.

An epoch is defined as the amount of time elapsing between the presentation of the first sample of the training set and the last one. Therefore, the termination condition may be a predetermined number of reasonably epochs coming to a point where the error is less than an established value. In KNIME we used WekaMultiLayerPerceptron and WekaPredictor to achieve the MLP network. It should be noted here that the neurons are using a sigmoidal activation function.

The block diagram of the subsystem used for the test is shown in Fig. 1. We can observe the presence of the "ARFF Reader" and the preprocessing block before treated. We have also added the following blocks:

- **Column Filter**: filters the columns by selecting only 11 relevant attributes.
- **Shuffle**: mixes samples of the data set.
- **Partitioning**: divides the sample into two groups of features. We call the first one TS (Training Set) and it is formed from 60% of the samples and the other is called TTS (Test Set) and is formed from 40% (the usual percentages).
- **MultiLayerPerceptron**: trains the MLP based on TS.

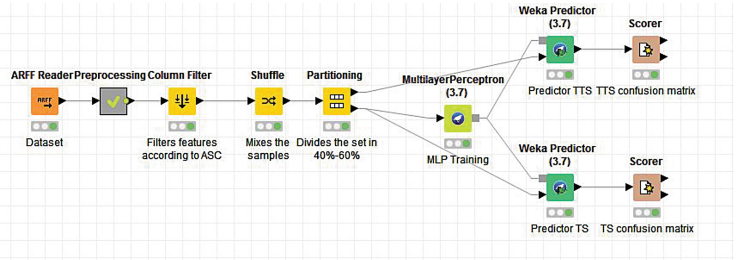

Fig. 1. KNIME training scheme.

- **WekaPredictor**: classifies data according to training carried out on the previous module. In the project once it is used with TS input and the other with TTS input.
- **Scorer**: compares the attributes of two columns and shows the confusion matrix. The latter gives us information on the erroneous classification. In addition, it also shows the accuracy statistics (true/false negative/positive, etc.).

5 Computational Results

Each subset was tested with two different configurations of the MLP:

- **a**: Number of hidden layers equal to the sum of features and classes divided by two.
- **t**: Number of hidden layers equal to the sum of features and classes.

In Figs. 2, 3 and 4, we show graphical representations of the results in terms of error rate with increasing training periods, in order to choose the appropriate parameters. It is observed that each time the system is trained again.

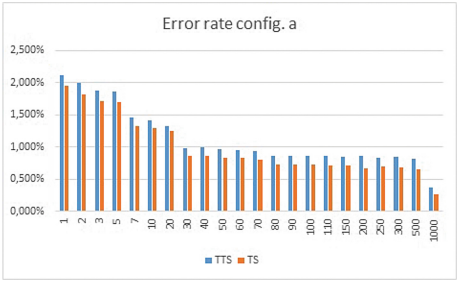

Fig. 2. Error rate bar graph corresponding to configuration a.

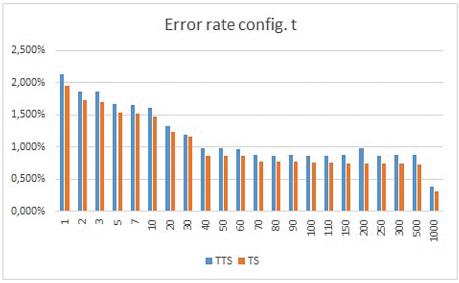

Fig. 3. Error rate bar graph corresponding to configuration t.

Based on the results, we have chosen to configure the WekaMultiLayerPerceptron with the following parameters:

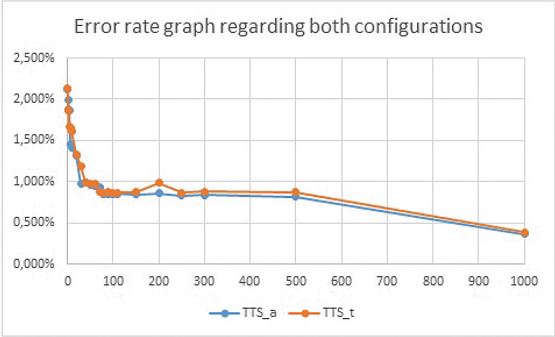

Fig. 4. Error rate graph corresponding to a and t configurations.

- **Learning Rate** and **Momentum**: 0.5
- **Training Time**: 80
- **HiddenLayers**: a

We observe that it is needed a large number of iterations to achieve an acceptable error and that we have chosen two usual values for learning rate and momentum.

6 Conclusion

In this paper we have addressed attack classification in intrusion detection systems (IDS) by multilayer perceptron (MLP) with the back-propagation algorithm and the sigmoidal activation function. The proposed attack classification system is validated and its performance studied through a subset of the DARPA dataset, known as KDD99, which is a public dataset labelled for an IDS and previously processed. We have observed that MLP neural networks are well suited to the classification and achieved a high classification rate with them, making error rate low. This was possible thanks to the analysis of various configurations. MPL networks require time and a good knowledge domain to be trained, however after that they are able to quickly classify both existing and new attacks. Another advantage of the proposed model is related to scalability, namely, we do not need to train again the entire network when we add a new type of attack, but only the set of layers that have the new attack as input.

Future work will be focused on the possibility of removing the classification errors, trying new types of attacks and changing other parameters such as learning rate and momentum as well as comparing the results with other machine learning models. In addition we will try some other Machine Learning framework on the Cloud like Google Cloud Machine Learning and Amazon Machine Learning in order to improve results and performances.

References

1. Alfantookh, A.A.: DoS attacks intelligent detection using neural networks. J. King Saud Univ. Comput. Inf. Sci. **18**, 27–45 (2006)

2. Amato, F., Moscato, F.: Pattern-based orchestration and automatic verification of composite cloud services. Comput. Electr. Eng. **56**, 842–853 (2016)
3. Amato, F., Moscato, F.: Amato, a model driven approach to data privacy verification in E-Health systems. Trans. Data Priv. **8**(3), 273–296 (2015)
4. Aydin, M.A., Zaim, A.H., Ceylan, K.G.: A hybrid intrusion detection system design for computer network security. Comput. Electr. Eng. **35**(3), 517–526 (2009)
5. Barapatre, P., Tarapore, N.: Training MLP neural network to reduce false alerts in IDS. In: Proceedings of the 2008 International Conference on Computing, Communication and Networking (ICCCN 2008), USA. https://doi.org/10.1109/ICCCNET.2008.4787714
6. Cilardo, A.: Efficient bit-parallel GF (2^M) multiplier for a large class of irreducible pentanomials. IEEE Trans. Comput. **58**(7), 1001–1008 (2009)
7. Heba, E.I., Sherif, M.B., Mohamed, A.S.: Adaptive layered approach using machine learning techniques with gain ratio for intrusion detection systems. Int. J. Comput. Appl. **56**(7) (2012)
8. Kajornrit, J.: A comparative study of optimization methods for improving artificial neural network performance. In: 7th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 35–40. IEEE CPS (2015)
9. KNIME.org: Software documentation. https://www.knime.org/
10. Laheeb, M.I., Dujan, T.B.: A comparison study for intrusion database. J. Eng. Sci. Technol. **8**(1), 107–119 (2013)
11. Jajodia, S., Park, N., Serra, E., Subrahmanian, V.S.: Using temporal probabilistic logic for optimal monitoring of security events with limited resources. J. Comput. Secur. **24**(6), 735–791 (2016)
12. Frank, E., Hall, M.A., Witten, I.H.: The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", 4th edn. Morgan Kaufmann (2016)
13. Long, J., Schwartz, D., Stoecklin, S.: Distinguishing false from true alerts in snort by data mining patterns of alerts. In: Proceedings of SPIE Defense and Security Symposium, pp. 62410B-1–10 (2006)
14. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. The MIT Press, Cambridge (1969)
15. Przemysław, K., Zbigniew, K.: Analysis of neural networks usage for detection of a new attack in IDS. Ann. UMCS Inf. **10**(1), 51–59 (2010)
16. Risto, V., Podins, K.: Network IDS alert classification with frequent itemset mining and data clustering. In: International Conference on Network and Service Management (CNSM), pp. 451–456. IEEE (2010)
17. Rodas, O., To, M.A.: A study on network security monitoring for the hybrid classification-based intrusion prevention systems. Int. J. Space-Based Situated Comput. **5**(2), 115–125 (2015)
18. Vaarandi, R.: Real-time classification of IDS alerts with data mining techniques. In: Proceedings of Military Communications Conference (MILCOM 2009), 7 pp. IEEE (2009). https://doi.org/10.1109/MILCOM.2009.5379762
19. Yasuoka, Y., Shinomiya, Y., Hoshino, Y.: Evaluation of optimization methods for neural network. In: Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), pp. 92–96. IEEE CPS (2016)
20. Xhafa, F., Barolli, L.: Semantics, intelligent processing and services for big data. Futur. Gener. Comput. Syst. **37**, 201–202 (2014)

21. Moore, P., Xhafa, F., Barolli, L.: Semantic valence modeling: emotion recognition and affective states in context-aware systems. In: Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014, pp. 536–541 (2014)
22. Javanmardi, S., Shojafar, M., Shariatmadari, S., Ahrabi, S.: Fr trust: a fuzzy reputation- based model for trust management in semantic p2p grids. Int. J. Grid Util. Comput. **6**(1), 57–66 (2015)
23. Yu, Q., Gu, X.: Network traffic anomaly detection based on dynamic programming. In: International Conference on Computing Intelligence and Information System (CIIS), pp. 62–65. IEEE CPS (2017)
24. Zhang, L., Chen, Y., Liao, S.: Algorithm optimization of anomaly detection based on data mining. In: 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), pp. 402–404. IEEE CPS (2018)