Machine Learning Based Network Intrusion Detection

Chie-Hong Lee, Yann-Yean Su
Department of Digital Content, Application and Management, Wenzao Ursuline University, of Languages, Kaohsiung, Taiwan
e-mail:
chichhong@gmail.com; leesj@mail.ee.nsysu.edu.tw

Yu-Chun Lin and Shie-Jue Lee
Department of Electrical Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan

***Abstract*—Network security has become a very important issue and attracted a lot of study and practice. To detect or prevent network attacks, a network intrusion detection (NID) system may be equipped with machine learning algorithms to achieve better accuracy and faster detection speed. One of the major advantages of applying machine learning to network intrusion detection is that we don't need expert knowledge as much as the black or white list model. In this paper, we apply the equality constrained-optimization-based extreme learning machine to network intrusion detection. An adaptively incremental learning strategy is proposed to derive the optimal number of hidden neurons. The optimization criteria and a way of adaptively increasing hidden neurons with binary search are developed. The proposed approach is applied to network intrusion detection to examine its capability. Experimental results show our proposed approach is effective in building models with good attack detection rates and fast learning speed.**

keywords-network intrusion detection; machine learning; extreme learning machine; incremental learning.

I. Introduction

In response to rising popularity and fast development of the Internet, more and more networks have been established for businesses, social media and governments. Because of the diverse and complicated expertise required for cyber-security, technical staff may not be able to manage their networks properly. As a result, the networks are prone to be attacked and the concept of network intrusion detection (NID) has been proposed. NID systems are mainly used to protect networks from being attacked and filter or detect malicious behavior launched by attackers, e.g., denial of services attacks (DoS) [1].

One of the most common detection approaches is signature-based detection using data mining algorithms, e.g., white list or black list models. Meng *et al.* proposed a way to improve signature-based detection by reducing detection load, speeding up the signature matching process, and reducing the false alarm rate [2]. Jamdagni *et al.* proposed a detection system based on packet payloads [3]. Text features are extracted from payloads by the n-gram method and used to build nomal behavior models. Both the above apply analysis and statistic methods to generate signatures for their systems. However, the signature-based models are created from the comprehensive and detailed data collected through protected networks. When unknown data are sent to such a signature-based detection system, they are more likely to be labeled as incorrect categories. As a result, if the system lacks a proper discriminating mechanism, it could cause lots of false positives or false negatives. For machine learning-based systems, their models can learn from training data and adjust the output classification results. The learning is not only based on one single type of data, as white or black list models, but also based on all kinds of data. Through computation and comparison, incoming data to a machine learning-based system are classified as the most possible category. Many machine learning techniques, e.g., genetic and fuzzy algorithms [4], clustering and K-nearest neighbor (k-NN) algorithms [5], etc., have been used to build NID systems.

For NID systems, the sampling rate of network data is very high, and an appropriate detection model must be trained with a large amount of data which may fail an iterative training model as BP neural networks. Fast learning speed of extreme learning machines (ELMs) makes them suitable for use in NID systems. Several modified versions of ELM have been developed to improve its capability. For classification problems, Huang *et al.* have proposed the inequality constrained-optimization-based ELM [6]. Huang *et al.* also proposed another ELM, called the equality constrained-optimization-based ELM (C-ELM) [7], which integrates with the learning rule of least squares SVM (LS-SVM) [8], a well-known variant of SVM [9]. C-ELM adopts a similar objective function as LS-SVM to obtain a global solution for the weights of output layer.

The number of hidden neurons in ELMs needs to be determined in advance by the user. If an ELM doesn't achieve the expected performance, different numbers of hidden neurons have to be tested until satisfied [10]. However, every time a different number of hidden neurons is tested, the output weights need to be re-computed from scratch. In this paper, we propose an efficient C-ELM construction approach by which hidden neurons are added in an adaptively incremental way and the output weights are derived without re-computation from scratch. The optimization criteria and a way of adaptively increasing hidden neurons with binary search are developed.

978-1-5386-2030-4/17/$31.00 ©2017 IEEE

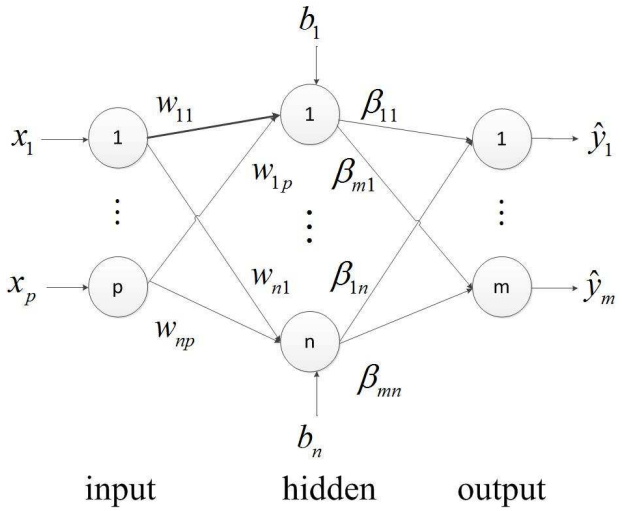

Fig. 1. C-ELM with p input, n hidden, and m output neurons.

As a result, time and efforts can be saved during the construction process of an optimal C-ELM. Our proposed approach is applied to network intrusion detection to examine its capability. Experimental results show that the proposed approach is effective in building models with good attack detection rates and fast learning speed.

The rest of this paper is organized as follows. C-ELM is briefly introduced in Section II. Our proposed C-ELM construction approach is described in Section III. Section IV presents experimental results. Finally, concluding remarks are given in Section V.

II. Equality Constrained-Optimization-Based ELM (C-ELM)

C-ELM is a variant of ELM neural networks. Like ELM, it consists of three layers: input, hidden, and output layers, as shown in Fig. 1 in which p is the number of input nodes, n is the number of hidden nodes, and m is the number of output nodes. The input and hidden layers are fully connected with input weights, $\mathbf{w}_j = \begin{bmatrix} w_{j1} & w_{j2} & \dots & w_{jp} \end{bmatrix}^{\mathrm{T}}$, $j = 1, \dots, n$, while the hidden and output layers are fully connected with output weights, $\boldsymbol{\beta}_k = \begin{bmatrix} \beta_{k1} & \beta_{k2} & \dots & \beta_{kn} \end{bmatrix}^{\mathrm{T}}$, $k = 1, \dots, m$. For any input vector $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_p \end{bmatrix}^{\mathrm{T}}$, the predicted vector $\hat{\mathbf{y}}$ produced by Fig. 1 at the output nodes is

$$\begin{aligned} \hat{\mathbf{y}} &= \begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \dots & \hat{y}_m \end{bmatrix}^{\mathrm{T}}, \qquad (1) \\ \hat{y}_k &= \sum_{j=1}^{n} \beta_{kj} g(\mathbf{w}_j^{\mathrm{T}} \mathbf{x} + b_j) \end{aligned}$$

where $g(\cdot)$ is the kernel function of the hidden nodes and b_j is the bias of the jth hidden neuron.

Like ELM, the input weights $\mathbf{w}_j$ and biases b_j, $j = 1, \dots, n$, of a C-ELM are assigned to random values and no training is done for them. However, output weights $\boldsymbol{\beta}_k$, $k = 1, \dots, m$, are learned from a given set of training instances. Let $\mathbf{S}$ be the given set of N training instances $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \dots, N$, where $\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix}^{\mathrm{T}} \in \mathbf{R}^p$ and $\mathbf{y}_i = \begin{bmatrix} y_{i1} & y_{i2} & \dots & y_{im} \end{bmatrix}^{\mathrm{T}} \in \mathbf{R}^m$ are the input vector and the target output vector, respectively, of training instance i. The output weights $\boldsymbol{\beta}_k$, $k = 1, \dots, m$, are learned from the training instances by solving the following objective function

$$\begin{aligned} \text{Minimize:} \quad & L(\boldsymbol{\Omega}, \boldsymbol{\xi}) = \tfrac{1}{2} ||\boldsymbol{\Omega}||^2 + \tfrac{C}{2} \textstyle\sum_{i=1}^{N} ||\boldsymbol{\xi}_i||^2 \\ \text{subject to:} \quad & \boldsymbol{\Omega}^{\mathrm{T}} h(\mathbf{x}_i) = \mathbf{y}_i - \boldsymbol{\xi}_i, \quad i = 1, \dots, N \qquad (2) \end{aligned}$$

where C is a regularization parameter, $\boldsymbol{\xi}_i$ is the predicted error of instance i, and

$$\begin{aligned} \boldsymbol{\Omega} &= \begin{bmatrix} \boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 & \dots & \boldsymbol{\beta}_m \end{bmatrix}, \qquad (3) \\ h(\mathbf{x}_i) &= \begin{bmatrix} g(\mathbf{w}_1^{\mathrm{T}} \mathbf{x}_i + b_1) \\ g(\mathbf{w}_2^{\mathrm{T}} \mathbf{x}_i + b_2) \\ \vdots \\ g(\mathbf{w}_n^{\mathrm{T}} \mathbf{x}_i + b_n) \end{bmatrix} \qquad (4) \end{aligned}$$

are matrices of size $n \times m$ and $n \times 1$, respectively.

By following the traditional procedure of applying Lagrangian operators and Karush-Kuhn-Tucker (KKT) conditions, a solution to Eq.(2) is obtained and

$$\boldsymbol{\Omega} = \left(\frac{\mathbf{I}_{n \times n}}{C} + \mathbf{H}^{\mathrm{T}} \mathbf{H} \right)^{-1} \mathbf{H}^{\mathrm{T}} \mathbf{Y}. \qquad (5)$$

where $\mathbf{I}_{n \times n}$ stands for the identity matrix of size $n \times n$ and

$$\mathbf{H} = \begin{bmatrix} h(\mathbf{x}_1)^{\mathrm{T}} \\ h(\mathbf{x}_2)^{\mathrm{T}} \\ \vdots \\ h(\mathbf{x}_N)^{\mathrm{T}} \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{\mathrm{T}} \\ \mathbf{y}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{y}_N^{\mathrm{T}} \end{bmatrix}. \qquad (6)$$

C-ELM has been shown to achieve similar or even better generalized performance than SVM and LS-SVM [7].

III. Proposed Construction Approach

One issue with C-ELM is that it is time consuming in finding an appropriate number of hidden nodes by trial-and-error in the construction process. When the number of hidden nodes in a C-ELM is increased, the output weights associated with the resulting C-ELM have to be re-computed from scratch. Assume we initially create a C-ELM with n hidden neurons for an application. The output weights for this n-hidden-neurons C-ELM is computed by Eq.(5) which involves the inversion of a matrix of size $n \times n$. Suppose we add Δn more hidden neurons to this machine, resulting in $n + \Delta n$ neurons in the hidden layer. Then the output weights for this $(n + \Delta n)$-hidden-neurons C-ELM is again computed from scratch by Eq.(5) which this time involves the inversion of a matrix of size $(n + \Delta n) \times (n + \Delta n)$. A number of such trials may be required before a satisfactory number of hidden neurons for the C-ELM is found. This causes a waste of time and efforts if a C-ELM with the optimal number of hidden neurons is pursued. Although IC-ELM [11] can incrementally add hidden nodes to its model, it still need trials for fixed size of Δn. We develop a construction approach by which hidden nodes can be added and the output weights can be

updated in an adaptively incremental way for C-ELM. Besides, re-computing the output weights from scratch is avoided and time can be saved.

A. Incremental Learning

Assume initially we create a C-ELM with n_0 hidden neurons for an application. The values of the input weights and biases are randomly generated. We compute the $\mathbf{H}$ matrix of this n_0-hidden-neurons C-ELM by Eq.(6) and denote it as $\mathbf{H}_0$ which is of size $N \times n_0$. The output weights $\mathbf{\Omega}_0$ associated with this n_0-hidden-neurons C-ELM, according to Eq.(5), is

$$\mathbf{\Omega}_0 = \left(\frac{\mathbf{I}_{n_0\times n_0}}{C} + \mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0\right)^{-1}\mathbf{H}_0^{\mathrm{T}}\mathbf{Y}. \quad (7)$$

Let

$$\mathbf{H}_0^{\dagger} = \left(\frac{\mathbf{I}_{n_0\times n_0}}{C} + \mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0\right)^{-1}\mathbf{H}_0^{\mathrm{T}}. \quad (8)$$

We have

$$\mathbf{\Omega}_0 = \mathbf{H}_0^{\dagger}\mathbf{Y}. \quad (9)$$

Now suppose we add Δn_0 more hidden neurons to the C-ELM. The resulting machine then has n_1 neurons in the hidden layer, where

$$n_1 = n_0 + \Delta n_0. \quad (10)$$

Let the $\mathbf{H}$ matrix of this n_1-hidden-neurons C-ELM be $\mathbf{H}_1$ which is of size $N \times n_1$, and can be written as

$$\mathbf{H}_1 = \begin{bmatrix}\mathbf{H}_0 & \Delta\mathbf{H}_0\end{bmatrix} \quad (11)$$

where $\Delta\mathbf{H}_0$, of size $N\times\Delta n_0$, is computed from the randomly generated values of the input weights and biases associated with the added Δn_0 hidden neurons. From Eq.(5), we have the output weights of this n_1-hidden-neurons C-ELM to be

$$\mathbf{\Omega}_1 = \mathbf{H}_1^{\dagger}\mathbf{Y} \quad (12)$$

where

$$\begin{aligned}\mathbf{H}_1^{\dagger} &= \left(\frac{\mathbf{I}_{n_1\times n_1}}{C} + \mathbf{H}_1^{\mathrm{T}}\mathbf{H}_1\right)^{-1}\mathbf{H}_1^{\mathrm{T}} \\ &= \left(\frac{\mathbf{I}_{n_1\times n_1}}{C} + \begin{bmatrix}\mathbf{H}_0^{\mathrm{T}}\\ \Delta\mathbf{H}_0^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\mathbf{H}_0 & \Delta\mathbf{H}_0\end{bmatrix}\right)^{-1}\begin{bmatrix}\mathbf{H}_0^{\mathrm{T}}\\ \Delta\mathbf{H}_0^{\mathrm{T}}\end{bmatrix}.\end{aligned} \quad (13)$$

Note that

$$\mathbf{I}_{n_1\times n_1} = \begin{bmatrix}\mathbf{I}_{n_0\times n_0} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{\Delta n_0\times\Delta n_0}\end{bmatrix}. \quad (14)$$

Note that Eq.(12) requires the computation of the inverse of a $n_1\times n_1$ matrix. To reduce the computation complexity, we can rewrite Eq.(13) as

$$\begin{aligned}&\mathbf{H}_1^{\dagger}\\ =\ &\begin{bmatrix}\frac{\mathbf{I}_{n_0\times n_0}}{C} + \mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0 & \mathbf{H}_0^{\mathrm{T}}\Delta\mathbf{H}_0\\ \Delta\mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0 & \frac{\mathbf{I}_{\Delta n_0\times\Delta n_0}}{C} + \Delta\mathbf{H}_0^{T}\Delta\mathbf{H}_0\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{H}_0^{\mathrm{T}}\\ \Delta\mathbf{H}_0^{\mathrm{T}}\end{bmatrix}\\ =\ &\begin{bmatrix}\mathbf{U}_1\\ \mathbf{D}_1\end{bmatrix}.\end{aligned} \quad (15)$$

Based on the inversion of block matrices, we have

$$\begin{aligned}\mathbf{D}_1 =\ &[\frac{\mathbf{I}_{\Delta n_0\times\Delta n_0}}{C} + \Delta\mathbf{H}_0^{\mathrm{T}}(\mathbf{I}_{N\times N} - \mathbf{H}_0\mathbf{H}_0^{\dagger})\Delta\mathbf{H}_0]^{-1}\times\\ &\Delta\mathbf{H}_0^{\mathrm{T}}(\mathbf{I}_{N\times N} - \mathbf{H}_0\mathbf{H}_0^{\dagger}).\end{aligned} \quad (16)$$

Similarly, we have

$$\mathbf{U}_1 = \mathbf{H}_0^{\dagger}(\mathbf{I}_{N\times N} - \Delta\mathbf{H}_0\mathbf{D}_1). \quad (17)$$

Note that it is only a $\Delta n_0\times\Delta n_0$ matrix involved in computing $\mathbf{U}_1$ and $\mathbf{D}_1$. Therefore,

$$\mathbf{\Omega}_1 = \begin{bmatrix}\mathbf{U}_1\\ \mathbf{D}_1\end{bmatrix}\mathbf{Y} \quad (18)$$

which can be computed much more efficiently.

B. Constructing C-ELM

Now we proceed with constructing a C-ELM with an optimal number of hidden neurons for any given application. Initially, we create a C-ELM having n_0 hidden neurons, a good choice of n_0 being equal to the number of output nodes, m. $\mathbf{H}_0^{\dagger}$ is computed, from which the output weights $\mathbf{\Omega}_0$ is computed by Eq.(9). If the C-ELM matches our pre-specified goal, e.g., training error bound, we are done. Otherwise, a number of Δn_0 hidden neurons are added and we set $n_1 = n_0 + \Delta n_0$. The output weights of the n_1-hidden-neurons C-ELM are computed by Eq.(18). Then we check if the resulting n_1-hidden-neurons C-ELM matches our goal. If so, we are done. Otherwise, another Δn_1 hidden neurons are added and we set $n_2 = n_1 + \Delta n_1$. The output weights of the n_2-hidden-neurons C-ELM are computed by Eq.(18). Then we check if the resulting n_2-hidden-neurons C-ELM matches our goal. If so, we are done. Otherwise, another Δn_2 hidden neurons are added and the output weights of the n_3-hidden-neurons C-ELM are computed by Eq.(18). This process iterates until the C-ELM matches our goal, as summarized below.

procedure Constructing C-ELM incrementally
  Initialize the C-ELM with n_0 hidden neurons;
  Compute $\mathbf{\Omega}_0$ by Eq.(9);
  Set $k = 0$;
  while the pre-specified goal is not met
    Set $k = k + 1$;
    Set $n_k = n_{k-1} + \Delta n_{k-1}$;
    Compute $\mathbf{\Omega}_k$ by Eq.(18);
  end while;
  Set $K = k$;
end procedure

The increments Δn_k, $k = 0, 1, \ldots$, are computed adaptively. At the time the procedure stops, K times of additions have been done, and the C-ELM obtained has n_K hidden neurons with the output weights $\mathbf{\Omega}_K$.

Let's give an example showing the construction of a C-ELM. In this example, there are 5 output nodes. Initially, 5 hidden nodes are set for the C-ELM. Therefore, $n_0 = 5$. We would like the accuracy of the C-ELM to be stable. We calculate the first addition, Δn_0, which is 1. So we have

TABLE I
A CONSTRUCTION EXAMPLE

k	Accuracy(%)	Δn_k	n_k
0	94.3918	1	5
1	91.4869	1	6
2	97.0425	1	7
3	97.5747	1	8
⋮	⋮	⋮	⋮
7	97.7875	9	14
8	98.3674	9	23
⋮	⋮	⋮	⋮
39	99.9055	8	413
40	99.9028	8	421
41	99.9042	-	429

TABLE II
THE NSL-KDD DATASET

Categories	# of instances
Normal	77,054
DoS	53,385
Probe	14,077
U2R	252
R2L	3,749
Total	148,517

$n_1 = 5 + 1 = 6$. The goal is not met. We then calculate Δn_1, which is also 1. So we have $n_2 = 6 + 1 = 7$. The goal is not met either. We then calculate Δn_2, and so on. The whole process is shown in TABLE I. At $k = 41$, the accuracy becomes stable, and the process stops. Therefore, the C-ELM has 429 neurons in the hidden layer.

IV. APPLICATION IN NETWORK INTRUSION DETECTION

In this section, we apply our proposed approach to build C-ELMs as classifiers for network intrusion detection. Experiments are conducted on benchmark NID datasets and comparisons with other methods are done to demonstrate the effectiveness of our approach.

A commonly used benchmark dataset, the NSL-KDD dataset [12], which is a less biased subset from the KDD-Cup 99 dataset [13], as shown in Table II, is adopted in our experiments. This dataset contains only 148,517 instances, with 41 features and 5 categories. Three criteria, ACC, REC, and FAR are used for performance evaluation. For an experiment with N_p actual positive instances and N_n actual negative instances, the four outcomes of an classifier can be formulated in a confusion matrix as shown in Table III. The evaluation criteria

TABLE III
CONFUSION MATRIX

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

TABLE IV
COMPARISON WITH SINGH'S IN BINARY CLASSIFICATION

	Ours	Singh's [14]		
		All	FS	FS+DR
ACC	**99.02**	98.67	98.67	98.66
REC	98.68	98.94	98.99	**99.02**
FAR	**0.79**	1.62	1.68	1.74
# of hidden neurons	**86**	325	325	325

TABLE V
COMPARISON WITH SINGH'S IN MULTI-CLASS CLASSIFICATION

		Ours	Singh's [14]		
			All	FS	FS+DR
Overall ACC		**98.82**	97.71	97.67	97.67
Normal	REC	98.86	99.06	99.04	**99.07**
	FAR	**0.92**	1.81	1.70	1.74
DoS	REC	**99.56**	99.18	99.13	99.14
	FAR	**0.22**	1.38	1.47	1.49
Probe	REC	**98.94**	91.04	90.48	90.35
	FAR	**0.23**	0.39	0.42	0.40
U2R	REC	**87.26**	55.95	54.36	56.75
	FAR	0.22	0.10	**0.02**	0.10
R2L	REC	**92.49**	77.01	78.69	78.10
	FAR	0.16	**0.15**	0.17	0.16
# of hidden neurons		**151**	325	325	325

are defined as

$$\begin{aligned} \text{ACC (Accuracy)} &= \frac{\text{TP} + \text{TN}}{N_p + N_n}, \\ \text{FAR (False Alarm Rate)} &= \frac{\text{FP}}{N_n}, \\ \text{REC (Recall)} &= \frac{\text{TP}}{N_p}, \end{aligned}$$

To measure the performance of a classifier on a given dataset, we divide the data into 10 folds, and run the classifier 10 times with the dataset. The average of the results of the 10 runs is then shown as the performance of the classifier on the given dataset.

Comparisons with the method proposed by Singh *et al.* [14] are shown in Table IV and Table V. Singh's method incorporates feature selection (FS) and data reduction (DR), and its performance depends on how FS and DR are used. In Table IV, the results are obtained by treating normal instances as negative and all the abnormal instances as positive. Consequently, only two categories are involved. Three different sets of results are shown for Singh's method, all taken directly from [14]. The column with 'All features' shows the results with all the original features involved. The column with 'FS' shows the results with feature selection. The column with 'FS+DR' shows the results with both feature selection and data reduction. We can observe that our approach is not the best in every case in Table IV, but it has advantages. Our approach has a much lower FAR, 0.79%, than Singh's method.

In Table V, the results are obtained by treating the instances of one category as positive and the instances of the other categories as negative. Consequently, 5 categories are involved. Three different sets of results are also shown for Singh's

method, all taken directly from [14]. ACC is given for the whole testing dataset, and REC and FAR are given for each category. From this table, we can see that our approach performs better than Singh's in almost every evaluation measure. Even for the minor category, like the U2R attack, its instances can be well detected by our approach with REC being 87.26%, much higher than those provided by Singh's. Furthermore, the number of hidden neurons derived from our approach is much lower than that derived from Singh's method.

V. Conclusion

We have presented an efficient C-ELM construction approach by which hidden neurons are added incrementally and the output weights are derived without re-computation from scratch. The optimization criteria and a way of adaptively increasing hidden neurons with binary search are developed. As a result, time and efforts can be saved during the construction process of an optimal C-ELM. Our proposed approach is applied to network intrusion detection to examine its capability. Experimental results have shown that the proposed approach is effective in building models with good attack detection rates and fast learning speed. We have also shown that even the attack instances in the minority are not ignored and are also detectable.

References

[1] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A system for denial-of-service attack detection based on multivariate correlation analysis," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 2, pp. 447–456, 2014.

[2] W. Meng, W. Li, and L.-F. Kwok, "Efm: Enhancing the performance of signature-based network intrusion detection systems using enhanced filter mechanism, Computers & Security, vol. 43, pp. 189–204, 2014.

[3] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu, "Repids: A multi tier real-time payload-based intrusion detection system," Computer Networks, vol. 57, pp. 811–824, 2013.

[4] S. Elhag, A. Fernndez, A. Bawakid, S. Alshomrani, and F. Herrera, "On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on intrusion detection systems," Expert Systems With Applications, vol. 42, no. 22, pp. 193–202, 2015.

[5] W.-C. Lin, S.-W. Ke, and C.-F. Tsai, "CANN: An intrusion detection system based on combining cluster centers and nearest neighbors," Knowledge-Based Systems, vol. 78, pp. 13–21,2015.

[6] G.-B. Huang, X. Ding, and H. Zhou, "Optimization method based extreme learning machine for classification," Neurocomputing, vol. 74, no. 1, pp. 155–163, 2010.

[7] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 42, no. 2, pp. 513–529, 2012.

[8] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," Neural processing letters, vol. 9, no. 3, pp. 293–300, 1999.

[9] C. Cortes and V. Vapnik, "Support-vector networks," Machine learning, vol. 20, no. 3, pp. 273–297, 1995. .

[10] R. Wang, S. Kwong, and D. D. Wang, An analysis of ELM approximate error based on random weight matrix," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 21, no.supp02, pp. 1–12, 2013.

[11] R.-F. Xu, Z.-Y. Wang, and S.-J. Lee, "onstrained-optimization-based extreme learning machine with incremental learning," Proc. International Conference on Multimedia, Communication and Computing Application (MCCA 2014), Xiamen, China, October 16-17, 2014, pp. 315–318.

[12] NSL-KDD data set, http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html

[13] KDD-Cup 99 data set, https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[14] R. Singh, H. Kumar, and R. K. Singla, "An intrusion detection system using network traffic profiling and online sequential extreme learning machine," Expert Systems With Applications, vol. 42, pp. 8609–8624, 2015.