Mathematical Biosciences and Engineering

MBE, 20(4): 6966–6992.
DOI: 10.3934/mbe.2023301
Received: 28 September 2022
Revised: 18 January 2023
Accepted: 28 January 2023
Published: 09 February 2023

http://www.aimspress.com/journal/MBE

Research article

A two-stage intrusion detection method based on light gradient boosting machine and autoencoder

Hao Zhang[1,2], Lina Ge[1,2,3,*], Guifen Zhang[1,2,*], Jingwei Fan[2,4], Denghui Li[1,2] and Chenyang Xu[1,2]

[1] School of Artificial Intelligence, Guangxi Minzu University, Nanning 530006, China
[2] Key Laboratory of Network Communication Engineering, Guangxi Minzu University, Nanning 530006, China
[3] Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis, Nanning 530006, China
[4] College of Electronic Information, Guangxi Minzu University, Nanning 530006, China

* **Correspondence:** Email:66436539@qq.com, 378012725@qq.com.

Abstract: Intrusion detection systems can detect potential attacks and raise alerts on time. However, dimensionality curses and zero-day attacks pose challenges to intrusion detection systems. From a data perspective, the dimensionality curse leads to the low efficiency of intrusion detection systems. From the attack perspective, the increasing number of zero-day attacks overwhelms the intrusion detection system. To address these problems, this paper proposes a novel detection framework based on light gradient boosting machine (LightGBM) and autoencoder. The recursive feature elimination (RFE) method is first used for dimensionality reduction in this framework. Then a focal loss (FL) function is introduced into the LightGBM classifier to boost the learning of difficult samples. Finally, a two-stage prediction step with LightGBM and autoencoder is performed. In the first stage, pre-decision is conducted with LightGBM. In the second stage, a residual is used to make a secondary decision for samples with a normal class. The experiments were performed on the NSL-KDD and UNSWNB15 datasets, and compared with the classical method. It was found that the proposed method is superior to other methods and reduces the time overhead. In addition, the existing advanced methods were also compared in this study, and the results show that the proposed method is above 90% for accuracy, recall, and F1 score on both datasets. It is further concluded that our method is valid when compared with other advanced techniques.

Keywords: cybersecurity; feature selection; focal loss; intrusion detection systems; machine learning

1. Introduction

Cyberattacks occurr frequently, causing serious impacts on people's daily life. In 2017, the WannaCry ransomware event broke out globally, hitting at least 300,000 users and causing 8 billion USD in damage [1]. In 2020, a cyberattack on Venezuela's national grid trunk line caused widespread power outages across the country [2]. In 2021, the United States refined product pipeline operator Colonial Pipeline was forced to shut down its fuel network in the eastern seaboard states due to a ransomware attack [3]. With the frequent occurrence of cyberattacks, existing methods, such as firewalls, data encryption, and authentication cannot meet security requirements [4]. Therefore, intrusion detection systems have gained the attention of researchers.

Intrusion detection systems play an important role in protecting critical information infrastructure [5]. According to detection techniques, they are categorized into signature-based intrusion detection systems (SIDS) and anomaly-based intrusion detection systems (AIDS) [6,7]. SIDS maintains an attack library that saves historical attack records. If the current traffic matches the record in the attack library, the traffic is judged to be attack class. AIDS analyzes historical traffic using statistical methods to learn a logical model. If the current traffic deviates from the normal traffic, the traffic is judged to be attack class. SIDS offers the advantages of fast detection and a low false alarm rate, but it cannot detect unknown attacks [8]. On the contrary, AIDS can detect unknown attacks and has a wide application prospect in the future. Figure 1 shows the block diagram of the intrusion detection system [9]. It consists of the following key components: (1) Information collection: network data, application logs, audit records, and other relevant information are collected from the network or hosts. The collected information will be used for intrusion analysis. (2) Analysis engine: modeling or behavior matching is performed based on the collected network information, which in turn forms the corresponding knowledge base. It will alert the network administrator if an intrusion is found. The intrusion process will also be part of the information collection. (3) Knowledge base: a list of historical behaviors or trained models are stored. The knowledge base can be used to analyze current traffic, but it needs to be updated regularly.

Intrusion detection is considered a classification problem, which has prompted researchers to adopt machine learning techniques to improve the performance of intrusion detection systems. In recent years, machine learning techniques have been applied broadly in intrusion detection, and have shown encouraging results in many studies [10]. Machine learning techniques can be classified as shallow learning and deep learning [11]. Shallow learning methods, such as K-nearest neighbors [12], decision trees [13], support vector machines [14] and random forests [15], are widely used because of their strong explainability. Among deep learning, autoencoders [16], deep belief networks [17] and convolutional neural networks [18] have achieved great success in intrusion detection owing to their ability to extract features. In the future, finding suitable machine learning techniques for improving the performance of intrusion detection systems has become a hot topic for researchers.

Researchers have proposed many approaches to detect intrusions based on machine learning techniques. This paper reviews related work from the perspectives of anomaly analysis and feature analysis. In anomaly analysis, Chouhan et al. [19] developed an autoencoder-based residual learning technique to enhance the classification capability of convolutional neural networks. Andresini et al. [20] combined feature selection techniques and residual learning to improve the performance of intrusion

detection systems. The above residual thresholds need to be set manually, so Aygun et al. [21] developed a method to determine the thresholds adaptively. Yang et al. [22] developed a method that uses a modified conditional variation autoencoder to generate attack samples for balancing the data. Min et al. [23] developed a memory-enhanced autoencoder to improve the generalizability of the model. Autoencoders are also available for nonlinear dimensionality reduction [24,25]. In addition, to improve the performance of intrusion detection systems, some researchers have developed two-stage decision methods. Belouch et al. [26] introduced a two-stage classification model. In the first stage, a RepTree classifier is used to classify the traffic into normal and abnormal. In the second stage, a classifier is used to classify the anomalies detected in the first stage to identify the attack classes. Niyaz et al. [27] proposed an intrusion detection system based on two phases. The first stage uses a sparse autoencoder for feature extraction from the original data. The second stage feeds the processed features into SoftMaxRegression (SM) and self-taught learning (STL) classifiers for learning, respectively. Zhang et al. [28] applied machine-learning techniques to intrusion detection in in-vehicle networks and proposed a two-stage anomaly detection framework.

In feature analysis, Gu et al. [29] used the marginal density ratio method for data enhancement to improve the performance of intrusion detection. Ieracitano et al. [30] used statistical analysis techniques to identify outliers and redundant data, and, thus, remove unnecessary features. Zhang et al. [31] developed a feature fusion technique to improve model classification performance. Tree-based methods are often used for feature selection. Kasongo et al. [32] used extreme gradient augmentation trees for feature selection followed by shallow methods for classification. Megantara and Ahmad [33] developed a hybrid feature analysis method. This method first uses a decision tree to select the important features. After that, local outlier factors are used to exclude outlier and anomalous features. Rashid et al. [34] used univariate techniques for feature analysis, and integrated methods for classification. Bioheuristics have also been used for feature selection, such as the commonly used particle swarm algorithm and genetic algorithm [35–37]. In addition, deep learning methods are often used for nonlinear feature dimensionality reduction. To address the problem that isolated points and noisy data can affect the model performance, Seo et al. [38] used a restricted Boltzmann machine to remove isolated points and noisy data from the dataset. Wuke et al. [39] proposed a combination of multilayer extreme learning machines and autoencoders to reduce the dimensionality of the data. The reduced-dimensional data are then trained by the extreme learning machine. Zhao et al. [40] proposed a method that used deep belief networks and least-squares vector machines. The method first uses a deep belief network for dimensionality reduction, and then uses a particle swarm algorithm to optimize the parameters of the least-squares vector machine.

Although there is a lot of research focused on intrusion detection systems, there are still some issues that need to be addressed. One of the important issues is the dimensionality curse. The high-dimensional data makes it difficult for intrusion detection systems to learn effective data representations, which affects their detection efficiency. Another problem is the increasing number of zero-day attacks [6]. Various attack methods are emerging, leaving network administrators with shorter response times. To address these issues, a two-stage anomaly detection framework based on LightGBM and autoencoder is proposed in this study. The framework can detect novel attacks while improving detection efficiency. LightGBM is an integrated approach that introduces an exclusive feature bundling algorithm and a gradient-based one-sided sampling algorithm. The exclusive feature bundling (EFB) algorithm reduces the number of features that are simultaneously zero, and the gradient-based one-sided sampling (GOSS) method reduces the number of small gradient samples during model training.

As a result, the LightGBM algorithm has less time overhead. The focal loss function can increase the weight of difficult samples, which is beneficial to learn the attack samples that are difficult to classify. The autoencoder learns the implicit representation of the data at the encoding layer, and reconstructs the original data at the decoding layer. Exploiting the reconstruction error, the autoencoder can enhance anomaly detection. Therefore, our main innovation is to introduce the focal loss function into LightGBM instead of the Cross-entropy function in it to improve the detection of attack samples. In addition, the reconstruction error of the autoencoder is utilized to further enhance the detection of misclassified samples. For data processing, we use recursive feature elimination, which is a packet-filtering feature selection method to select the best features based on the feature scores. Differently from existing methods, we use a two-stage decision step based on the reinforced LightGBM and Autoencoder. According to our survey, it is the first time the method is proposed. The proposed method has less time overhead and improves the performance of the intrusion detection system.

In the literature [41], we use autoencoder to fit the sampled data and use the LightGBM classifier for multiclassification prediction. However, in this work, we utilize the autoencoder and a modified LightGBM model for anomaly detection. We modified the objective function of the LightGBM and designed a two-stage decision step. The main contributions of this paper are as follows:

(1) To address the dimensionality curse, we propose to use a recursive feature elimination method based on LightGBM to reduce the dimensionality of the original data. The detection efficiency of the intrusion detection system is improved.

(2) To address the problem that the standard LightGBM method cannot effectively detect difficult samples, the focal loss function is introduced into LightGBM. In addition, the improved LightGBM is combined with an autoencoder to effectively respond to zero-day attacks.

(3) Finally, we have conducted experiments on the NSL-KDD and UNSWNB5 datasets. The experiments compare not only the classical methods, but also the current state-of-the-art methods.

The remainder of this paper is structured as follows: Section 2 introduces the relevant theories. Section 3 presents our method. Section 4 provides the experimental results and discussion. Section 5 presents the conclusions and future work of this paper.

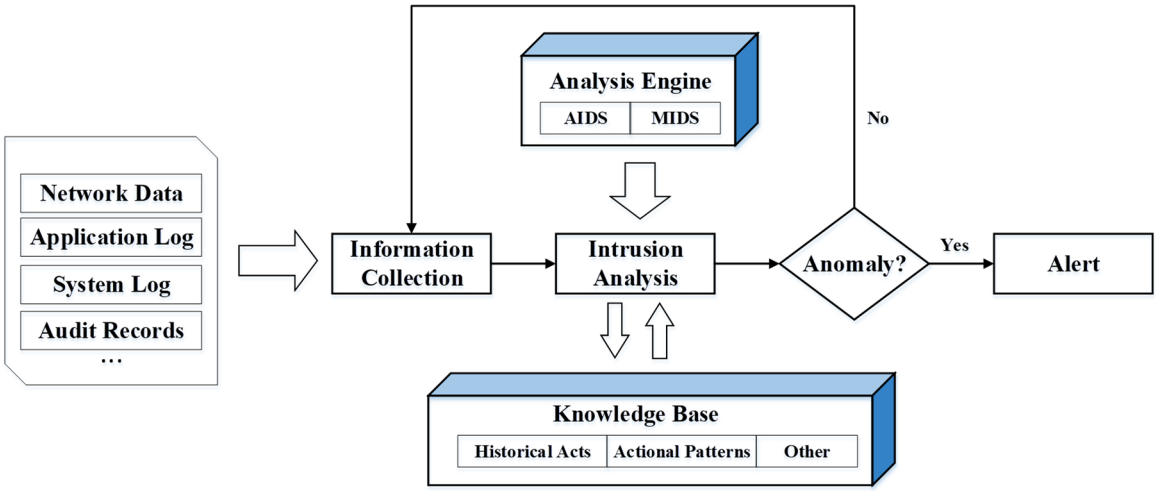


Figure 1. Block diagram of an intrusion detection system.

2. Related theories

2.1. Light gradient boosting machine

In 2017, the Microsoft team proposed the LightGBM model [42]. LightGBM has less time overhead compared to extreme gradient boosting (Xgboost). The Xgboost uses a pre-sorting algorithm when dividing the best partition nodes of the tree [43]. Since the presorting algorithm needs to traverse all the features, it leads to the inefficiency of the algorithm. In general, the time complexity of the Xgboost algorithm is proportional to the size of the data volume [44]. It means that the larger the data volume, the higher the computational overhead. The LightGBM algorithm bins the continuous features and divides different features into different bins, which reduces the computational overhead of the model. This process is called the histogram algorithm. In addition, to further improve the training efficiency of the model, LightGBM introduces the gradient-based one-sided sampling method and the mutually exclusive feature bundling algorithm. The details of the LightGBM are described in Algorithm 1.

Gradient-based one-sided sampling method. The gradient is a vector that denotes the direction of the greatest change in the value of the function, and the maximum value in that direction is the value of the gradient. In machine learning, the size of the gradient of a sample during training indicates how much that sample contributes to the final model. Because a sample with a large gradient reflects that the model has room for convergence, it is beneficial to train the model. In contrast, a sample with a small gradient indicates that the sample is already well-trained and contributes less to the training model. Therefore, it is possible to keep all of the large gradient samples, and reduce the number of less gradient samples. This process is called the gradient-based one-sided sampling method. Specifically, the gradient information of each sample is calculated. For selection purposes, the gradients of all samples are sorted in descending order according to their absolute values. After that, the samples with large gradients are retained, and some samples with small gradients are randomly excluded.

Assume the training set has n samples, denoted as $\{x_1, \ldots, x_n\}$. At each iteration, the negative gradient of the model output is denoted as $\{g_1, \ldots, g_n\}$. For the gradient boosting decision tree, its information gain is calculated as follows. Let O be the training set of the node on the decision tree, then the information gain of the split feature j of the node at d is calculated as:

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{\left(\sum_{\{x_i \in O: x_{ij} \le d\}}\right)^2}{n^j_{l|O}(d)} + \frac{\left(\sum_{\{x_i \in O: x_{ij} > d\}}\right)^2}{n^j_{r|O}(d)}\right), \tag{1}$$

where $n_O = \sum I[x_i \in O], n^j_{l|O}(d) = \sum I\left[x_i \in O: x_{ij} \le d\right]$ and $n^j_{r|O}(d) = \sum I\left[x_i \in O: x_{ij} > d\right]$.

For the GOSS algorithm, the top $a \times 100\%$ large gradient samples are selected to form set A. After that, $b \times 100\%$ small gradient samples are selected from the remaining sets to form set B. To maintain the original sample distribution, all small gradient samples in set B need to be multiplied by a coefficient $(1–a)/b$. Therefore, the final information gain is calculated as follows: Let a and b be the sampling ratios of large gradient and small gradient instances, respectively. According to the sorted instance gradient values, the first $a \times 100\%$ large gradient sample is selected, and then randomly selects $b \times 100\%$ small gradient samples from the rest of the data. After many iterations, the final calculated information gain is:

$$\widetilde{V}_j(d) = \frac{1}{n}\left(\frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n_r^j(d)}\right), \tag{2}$$

where $A_l = \{x_i \in A: x_{ij} \leq d\}, A_r = \{x_i \in A: x_{ij} > d\}, B_l = \{x_i \in B: x_{ij} \leq d\}, B_r = \{x_i \in B: x_{ij} > d\}$.

Exclusive feature bundling. GOSS reduces the number of samples, while EFB reduces the dimensionality of the features. The dimensionality of the features is another important factor that affects the time overhead. EFB uses the mutually exclusive nature of the features to reduce its dimensionality. Specifically, the EFB algorithm solves this problem by constructing a graph with weights. The nodes of the graph are represented by the features of the samples, while the weights indicate the degree of feature mutual exclusion. Finally, it is transformed into a graph coloring problem and a greedy strategy is used to solve it.

Algorithm 1: LightGBM

Input:

Training data: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}, x_i \in x, x \subseteq R, y_i \in \{-1, +1\}$;

Loss function: $L(y, \theta(x))$; // y is the true value and $\theta(x)$ is the predicted value

Iterations: *M*;

Big gradient data sampling ratio: a;

Small gradient data sampling ratio: b;

1: Exclusive Feature Bundling (EFB) techniques are used to combine mutually exclusive features of $x_i, i = \{1, \ldots, n\}$ that are not simultaneously non-zero;

2: Initialize the predicted values: $\theta_0(x) = \text{argmin}_c \sum_i^n L(y_i, c)$;

3: **For** *m*=1 to *M* do:

4: Calculate gradient absolute values: $g_i = \left|\frac{\partial L(y_i, \theta(x_i))}{\partial \theta(x_i)}\right|_{\theta(x_i)=\theta_{m-1}(x)}, i = \{1, \ldots, n\}$;

5: Resample dataset using gradient-based one-side sampling (GOSS):

$\text{topN} = a \times \text{len}(D)$; $\text{randN} = b \times \text{len}(D)$;

$\text{sorted} = \text{GetSortedIndices}(\text{abs}(g))$;

$A = \text{sorted}[1{:}\,\text{topN}]$; $B = \text{RandomPick}(\text{sorted}[\text{topN}{:}\,\text{len}(D)], \text{randN})$;

$D' = A + B$;

6: Calculate the information gains:

$$\widetilde{V}_j(d) = \frac{1}{n}\left(\frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n_r^j(d)}\right);$$

7: Get a new decision tree $\theta_m(x)'$ on set D';

8: Update $\theta_m(x) = \theta_{m-1}(x) + \theta_m(x)'$;

9: **End for**

10: **Return** $\tilde{\theta}(x) = \theta_M(x)$;

2.2. Focal loss

The focal loss function is derived from the cross entropy loss function to boost the recognition of

difficult samples [45,46]. The cross-entropy loss function is a typical objective function that measures the closeness of true and observed distributions. A smaller cross-entropy shows a better classification result. The expression of the binary classification cross-entropy (BCE) loss function is shown below:

$$\mathrm{BCE} = -y log\tilde{y} - (1-y) log(1-\tilde{y}), \tag{3}$$

where y and $\tilde{y}$ are the true label and the predicted label, respectively.

The focal loss function adds modulation factor $(1-\tilde{y})^{\gamma}$ and $\tilde{y}^{\gamma}$ to the cross-entropy function, which enables the model to assign greater learning weights to difficult samples. As such,

$$\mathrm{FL} = -y(1-\tilde{y})^{\gamma} log\tilde{y} - (1-y)\tilde{y}^{\gamma} log(1-\tilde{y}), \tag{4}$$

where $\gamma \in [0,5]$ is the focal parameter. When $\gamma = 0$, it is the cross-entropy loss function. The effect of the value of γ on the loss is shown in Figure 2.

In addition, the focus loss function introduces an alpha weighting factor. This factor is used to adjust the weighted losses of different categories. The final focal loss function is represented as:

$$\mathrm{FL} = -\alpha y(1-\tilde{y})^{\gamma} log\tilde{y} - (1-\alpha)(1-y)\tilde{y}^{\gamma} log(1-\tilde{y}), \tag{5}$$

where $\alpha \in [0,1]$.

When performing the binary classification task, the objective function of the LightGBM defaults to the binary cross-entropy loss function. As shown in Figure 2, the classification results with the focal loss function are better than the binary cross-entropy loss. In this paper, we adopt the focal loss function as the objective function in LightGBM to enhance the learning of difficult samples.

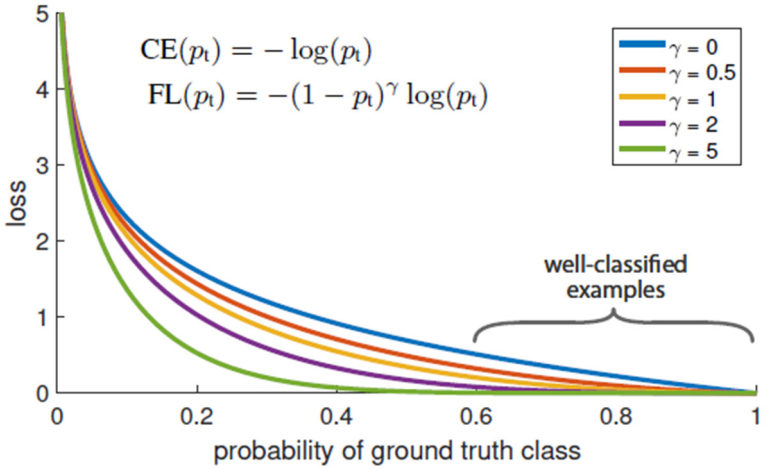


Figure 2. The effect of γ on the focal loss function [45].

2.3. Autoencoder

Autoencoders are neural networks composed of multiple layers of neurons. Essentially, it is a multilayer perceptron that uses a feed-forward algorithm [19,47]. The difference is that the autoencoder has the same number of neurons in the input and output layers, which facilitates the reconstruction of the data. In general, an autoencoder consists of input layer, encoder, middle layer, decoder, and output layer [24]. The encoder, middle layer, and decoder are also called hidden layers.

Its structure is shown in Figure 3. The size of the input and output layers is determined by the dimensionality of the dataset. The encoder is used to compress the dataset, and the decoder is used to reconstruct the dataset. The middle layer is a compressed representation of the dataset, and its size is less than the dimensionality of the dataset. In the encoder module, for input x, a compressed representation of the dataset y is obtained after mapping. Iits mathematical expression is shown in Eq (6). In the decoder module, the data x' is reconstructed using different weights w' and biases b'. This process is the opposite of the encoder, and its mathematical expression is shown in Eq (7). Usually, we use an activation function f that is nonlinear, since it can fit arbitrary functions. In addition, the autoencoder needs to define an objective function to measure the similarity of x and x'. When x and x' are close, it means that the autoencoder is well trained. In this study, we use the mean square error (MSE) function to define the loss of the autoencoder, which is one of the functions that are used the most. As such,

$$y = f(wx + b), \tag{6}$$

$$x' = f(w'y + b'), \tag{7}$$

where w is the weight coefficient of the encoder layer and b is the bias vector. w' and b' are the weight coefficients and bias vectors of the decoder layer, respectively. These parameters are updated by the backpropagation of the network. Thus,

$$\text{MSE} = \frac{1}{m}\sum(x' - x), \tag{8}$$

where m denotes the number of samples.

To avoid overfitting, adding regularization to the objective function is a common strategy. In this paper, we use L_1 regularization to impose restrictions on the weight coefficient to give them better generalization. Autoencoders that use regularization are called sparse autoencoders [48]. In addition, they can be further classified into shallow sparse autoencoders and deep sparse autoencoders, based on the number of hidden layers. The difference between them is shown in Figure 4. In the figure, $x \in R^n$ is the input data. $y \in R^m$ is the output of the middle layer. $h^l \in R^k$ is the vector of the lth hidden layer, and $x' \in R^n$ is the output vector in the sparse autoencoder. A shallow sparse autoencoder consists of three layers, i.e., an input layer, a single hidden layer (middle layer) and an output layer [41]. The deep sparse autoencoder consists of multiple hidden layers stacked on top of each other. It can learn more important implicit information from the original data than the shallow sparse autoencoder. In this study, we use a deep sparse autoencoder for our work. As such,

$$L_1 = \alpha\|\omega\|, \tag{9}$$

where $\alpha\|w\|$ denotes the L_1 regularization, which refers to the sum of absolute values of all weight parameters ω. α is the penalty factor.

According to the above theory, the original data x and the reconstructed data x' are very similar when an autoencoder is trained successfully. Their differences are also called reconstruction errors. In intrusion detection, there is a vast difference between normal samples and attack samples in the dataset. When an autoencoder trained with normal samples is used to reconstruct the attack samples, their reconstruction error will be larger than the reconstructed normal samples. Therefore, we use the reconstruction error to perform anomaly detection. Suppose the normal sample is x_+, and the attack sample is x_-. We use only the normal sample x_+ to train the autoencoder. Let the reconstructed

normal sample be x'_+ and the reconstructed attack sample be x'_-. Then we can find $x_+ - x'_+ < x_- - x'_-$. Let the current sample be noted as x_* and after autoencoder reconstruction as x'_*. Assume that $x_+ - x'_+$ is less than a certain threshold c. When $c < x_* - x'_*$, the sample can be judged as an attack sample.

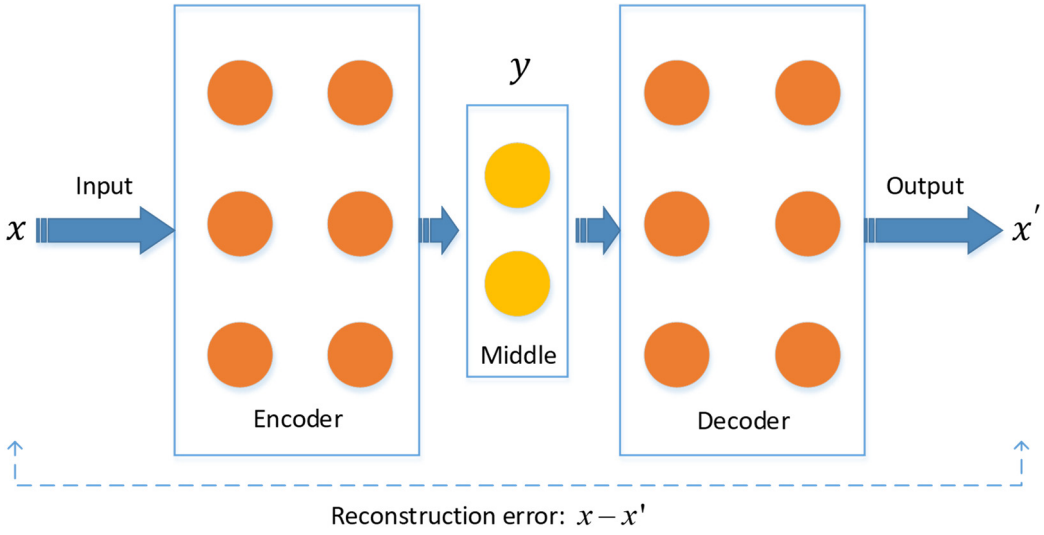


Figure 3. The structure of the autoencoder.

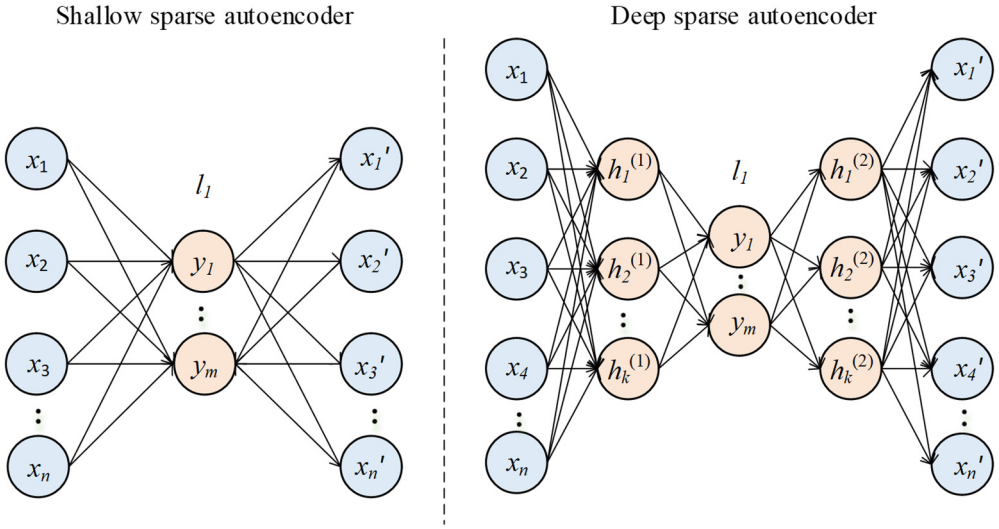


Figure 4. Shallow sparse autoencoder and deep sparse autoencoder.

3. Proposed method

3.1. Method design

Figure 5 shows the flow chart of the proposed method. It consists of four parts including data preprocessing, feature selection, model training and classification decision. The details are described below.

Data pre-processing. In the data preprocessing, since the model cannot handle non-numerical features, the training and test sets are first numerized. Sparse coding is helpful to enrich the data features. In addition to the category features, other non-numerical features are sparsely coded by the one-hot coding method in this paper. For example, the non-numeric feature "Protocol" has three values [TCP, UDP, ICMP], which can be coded as [100, 010, 001]. For the numerical features, the variation range of the values is different, which is not conducive to the training of the model. Therefore, in order to reduce the convergence time of the model, the normalization method is needed. In this paper, the maximum-minimum normalization method is used to scale the values in the range of [0, 1]. The maximum-minimum normalization method is represented as follows:

$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{10}$$

where $x_{\max}$ and $x_{\min}$ denote the max and min values of feature x, respectively.

Feature selection. In the feature selection, the recursive feature elimination method is adopted for feature selection. The recursive feature elimination method is a wrapper method that selects features based on the performance of the classification algorithm. Essentially, the recursive feature elimination method is a greedy algorithm. The recursive deletion is performed based on the ranking score of the features. The method needs to iterate through all the features and remove those that have little impact on the model performance until the desired number of features is satisfied.

Model training. In the model training, we use the process described in Algorithm 1 to build the model. First, the iteration number of the model is set. According to the number of iterations, several different decision trees are trained. Each decision tree is built relying on the performance of the previous decision tree. After several iterations, an integrated model consisting of several weak decision trees is obtained. In particular, we use the focal loss function instead of the default cross-entropy loss function in the definition of the objective function.

Classification decision. In the classification decision, there are two decision phases. In the first decision phase, the LightGBM with the introduction of the focal loss function is used for pre-classification. In the second decision stage, secondary classification is performed using a sparse autoencoder for samples predicted as normal in the first decision stage. Generally, if the sample is judged to be abnormal, it is finally predicted to be attack. On the contrary, if the sample is judged as normal, then it will finally be predicted as normal. The two-stage classification decision step enables the intrusion detection system to improve the accuracy, and the ability to detect unknown attacks.

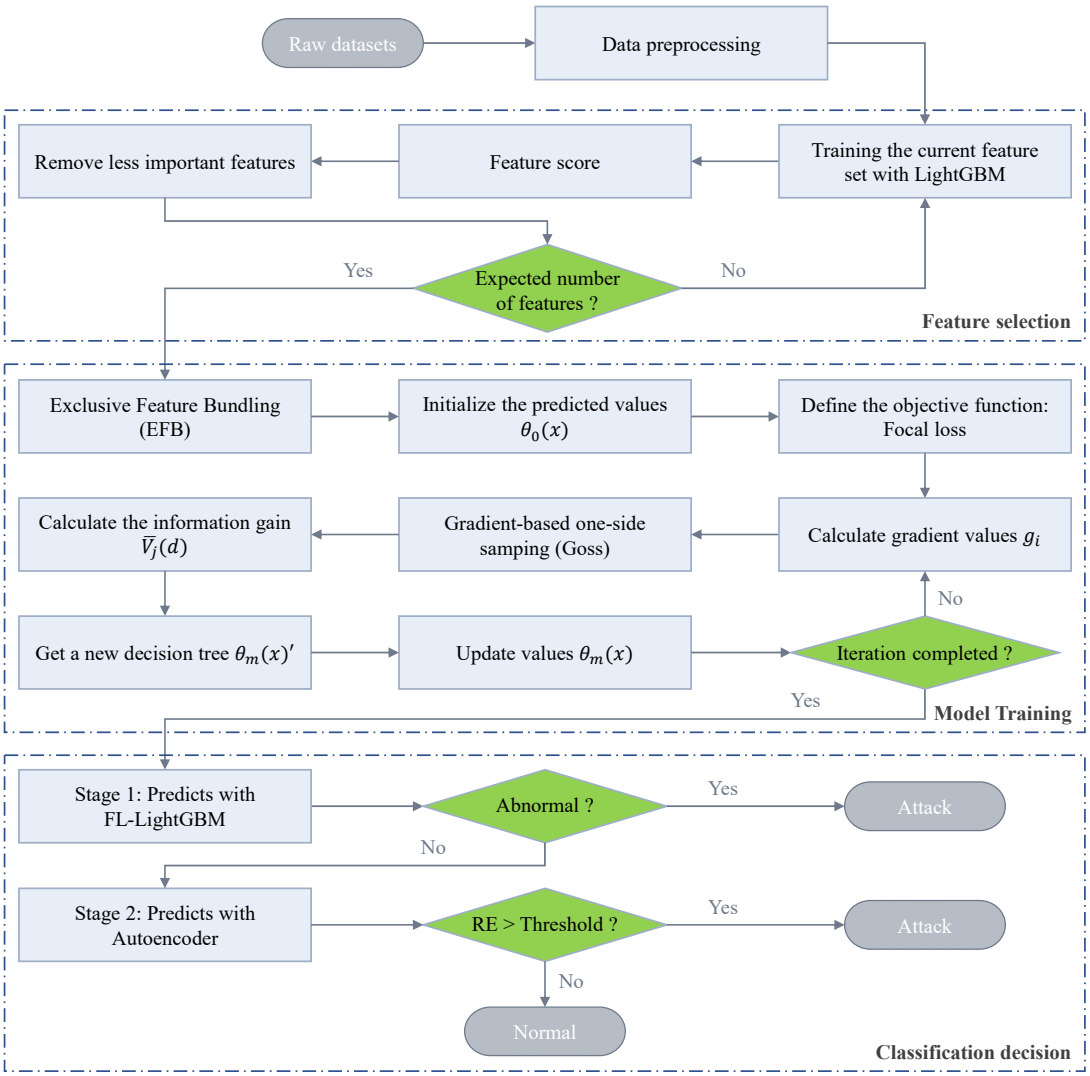


Figure 5. Flow chart of the proposed method.

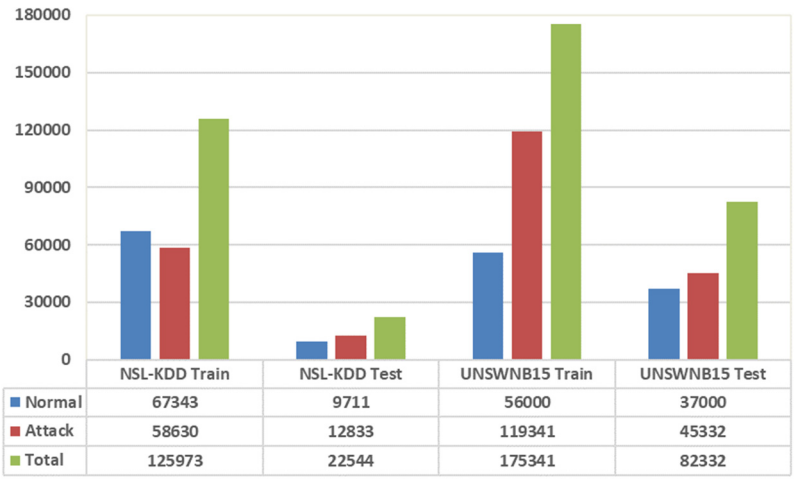
3.2. Dataset description and preprocessing

Description. The NSL-KDD dataset is an improved version of the KDDCup99 dataset [49]. The KDDCup99 dataset is derived from the MIT Lincoln laboratory's intrusion detection evaluation project, which is data collected from nine weeks of network connectivity and system audits. According to Tavallaee et al. [49], the training and testing sets in the KDDCup99 dataset contain 78% and 75% of redundant data, respectively. To address the redundancy problem in the KDDcup99 dataset, Tavallaee extracted the NSL-KDD dataset without redundant data from the KDDCUp99 dataset. The improved NSL-KDD dataset has the following advantages: (1) There are no duplicate records in the training and test sets, which makes the classifier not affected by duplicate records. (2) The number of records in the training and test sets are reasonable, and they do not require high performance of the computer. As shown in Table 1, the NSL-KDD dataset consists of 42 features. The values of each feature are divided into numerical and non-numerical types. Among them, the values of three features including protocols, services and flags are non-numeric types, and the rest is numeric types. The NSL-KDD dataset contains

four attack categories, namely Dos, Probe, user-to-root (U2R) and root-to-local (R2L). All these attack categories are considered anomalies. The size of the NSL-KDD dataset is shown in Figure 6. The training set and test set contain 125,973 and 22,544 records, respectively. Among them, the proportion of normal samples and attack samples in the training set are 53.46% and 46.54%, respectively. In the test set, the proportion of normal samples and attack samples are 43.08% and 56.92%, respectively. It is important to note that the attack samples are composed of a variety of different attack types. In the test set, an additional 18 attacks are contained, which means that the test set has different attack patterns [21]. Therefore, it can be used to simulate the detection of zero-day attacks.

The UNSWNB15 dataset was created by the Australian Center for Cyber Security in 2015 using the IXIA tool [50]. The dataset contains a total of 2 million records that were saved in four different CSV files [51]. To facilitate the use of the dataset, the UNSWNB15 dataset was divided into a training set and a test set, named UNSWNB15Train and UNSWNB15Test, respectively. As shown in Table 1, a total of 49 features are included in the dataset. Among them, three features containing protocol, service and state are non-numeric types, and the rest are numeric types. Different from the NSL-KDD dataset, the UNSWNB15 dataset includes nine new attack types: Backdoor, Shellcode, Reconnaissance, Worms, Fuzzers, DOS and Generic. In this paper, we used the UNSWNB15Train and UNSWNB15Test datasets for our experiments. The information on this dataset is shown in Figure 6. Specifically, the training set and test set contain 175,341 and 82,332 samples, respectively. In the training set, the proportion of normal samples and attack samples are 31.94% and 68.06%, respectively. In the test set, the proportion of normal samples and attack samples are 44.94% and 55.06%, respectively.

Preprocessing. The NSL-KDD dataset sample contained 41-dimensional features. Because the model cannot handle symbolic data, it is necessary to convert the characteristics of symbolic types into numeric kinds. In addition, the data are encoded with the one-hot method. Specifically, the protocol feature contains three values, represented by three numbers containing 0 and 1. The service features have 70 values, so they are represented by 70 numbers consisting of 0 and 1. The flag feature has 11 values, so it is represented by 11 numbers containing 0 and 1. After one-hot encoding, the dimension of the NSL-KDD dataset is expanded to 122. Similarly, the UNSWNB15 dataset is processed in the same way. Finally, the dimensionality of the UNSWNB15 dataset becomes 196.



	NSL-KDD Train	NSL-KDD Test	UNSWNB15 Train	UNSWNB15 Test
Normal	67343	9711	56000	37000
Attack	58630	12833	119341	45332
Total	125973	22544	175341	82332

Figure 6. Statistics on the two datasets.

Table 1. Dataset feature.

Dataset	Feature name	Size
NSL-KDD	duration, protocols_types, services, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_shells, num_access_files, num_outbound_cmds, is_hot_login, Is_guest_login, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, label.	42
UNSWNB15	srcip, sport, dstip, dsport, protocol, state, dur, sbytes, dbytes, sttl, dttl, sloss, dloss, service, sload, dload, skts, dpkts, swin, dwin, stcpb, dtcpb, smeansz, dmeansz, trans_depth, res_bdy_len, sjit, djit, stime, ltime, sintpkt, dintpkt, tcprtt, synack, ackdat, is_sm_ips_ports, ct_state_ttl, ct_flw_http_mthd, is_ftp_login, ct_ftp_cmd, ct_srv_src, ct_srv_dst, ct_dst_ltm, ct_src_ ltm, ct_src_dport_ltm, ct_dst_sport_ltm, ct_dst_src_ltm, attack_type, label.	49

3.3. Feature selection and reconstruction

In this study, the recursive feature elimination method was used to reduce the data dimensionality. The recursive feature elimination method selects features based on feature importance [52]. First, all the original features are trained using the LightGBM classifier to obtain the weight coefficients of each feature. After that, the features with the smallest weight coefficients are selected and removed from the original feature set to obtain a new subset of features. Finally, the new feature subset is trained again using the LightGBM classifier to obtain the weight coefficients. This process is repeated until the required number of features is obtained. Algorithm 2 describes this process. According to the results in Section 4.3.1, we select 40 and 60 features for the NSL-KDD and UNSWNB15 datasets, respectively.

The sparse autoencoder is used for the second stage of prediction. First, the autoencoder is trained on the normal class samples from the training set. Then, the trained model is reconstructed from the test set. In this study, the sparse autoencoder is composed of 7 hidden layers. Among them, the structures of the encoder are 64 and 32, denoting the number of nodes in each layer. The size of the middle layer is 16. Since the encoder and decoder are symmetric structures, the structures of the decoder are 32 and 64, respectively. The sizes of the input and output layers are 122 for the NSL-KDD dataset, and 196 for the UNSWNB15 dataset. In particular, the Relu function is used as an activation function between neurons.

Algorithm 2: RFE

Input:

Original feature set $S = [1,2,3, \ldots, D]$ // D denotes the features in the sample
Expected number of features: N
Output:
Feature ordering set $R = []$
Start: Initialize feature weights $w_i = 1 \ (i = 1, \ldots, d)$ // d denotes the dimensionality of the features in the original dataset
1: **if len(S)≠N, do**:
2: Train the current feature set S with the LightGBM classifier
3: Calculate the feature weight coefficients in set S
4: Find the feature with the smallest weight coefficient: $r = \text{argmin}_j(w_j) \ (j = 1, \ldots, d)$
5: Update feature ordering set: $R = [r, R]$
6: Remove less important features: $S = S - [r]$
7: d=d-1
8: **until len(S)**=N
9: **end if**

4. Results and discussion

4.1. Experiment setup

The experiment was conducted on a Dell host, and was configured as follows: 32 G RAM, Intel Core i7-9700 CPU, and Radeon Rx 550x. To speed up the training of the model, we used the GPU on a Linux server to train the autoencoder. We used tensorflow with version 2.2.0 as the backend. Furthermore, sklearn and keras were used to process the dataset. The native lgb [53] library was used to build the model. In the experiments, we set the number of iterations as 200, and the random seed as 42. In addition, for the NSL-KDD dataset, we set α = 0.1 and γ = 0.9. For the UNSWNB15 dataset, we set α = 0.2 and γ = 5.

4.2. Evaluating metrics

In this paper, we used accuracy, precision, recall and F1 score to evaluate the performance of the model. The accuracy represents the proportion of instances that are correctly predicted to account for all instances. The precision represents the proportion of correctly predicted attack instances to all predicted attack instances. The recall represents the proportion of attack instances that were correctly predicted by the classifier. The F1 score is a metric of balancing precision and recall. The formula of each metric is determined by Eqs (11)–(14).

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}, \tag{11}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}, \tag{12}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}, \tag{13}$$

$$\text{F1 score} = 2/(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}), \tag{14}$$

where TP represents the number of attack instances that are correctly predicted. TN represents the number of correctly predicted normal instances. FP represents the number of normal instances that are mispredicted. FN represents the number of attack instances that are mispredicted.

4.3. Performance

4.3.1. Feature dimensionality reduction

In order to find the appropriate number of features, different numbers of features were used for comparison. Figures 7 and 8 show the accuracy of the two datasets for the different number of features. As can be seen from Figure 7, the highest accuracy is obtained on the NSL-KDD dataset when the number of features is 40. However, for the UNSWNB15 dataset, the number of features is 60. Therefore, we set the number of features to 40 and 60 for the NSL-KDD and UNSWNB15 datasets, respectively.

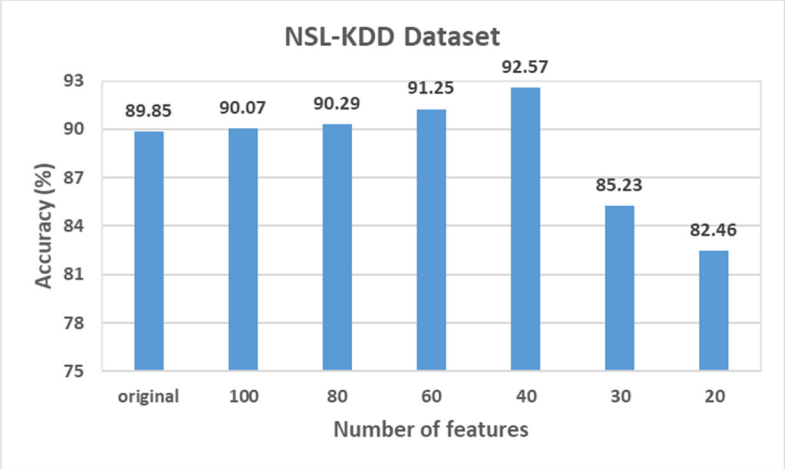


Figure 7. The number of features for the NSL-KDD dataset.

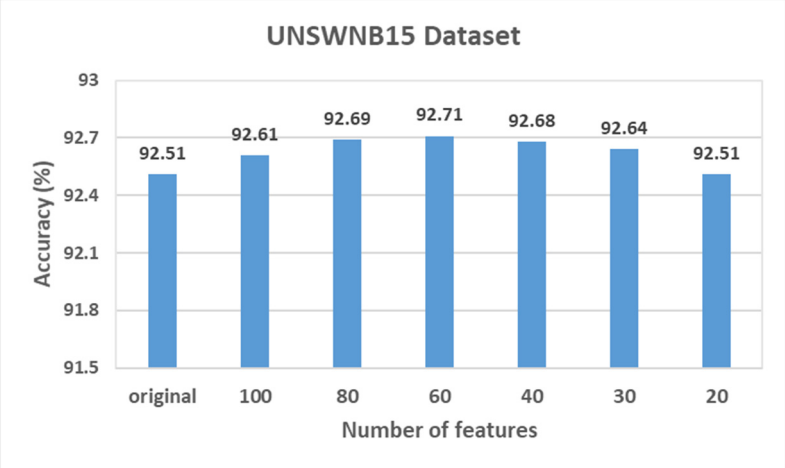


Figure 8. The number of features for the UNSWNB15 dataset.

4.3.2. Evaluation on different metrics

Figure 9 shows the performance of the proposed method on each metric. On the NSL-KDD dataset, the proposed method achieves 92.57%, 89.93%, 97.91% and 93.75% in accuracy, precision, recall and F1 score, respectively. Among them, recall is the best, which indicates that the proposed method can detect almost all attack classes. The confusion matrix on the NSL-KDD dataset is given in Table 2. It can be seen that only 267 attack classes are not recognized by the proposed method. For the UNSWNB15 dataset, the proposed method achieves 92.71%, 93.43%, 93.32% and 93.38% in terms of accuracy, precision, recall and F1 score, respectively. It can be concluded that the performance of all metrics on this dataset is more balanced. Table 3 shows the confusion matrix on the UNSWNB15 dataset.

Table 4 shows the time overhead (the sum of training time and prediction time for the proposed method). When the feature selection method is not used, the time overhead on the two datasets are 8.22 and 18.6 seconds, respectively. In contrast, when using the recursive feature elimination method, the time overhead of the proposed method is 5.9 and 17.25 seconds, respectively. It indicates that the decision efficiency of the model is improved after using the recursive feature reduction method.

Table 2. Confusion matrix for the NSL-KDD dataset.

NSL-KDD	Predicted label		
		Normal	Attack
True label	Normal	8305	1406
	Attack	267	12,566

Table 3. Confusion matrix for the UNSWNB15 dataset.

UNSWNB15	Predicted label		
		Normal	Attack
True label	Normal	34,029	2971
	Attack	3026	42,306

Table 4. Prediction time of the proposed method for the original features and RFE.

Dataset	Original	RFE
NSL-KDD	8.22 s	5.9 s
UNSWNB15	18.6 s	17.25 s

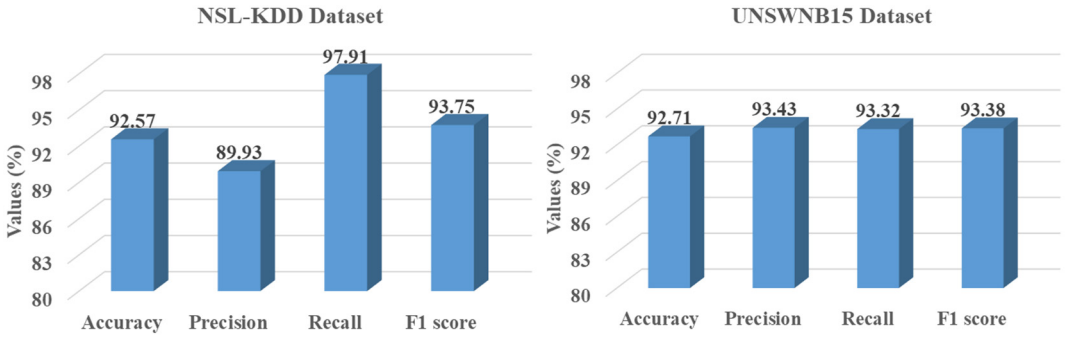


Figure 9. Evaluate the accuracy, precision, recall and F1 score for both datasets.

4.3.3. Ablative analysis

In this section, we perform the ablation analysis of the proposed method. The LightGBM model without introducing the focal loss function is taken as the base model, which is LGBM. The model introducing the focal loss function is the improved model, which is FL_LGBM. The proposed model is FL_LGBM-AE. As shown in Figure 10, the performance of these three different models in terms of accuracy and F1 score is shown. The F1 score reflects the harmonic value of precision and recall. It can be concluded that, for both datasets, the FL_LGBM model has a larger improvement compared to the base model. It shows that the focal loss function introduced in LGBM is valid. Furthermore, on the NSL-KDD dataset, compared to FL_LGBM, the FL_LGBM-AE model improved by 11.5% and 13.08% in accuracy and F1 score, respectively. The FL_LGBM-AE model also performs better than the FL_LGBM model on the UNSWNB15 dataset.

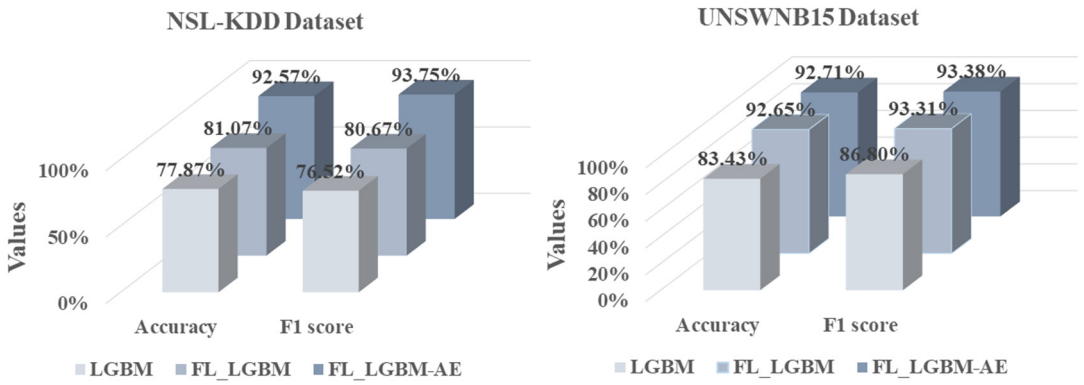


Figure 10. Ablation analysis of the proposed method on two datasets.

4.3.4. Hyperparametric analysis

In the proposed method, the learning rate and threshold are two important hyperparameters. The proposed method enables the model to learn from difficult samples by the introduction of the focal loss function. It makes the learning rate of the proposed model more important.

Figures 11 and 12 show the effect of different learning rates on these two datasets. On the NSL-KDD dataset, the model performs the worst when the learning rate is equal to 0.003. When the learning rate equals 0.0035, the model without the autoencoder performs the best, reaching an accuracy of 87.85%. As the learning rate increases, the accuracy of the model gradually decreases. Conversely, the accuracy of the model using autoencoder increased as the learning rate increased. The reason is that, when the learning rate increases, the learning pace of the model becomes larger, resulting in the model not converging to the global minimum. As the learning rate becomes larger, it misclassifies most of the attack classes as normal classes. When using the autoencoder, the attack samples that are misclassified as normal classes are accurately identified. The highest accuracy of the model is obtained when the learning rate is equal to 0.03. Overall, most models that used autoencoders were above 90% accurate, which was higher than the models that did not use autoencoders.

On the UNSWNB15 dataset, the performance of the models is stable, whether or not we use autoencoders. When the learning rate reaches 0.004, the accuracy of the model with the autoencoder

is slightly higher than the model without the autoencoder. After this point, the accuracy of the model with the autoencoder is slightly smaller than the model without the autoencoder. The possible reason is that the accuracy of the model without the autoencoder is already over 90%. When using the autoencoder, the accuracy of the model only slightly improves. As the learning rate increases, the model performance deteriorates and becomes worse with the autoencoder. However, it cannot demonstrate that the proposed model is ineffective. Autoencoders can still play an important role, as long as a suitable learning rate is found. As can be seen in Figure 12, the best performance of the model is obtained when the learning rate is 0.004.

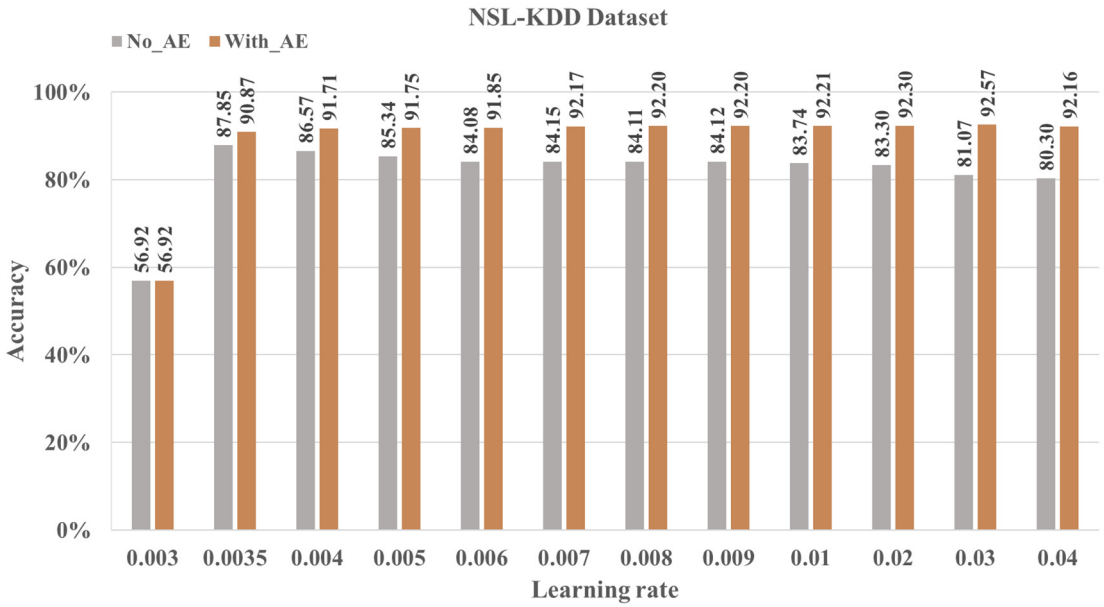


Figure 11. Effects of different learning rates for No_AE and With_AE on the NSL-KDD dataset.

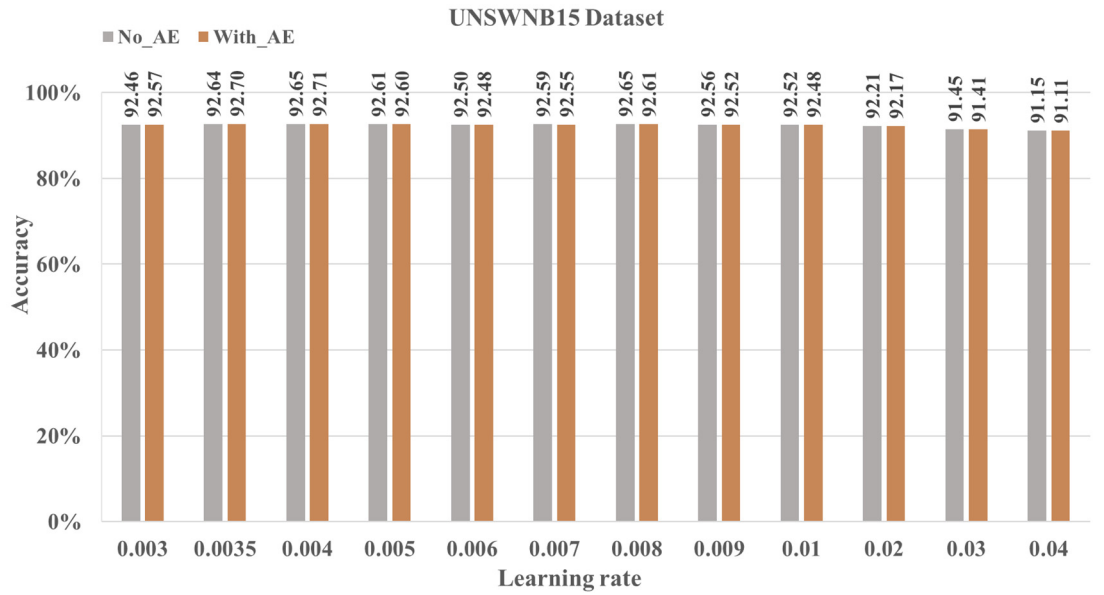


Figure 12. Effects of different learning rates for No_AE and With_AE on the UNSWNB15 dataset.

When training datasets with autoencoders, the range of reconstruction errors produced by different datasets is different. Figure 13 presents the mean squared error for the both datasets. Figures 14 and 15 show the effect of different thresholds on the model. For the NSL-KDD dataset, the proposed method produces the best results when the threshold reaches 0.00095. After that, as the threshold increases, the accuracy of the model gradually decreases. The reason is that the increase in threshold causes the autoencoder to fail to identify the attack samples with large reconstruction errors. For the UNSWNB15 dataset, the proposed method achieves the best results when the thresholds are 0.0095. As the threshold increases, the performance of the model plateaus. Most of the attack samples with large reconstruction errors have already been identified, meaning increasing the threshold has no influence.

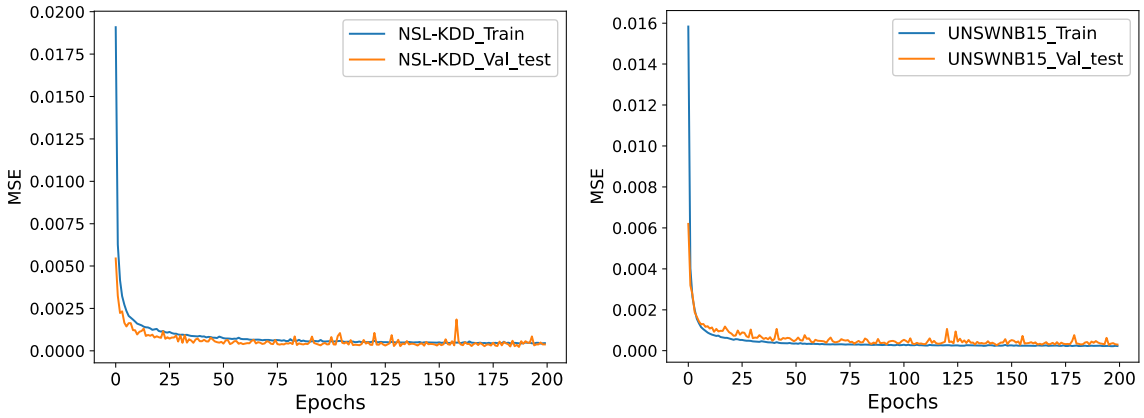


Figure 13. Reconstruction errors of sparse autoencoders.

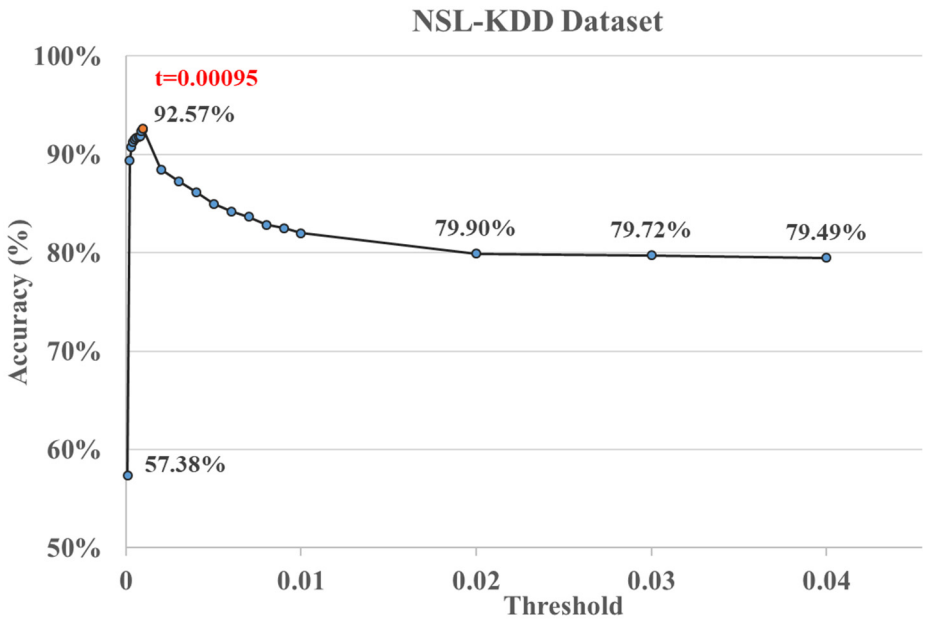


Figure 14. The effect of thresholds on the NSL-KDD dataset.

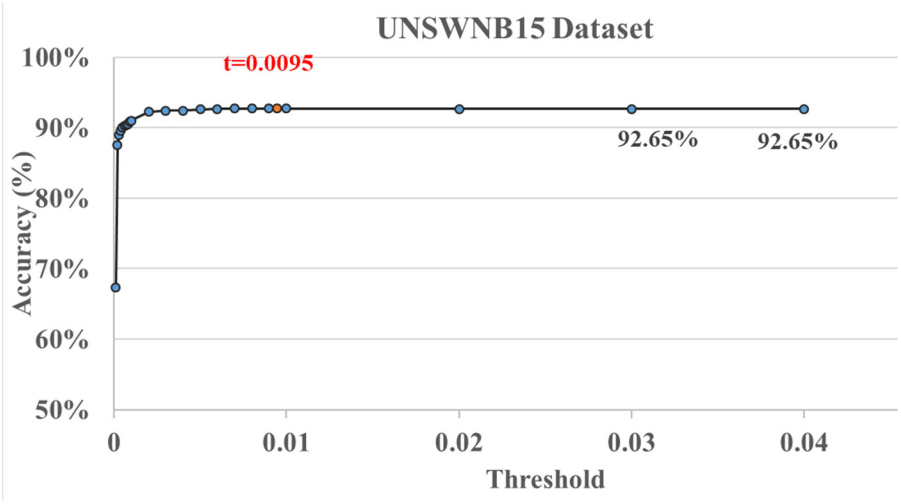


Figure 15. The effect of thresholds on the UNSWNB15 dataset.

4.4. Comparison

4.4.1. Comparison with classical methods

Tables 5 and 6 show the performance of different methods on the four evaluation metrics. For the NSL-KDD dataset, the proposed method achieves the best results in terms of recall rate, F1 score and accuracy. It is notable that the random forest (RF), gradient boosting decision Tree (GBDT) and Xgboost models all achieve 90% accuracy. For the UNSWNB15 dataset, the proposed method achieves the best performance in terms of precision, F1 score and accuracy. In contrast, the other methods did not exceed 81% in precision, and did not reach 90% in accuracy. Although these methods have a higher recall, they perform poorly on other metrics. In particular, the proposed method exceeds 90% on the F1 score, which indicates that our method performs more balanced in precision and recall. The other methods have a significant imbalance in precision and recall. Overall, the proposed method has better performance on these two datasets, which proves the effectiveness of the proposed method.

Table 5. Comparison of the proposed method with classical methods on the NSL-KDD dataset.

Metrics / Classifiers	Precision (%)	Recall (%)	F1 score (%)	Accuracy (%)
DT	89.80	97.12	93.32	92.08
SVM	89.89	94.45	92.12	90.80
RF	90.05	96.89	93.35	92.14
GBDT	90.00	97.09	93.41	92.21
Xgboost	90.00	96.94	93.34	92.13
Adaboost	89.68	97.15	93.27	92.02
Proposed method	89.93	**97.91**	**93.75**	**92.57**

Table 6. Comparison of the proposed method with classical methods on the UNSWNB15 dataset.

Metrics / Classifiers	Precision (%)	Recall (%)	F1 score (%)	Accuracy (%)
DT	80.42	95.40	87.27	84.68
SVM	75.03	99.58	85.58	81.52
RF	77.51	99.32	87.07	83.76
GBDT	76.04	99.51	86.20	82.46
Xgboost	76.47	98.86	86.23	82.26
Adaboost	76.57	98.85	86.30	82.72
Proposed method	**93.43**	93.32	**93.38**	**92.71**

Figures 16 and 17 show the time overhead for the different methods. It can be seen that the support vector machine (SVM) model has the highest time overhead on these two datasets. The reason is that, after mapping the data to the nonlinear space, the SVM needs to calculate the maximum interval of the decision boundary, which increases the computational overhead. As the number of samples increases, the time overhead becomes larger. It shows that the SVM is not suitable for handling large datasets. In addition, the decision tree (DT) model has the least time overhead owing to the simple decision tree algorithm. The proposed method adds a portion of time overhead due to the use of the focal loss function. The time overheads of the proposed method are 5.9 and 17.25 seconds for these two datasets, respectively. Although the proposed method is not optimal in terms of time overhead, it is still less than the overheads of the SVM, RF, GBDT and Adaboost models. It means that the proposed method still has an advantage in terms of time overhead.

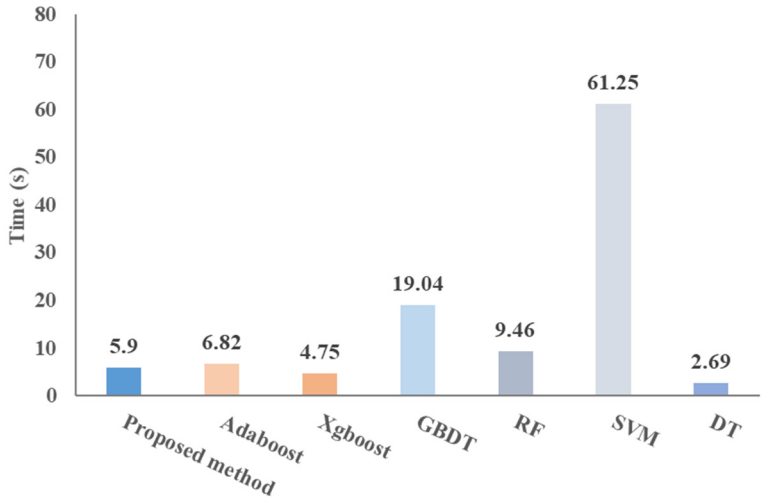


Figure 16. The time cost comparison between the proposed method and the classical method for the NSL-KDD dataset.

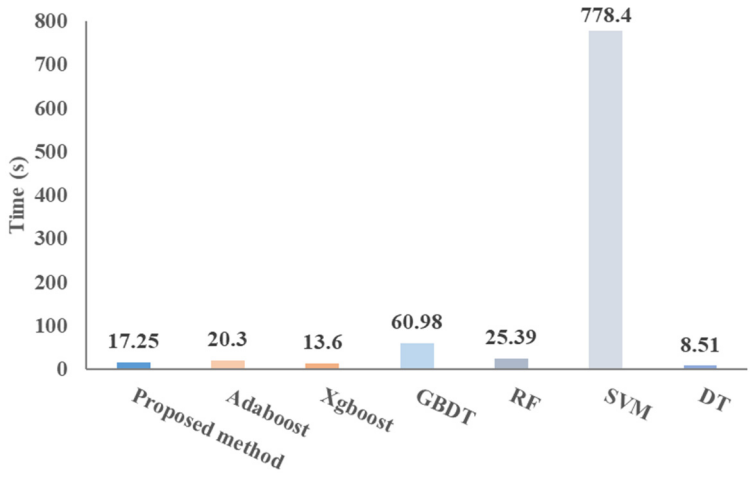


Figure 17. The time cost comparison between the proposed method and the classical method for the UNSWNB15 dataset.

4.4.2. Comparison with advanced methods

Table 7. Comparing our method with existing techniques on two datasets.

Dataset	Method	Accuracy (%)	Recall (%)	Precision(%)	F1 score(%)
NSL-KDDTest	RandomTree+NBtree [13]	89.24	N/A	N/A	N/A
	CBR-CNN [19]	89.41	N/A	N/A	N/A
	AE-LSTM [24]	89.00	88.00	N/A	N/A
	AIDA [20]	92.41	92.00	**94.52**	93.24
	STL [27]	88.39	95.95	85.44	90.40
	AE-IDS [30]	84.21	80.37	87.00	81.98
	MFFSEM [31]	84.33	96.43	74.61	84.13
	Our Method	**92.57**	**97.91**	89.93	**93.75**
UNSWNB15Test	Voting-CMN [15]	89.29	**99.28**	82.37	90.04
	RepTree [26]	88.95	N/A	N/A	N/A
	ANN [32]	86.71	**98.06**	81.54	89.04
	MFFSEM [31]	88.85	80.44	**93.88**	86.64
	LOF [33]	91.86	N/A	N/A	N/A
	GAA [54]	91.80	91.00	N/A	N/A
	GBM [55]	91.31	N/A	N/A	N/A
	Our Method	**92.71**	93.32	93.43	**93.38**

Table 7 shows the comparison of our method with existing methods. We present the results from these publications. For the NSL-KDD dataset, it can be seen that our method performs the best in terms of accuracy, recall and F1 score, reaching 92.57%, 97.91% and 93.75%, respectively. In terms of precision, the literature [20] performs the best. For the UNSWNB15 dataset, our method performs the best in terms of accuracy and F1 score, reaching 92.71% and 93.38%, respectively. In terms of recall,

literature [15] and literature [32] obtained 99.28% and 98.06%, respectively, which are the best among all methods. In terms of precision, the literature [31] obtained the best results, achieving 93.88%. The reason is that the literature [15] and [31] used the integration of several different classifiers. The literature [32] used a deep learning approach based on artificial neural networks. However, they did not achieve an accuracy of 90%. In contrast, our method exceeded 90% in all metrics, showing that our method is more effective. This is owed to our proposed two-stage decision step.

5. Discussion, conclusions, limitations and future research

In this work, we proposed a two-stage intrusion detection framework based on LightGBM and autoencoders. In this framework, to solve the curse of dimensionality, the recursive feature elimination method was used for feature selection. In addition, the focal loss function was introduced in LightGBM to enhance the learning of difficult samples. In order to improve the detection capability of zero-day attacks, this study divided the decision-making process into two stages, thereby improving the performance of the intrusion detection system. The experiments were performed on the NSL-KDD and UNSWNB15 datasets, and the accuracy rates were 92.57% and 92.71%, respectively. The recall reached 97.91% and 93.32%, respectively. Experiments compared classical methods and advanced methods respectively, and the results proved the effectiveness of the proposed method. We can conclude that the proposed method can improve the efficiency and performance of intrusion detection systems.

Although the proposed method achieves a high recall for the NSL-KDD dataset, the recall for UNSWNB15 still needs to be improved. In addition, the precision of our method on both datasets is not yet advanced. This means that we need to further optimize the model. In future work, we will mainly focus on two aspects: First, since the threshold of the autoencoder is the key factor affecting the model, we will develop a method to set the threshold automatically. Second, we segment the attack types and adopt a suitable sampling method to further improve the model performance.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61862007, and the Guangxi Natural Science Foundation under Grant 2020GXNSFBA297103.

Conflict of interest

We declare that there are no conflicts of interest.

References

1. *An Article to Understand Ransomware Attacks: Characteristics, Trends and Challenges.* Available from: https://www.secrss.com/articles/33928
2. D. J. Du, M. G. Zhu, M. R. Fei, M. Fei, S. Bu, L. Wu, et al., A Review on cybersecurity analysis, attack detection, and attack defense methods in cyber-physical power systems, *J. Mod. Power Syst. Clean Energy*, **2022** (2022), 1–18. https://doi.org/10.35833/MPCE.2021.000604
3. *Ransomware Attack Forces Shutdown of Largest Fuel Pipeline in the U.S.* Available from:

https://www.cnbc.com/2021/05/08/colonial-pipeline-shuts-pipeline-operations-after-cyberattack.html
4. P. R. Kanna, P. Santhi, Unified deep learning approach for efficient intrusion detection system using integrated spatial–temporal features, *Knowl. Based Syst.*, **226** (2021), 107132. https://doi.org/10.1016/j.knosys.2021.107132
5. M. Bijone, A survey on secure network: intrusion detection & prevention approaches, *Am. J. Inf. Syst.*, **4** (2016), 69–88. https://doi.org/10.12691/ajis-4-3-2
6. A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges, *Cybersecurity*, **2** (2019), 1–22. https://doi.org/10.1186/s42400-019-0038-7
7. A. Thakkar, R. Lohiya, A review of the advancement in intrusion detection datasets, *Procedia Comput. Sci.*, **167** (2020), 636–645. https://doi.org/10.1016/j.procs.2020.03.330
8. C. Guo, Y. Ping, N. Liu, S. S. Luo, A two-level hybrid approach for intrusion detection, *Neurocomputing*, **214** (2016), 391–400. https://doi.org/10.1016/j.neucom.2016.06.021
9. *Intrusion Detection System*. Available from: https://blog.51cto.com/u_12632800/4810474
10. I. F. Kilincer, F. Ertam, A. Sengur, Machine learning methods for cyber security intrusion detection: Datasets and comparative study, *Comput. Networks*, **188** (2021), 107840. https://doi.org/10.1016/j.comnet.2021.107840
11. X. Xue, Y. Jia, Y. Tang, Expressway project cost estimation with a convolutional neural network model, *IEEE Access*, **8** (2020), 217848–217866. https://doi.org/10.1109/ACCESS.2020.3042329
12. N. Sameera, M. Shashi, Encoding approach for intrusion detection using PCA and KNN classifier, in *Proceedings of the Third International Conference on Computational Intelligence and Informatics*, **1090** (2020), 187–199. https://doi.org/10.1007/978-981-15-1480-7_15
13. J. Kevric, J. Samed, S. Abdulhamit, An effective combining classifier approach using tree algorithms for network intrusion detection, *Neural Comput. Appl.*, **28** (2017), 1051–1058. https://doi.org/10.1007/s00521-016-2418-1
14. M. Yousefnezhad, J. Hamidzadeh, M. Aliannejadi, Ensemble classification for intrusion detection via feature extraction based on deep Learning, *Soft Comput.*, **25** (2021), 12667–12683. https://doi.org/10.1007/s00500-021-06067-8
15. R. Swami, M. Dave, V. Ranga, Voting-based intrusion detection framework for securing software-defined networks, *Concurrency Comput. Pract. Exper.*, **32** (2020), e5927. https://doi.org/10.1002/cpe.5927
16. A. Basati, M. M. Faghih, PDAE: Efficient network intrusion detection in IoT using parallel deep auto-encoders, *Inf. Sci.*, **598** (2022), 57–74. https://doi.org/10.1016/j.ins.2022.03.065
17. A. S. Almogren, Intrusion detection in edge-of-things computing, *J. Parallel Distrib. Comput.*, **137** (2020), 259–265. https://doi.org/10.1016/j.jpdc.2019.12.008
18. M. S. ElSayed, N. Le-Khac, M. A. Albahar, A. Jurcut, A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique, *J. Network Comput. Appl.*, **191** (2021), 1–18. https://doi.org/10.1016/j.jnca.2021.103160
19. N. Chouhan, A. Khan, Network anomaly detection using channel boosted and residual learning based deep convolutional neural network, *Appl. Soft Comput.*, **83** (2019), 1–18. https://doi.org/10.1016/j.asoc.2019.105612
20. G. Andresini, A. Appice, N. D. Mauro, C. Loglisci, D. Malerba, Exploiting the auto-encoder residual error for intrusion detection, in *2019 IEEE European Symposium on Security and Privacy*

Workshops (EuroS&PW), (2019), 281–290. https://doi.org/10.1109/EuroSPW.2019.00038
21. R. C. Aygun, A. G. Yavuz, Network anomaly detection with stochastically improved autoencoder based models, in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, (2017), 192–198. https://doi.org/10.1109/CSCloud.2017.39
22. Y. Yang, K. Zheng, C. Wu, Y. Yang, Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network, *Sensors*, **19** (2019), 2528. https://doi.org/10.3390/s19112528
23. B. Min, J. Yoo, S. Kim, D. Shin, Network anomaly detection using memory-augmented deep autoencoder, *IEEE Access*, **9** (2021), 104695–104706. https://doi.org/10.1109/ACCESS.2021.3100087
24. E. Mushtaq, A. Zameer, M. Umer, A. A. Abbas, A two-stage intrusion detection system with auto-encoder and LSTMs, *Appl. Soft Comput.*, **121** (2022), 1–16. https://doi.org/10.1016/j.asoc.2022.108768
25. M. Al-Qatf, Y. Lasheng, M. Al-Habib, K. Al-Sabahi, Deep learning approach combining sparse autoencoder with SVM for network intrusion detection, *IEEE Access*, **6** (2018), 52843–52856. https://doi.org/10.1109/ACCESS.2018.2869577
26. M. Belouch, S. E. Hadaj, M. Idhammad, A two-stage classifier approach using reptree algorithm for network intrusion detection, *Int. J. Adv. Comput. Sci. Appl.*, **8** (2017), 389–394. https://doi.org/10.14569/IJACSA.2017.080651
27. A. Javaid, W. Q. Sun, A. Y. Javaid, M. Alam, A deep learning approach for network intrusion detection system, in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, **3** (2016), 1–6. http://dx.doi.org/10.4108/eai.3-12-2015.2262516
28. L. X. Zhang, D. Ma, A hybrid approach toward efficient and accurate intrusion detection for in-vehicle networks, *IEEE Access*, **10** (2022), 10852–10866. http://dx.doi.org/10.1109/ACCESS.2022.3145007
29. J. Gu, L. H. Wang, H. W. Wang, S. S. Wang, A novel approach to intrusion detection using SVM ensemble with feature augmentation, *Comput. Secur.*, **86** (2019), 53–62. https://doi.org/10.1016/j.cose.2019.05.022
30. C. Ieracitano, A. Adeel, F. C. Morabito, A. Hussain, A novel statistical analysis and autoencoder driven intelligent intrusion detection approach, *Neurocomputing*, **387** (2020), 51–62. https://doi.org/10.1016/j.neucom.2019.11.016
31. H. Zhang, J. L. Li, X. M. Liu, C. Dong, Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection, *Future Gener. Comput. Syst.*, **122** (2021), 130–143. https://doi.org/10.1016/j.future.2021.03.024
32. S. M. Kasongo, Y. X. Sun, Performance analysis of intrusion detection systems using a feature selection method on the UNSW-NB15 dataset, *J. Big Data*, **7** (2020), 1–20. https://doi.org/10.1186/s40537-020-00379-6
33. A. A. Megantara, T. Ahmad, A hybrid machine learning method for increasing the performance of network intrusion detection systems, *J. Big Data*, **8** (2021), 1–19. https://doi.org/10.1186/s40537-021-00531-w
34. M. Rashid, J. Kamruzzaman, T. Imam, S. Wibowo, S. Gordon, A tree-based stacking ensemble technique with feature selection for network intrusion detection, *Appl. Intell.*, **52** (2022), 1–14. https://doi.org/10.1007/s10489-021-02968-1

35. A. Chohra, P. Shirani, E. B. Karbab, M. Debbabi, Chameleon: Optimized feature selection using particle swarm optimization and ensemble methods for network anomaly detection, *Comput. Secur.*, **117** (2022), 102684. https://doi.org/10.1016/j.cose.2022.102684
36. B. Y. Tama, M. Comuzzi, K. H. Rhee, TSE-IDS: A two-stage classifier ensemble for intelligent anomaly-based intrusion detection system, *IEEE Access*, **7** (2019), 94497–94507. https://doi.org/10.1109/ACCESS.2019.2928048
37. B. I. Seraphim, E. Poovammal, K. Ramana, N. Kryvinska, N. Penchalaiah, A hybrid network intrusion detection using darwinian particle swarm optimization and stacked autoencoder hoeffding tree, *Math. Biosci. Eng.*, **18** (2021), 8024–8044. https://doi.org/10.3934/mbe.2021398
38. S. Seo, S. Park, J. Kim, Improvement of network intrusion detection accuracy by using restricted Boltzmann machine, in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN*), (2016), 413–417. https://doi.org/10.1109/CICN.2016.87
39. W. Li, G. Yin, X. Chen, Application of deep extreme learning machine in network intrusion detection systems, *IAENG Int. J. Comput. Sci.*, **47** (2020), 136–143.
40. Z. R. Zhao, L. N. Ge, G. F. Zhang, A novel DBN-LSSVM ensemble method for intrusion detection system, in *2021 9th International Conference on Communications and Broadband Networking*, (2021), 101–107. https://doi.org/10.1145/3456415.3456431
41. H. Zhang, L. N. Ge, Z. Wang, A high performance intrusion detection system using LightGBM based on oversampling and undersampling, in *International Conference on Intelligent Computing*, **13393** (2022), 638–652. https://doi.org/10.1007/978-3-031-13870-6_53
42. G. L. Ke, Q. Meng, T. Finley, T. F. Wang, W. Cheng, W. D. Ma, et al., Lightgbm: A highly efficient gradient boosting decision tree, *Adv. Neural Inf. Process. Syst.*, **30** (2017).
43. T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in *Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining*, (2016), 785–794. https://doi.org/10.1145/2939672.2939785
44. K. Mo, J. Li, A deep auto-encoder based LightGBM approach for network intrusion detection system, in *Proceedings of the International Conference on Advances in Computer Technology, Information Science and Communications*, (2019), 142–147. http://doi.org/10.5220/0008098401420147
45. T. Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollar, Focal loss for dense object detection, in *Proceedings of the IEEE International Conference on Computer Vision*, (2017), 2980–2988.
46. Q. Liu, D. Wang, Y. Jia, S. Luo, C. Wang, A multi-task based deep learning approach for intrusion detection, *Knowl. Based Syst.*, **238** (2022), 1–12. https://doi.org/10.1016/j.knosys.2021.107852
47. N. Shone, T. N. Ngoc, V. D. Phai, Q. Shi, A deep learning approach to network intrusion detection, *IEEE Trans. Emerging Top. Comput. Intell.*, **2** (2018), 41–50. https://doi.org/10.1109/TETCI.2017.2772792
48. S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, et al., Enhanced network anomaly detection based on deep neural networks, *IEEE Access*, **6** (2018), 48231–48246. https://doi.org/10.1109/ACCESS.2018.2863036
49. M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, (2009), 1–6. https://doi.org/10.1109/CISDA.2009.5356528

50. N. Moustafa, J. Slay, UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), in *2015 Military Communications and Information Systems Conference (MilCIS)*, (2015), 1–6. https://doi.org/10.1109/MilCIS.2015.7348942
51. N. Moustafa, J. Slay, The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set, *Inf. Secur. J. Global Perspect.*, **25** (2016), 18–31. http://dx.doi.org/10.1080/19393555.2015.1125974
52. W. J. Lian, G. Q. Nie, B. Jia, D. D. Shi, Q. Fan, Y. Q. Liang, An intrusion detection method based on decision tree-recursive feature elimination in ensemble learning, *Math. Prob. Eng.*, **2020** (2020). https://doi.org/10.1155/2020/2835023
53. *LightGBM*. Available from: https://lightgbm.readthedocs.io/
54. N. Moustafa, J. Slay, G. Creech, Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks, *IEEE Trans. Big Data*, **5** (2017), 481–494. https://doi.org/10.1109/TBDATA.2017.2715166
55. B. A. Tama, K. H. Rhee, An in-depth experimental study of anomaly detection using gradient boostedmachine, *Neural Comput. Appl.*, **31** (2019), 955–965. https://doi.org/10.1007/s00521-017-3128-z

AIMS Press