The Journal of Supercomputing (2019) 75:5597–5621
https://doi.org/10.1007/s11227-019-02805-w

Unsupervised learning approach for network intrusion detection system using autoencoders

Hyunseung Choi[1] · Mintae Kim[1] · Gyubok Lee[1] · Wooju Kim[1]

– We can use CIC 2017 data

Published online: 9 March 2019

Abstract

Network intrusion detection systems are useful tools that support system administrators in detecting various types of intrusions and play an important role in monitoring and analyzing network traffic. In particular, anomaly detection-based network intrusion detection systems are widely used and are mainly implemented in two ways: (1) a supervised learning approach trained using labeled data and (2) an unsupervised learning approach trained using unlabeled data. Most studies related to intrusion detection systems focus on supervised learning. However, the process of acquiring labeled data is expensive, requiring manual labeling by network experts. Therefore, it is worthwhile investigating the development of unsupervised learning approaches for intrusion detection systems. In this study, we developed a network intrusion detection system using an unsupervised learning algorithm autoencoder and verified its performance. As our results show, our model achieved an accuracy of 91.70%, which outperforms previous studies that achieved 80% accuracy using cluster analysis algorithms. Our results provide a practical guideline for developing network intrusion detection systems based on autoencoders and significantly contribute to the exploration of unsupervised learning techniques for various network intrusion detection systems.

Keywords Intrusion detection system · Unsupervised learning · Autoencoder · Anomaly detection · NSL-KDD

✉ Wooju Kim
wkim@yonsei.ac.kr

Hyunseung Choi
hyunseungchoi@yonsei.ac.kr

Mintae Kim
iammt@yonsei.ac.kr

Gyubok Lee
glee48@yonsei.ac.kr

1 Department of Industrial Engineering, Yonsei University, 50 Yonsei-ro, Seodaemun-gu, Seoul, Republic of Korea

1 Introduction

In security research, network intrusion is considered a high-tech crime [1] and is a very important research issue, together with automated surveillance systems [2] and information encryption studies [3]. A network intrusion detection system (NIDS) is an automated process for monitoring network events to identify security issues occurring on a computer system or network. As an important tool to help system administrators detect various types of network intrusion attacks, NIDSs have been widely utilized to detect and analyze network traffic and to perform alarm functions for intrusions [4, 5].

NIDSs are divided into two major categories: (1) signature-based NIDSs (SNIDSs) and (2) anomaly detection-based NIDSs (ADNIDSs) [4]. A SNIDS uses predefined attack rules to determine if an intrusion is detected (e.g., Snort[1]) when a specific traffic pattern is found. Although a SNIDS is an effective method for detecting known attacks, there are limitations on detecting unknown or new types of attacks. On the other hand, an ADNIDS classifies an intrusion when the network traffic deviates from the normal traffic pattern. Unlike SNIDSs, ADNIDSs have been widely used in various types of NIDS studies, because they are suitable for detecting unknown or new types of attacks.

There are two types of ADNIDSs: supervised learning-based NIDSs and unsupervised learning-based NIDSs [6]. A supervised learning-based NIDS refers to a model that learns to predict attacks using attack and non-attack labeled data. On the other hand, an unsupervised learning-based NIDS refers to a model that learns using input data that have no attack label. By using labeled data, a supervised learning-based NIDS has a great advantage in that it can effectively learn a high-performance model compared to unsupervised learning-based models. Therefore, most studies have focused on supervised learning-based NIDSs for intrusion classification.

However, there are many difficulties in the development of a supervised learning-based NIDS. The main problem is that acquiring attack labeled data is expensive due to the fact that network professionals must scrutinize the traffic data and decide whether a particular pattern is an attack or not. Therefore, there is an increasing need to study unsupervised learning-based NIDSs that can learn the network model without having to use labeled data.

This paper confirms the practicality and potential of a popular unsupervised learning framework autoencoder for NIDS. In Sect. 2 of this paper, the theoretical background of different types of autoencoders and previous NIDS studies are discussed. In Sect. 3, our research methodology is described. In Sect. 4, the performance of our proposed model is evaluated and the results are discussed. Finally, in Sect. 5, the implications and limitations of the research and future research directions are discussed.

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

2 Theoretical background and literature review

2.1 Literature review on network intrusion detection systems

In this paper, we did a model performance comparative analysis of NIDS studies using the NSL-KDD data set. The studies we referenced are summarized in Table 1. We divided these studies into three main types: (1) NIDS studies based on supervised learning techniques with explicit labels [7–9], (2) NIDS studies based on unsupervised learning techniques using only the input data without labels [10], and (3) NIDS studies based on supervised learning and unsupervised learning techniques together which classify normal and abnormal data using supervised learning techniques after designing derived variables using unsupervised learning techniques [4, 11].

Most of the NIDS studies are based on supervised learning. Panda et al. [7] achieved a 96.5% accuracy using a two-class classification (normal/abnormal) with a naïve Bayes classifier. Naoum et al. [8] conducted a five-class classification using an artificial neural network to distinguish abnormal data attack types and showed a 94.7% accuracy. In Thaseen and Kumar's study [9], a 97.49% accuracy using decision tree-based classification techniques was achieved.

NIDS development based on unsupervised learning has not been frequently studied, presumably due to its low performance. In the study by Syarif et al. [10], a NIDS based on cluster analysis algorithms was developed. They compared the performance of various unsupervised learning NIDS algorithms, such as k-means, k-medoids, expectation–maximization (EM) clustering, and distance-based outlier detection. The results showed that the performance of the distance-based outlier detection model was the best with 80.15% accuracy. The accuracy of other algorithms ranged from 57.81% to 78.06%. These are relatively disappointing results considering that models based on supervised learning algorithms showed high performance with about 90% accuracy. Syarif et al. [10] developed a NIDS that can detect anomalies without labeled data using unsupervised clustering algorithms. They hypothesized that clusters having many instances are normal and the ones having relatively few instances are abnormal. One of their work's limitations, however, is that their approach does not consider how to set up a threshold value to distinguish between normal and abnormal. If the threshold is not appropriate, it may not be possible to perform effective anomaly detection, since it is up to the researcher how many instances in a cluster should be normal or abnormal. In addition, cluster analysis may not be able to distinguish normal and abnormal data effectively, since its result can vary based on the initialization of clusters.

Despite the fact that only using unsupervised learning approaches for NIDS development has not been explored much, other works used unsupervised learning methodologies to design the derived variables in supervised learning-based NIDSs. Heba et al. [11] improved the time complexity by preprocessing the input variables using principal component analysis (PCA) unsupervised learning algorithm to reduce dimension before training the support vector machine classifier in

Table 1 Literature review on network intrusion detection system

Researcher	Methodology	Abstract
Panda et al. [7]	Supervised learning	Multinomial naïve Bayes-based NIDS
Naoum et al. [8]	Supervised learning	Artificial neural network-based NIDS
Thaseen and Kumar [9]	Supervised learning	Decision tree-based NIDS
Syarif et al. [10]	Unsupervised learning	Clustering-based NIDS
Javaid et al.[4]	Supervised learning and unsupervised learning	Development of the NIDS using autoencoder and logistic regression
Heba et al. [11]	Supervised learning and unsupervised learning	Development of the NIDS using principal component analysis and support vector machine

their NIDS. Javaid et al. [4] demonstrated the performance improvement by using an autoencoder to extract new features before applying logistic regression.

In summary, most NIDS studies have used labeled data to classify data through supervised learning algorithms. Unsupervised learning algorithms applied in NIDS development are mainly used to add richer representations of the original data by extracting newly derived variables before classification (supervised learning). When only using unsupervised learning algorithms in NIDS development, the results have shown lower performance than that of supervised learning approaches, which consequently has led to less frequent research studies.

Nonetheless, the process of acquiring labeled data for NIDSs has many practical difficulties, mainly because it is expensive for network professionals to investigate whether data are normal or abnormal. Therefore, there is a need for more extensive investigations into using unsupervised learning approaches in NIDS development. In this study, we developed a solely unsupervised learning method for NIDS development using an unsupervised artificial neural network autoencoder.

2.2 Autoencoders

As an artificial neural network algorithm based on unsupervised learning, autoencoder aims to reconstruct its input vectors [12]. Ever since it was invented, autoencoders have been used in a variety of applications including anomaly detection. Sakurada and Yairi [13] studied a methodology utilizing autoencoders to perform anomaly detection based on thresholds of reconstruction errors. In the training phase, the autoencoder model learns to reconstruct its input data consisting of only normal data. In the test phase, test data are input to the learned model to output reconstructed test data. When the reconstruction error is higher than a certain threshold value arbitrarily chosen by the user, the data are determined to be abnormal, and vice versa. The underlying assumption of this approach is that test data that are similar to training data will be relatively reconstructed well and therefore have low reconstruction errors. On the other hand, test data that are different from training data will have higher reconstruction errors because they are different from what the model was trained to reconstruct.

Mirsky et al. [14] applied an ensemble strategy using various autoencoder algorithms to the NIDS domain based on the approaches by Sakurada and Yairi [13]. The study done by Mirsky et al. [14] is meaningful in that it applies the anomaly detection methodology using autoencoders to the NIDS domain. However, there are two aspects of limitations. First, it assumes that the data used in autoencoder training must be all normal. To construct a high-quality data set, it is necessary to first gather only normal data, but this is difficult in practice due to manual labeling by network experts. Thus, it is necessary to address this issue of training a model on data that have far more normal than abnormal patterns. Second, to use this unsupervised learning approach, a certain value of threshold must be set. The reconstruction loss threshold is critical in that this very value is the criterion that distinguishes between normal and abnormal. If the threshold is not properly set, the performance

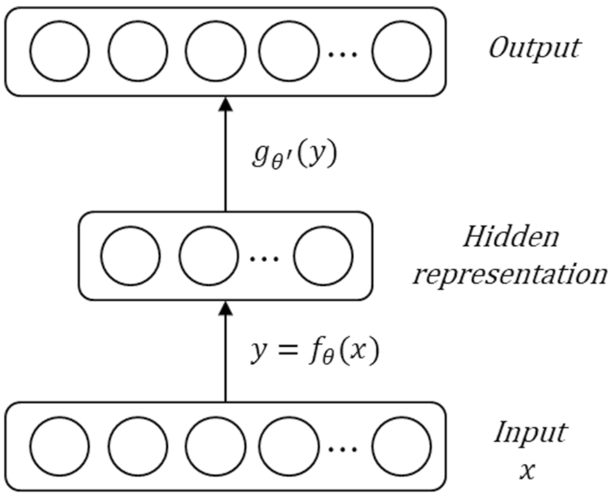

Fig. 1 Basic autoencoder

of the unsupervised model is not guaranteed. There needs to be research done on systematical approaches to investigate how to set threshold values.

In this study, we have developed a NIDS based on unsupervised learning approach using autoencoder and suggest a threshold heuristic of reconstruction error—the proportion of abnormality in training data. Our approach is distinguished from other NIDS studies in that we use training data that reflect the real network environment that has varying degrees of abnormality in data.

2.2.1 Notation

In this study, we follow the notation used in Vincent et al. [15] for formulating autoencoders. For random variables X, Y, the joint probability density of X, Y is denoted by $p(X, Y)$, and the marginal distribution is denoted by $p(X), p(Y)$. Based on this, we use the following notation. Expectation of $f(x)$ is denoted by $\mathbb{E}_{p(X)}\left[f(X)\right] = \int p(x)f(x)dx$ where $f(x)$ is a function of X. Entropy of X is denoted by $\mathbb{H}(X) = \mathbb{H}(p) = \mathbb{E}_{p(X)}\left[-\log p(X|Y)\right]$. Cross-entropy is denoted by $\mathbb{H}(p \parallel q) = \mathbb{E}_{p(X)}\left[-\log p(X)\right] = \mathbb{H}(p) + \mathbb{D}_{KL}(p \parallel q)$. Sigmoid is denoted by $s(x) = \frac{1}{1+e^{-x}}$, $s(x) = \left(s\left(x_1\right), \ldots, s\left(x_d\right)\right)^T$. N pairs (input, target) of training data set are denoted by $D_n = \left\{\left(x^{(1)}, t^{(1)}\right), \ldots, \left(x^{(n)}, t^{(n)}\right)\right\}$. Extracted distribution from unknown marginal distributions is denoted by $q(X), q(T)$: $q(X, T)$.

2.2.2 Basic autoencoder and stacked autoencoder

With one hidden layer, basic autoencoder aims to learn a model to extract the most representative data features by reconstructing the original input data under specified constraints given in the middle of the reconstruction process [12]. Figure 1 illustrates the structure of basic autoencoder. The reconstruction process consists of two stages [15]. First, during the encoding stage, a basic autoencoder maps the input x to a hidden representation $y = f_\theta(x) = s(Wx + b)$ in a parameterizing fashion using a set of encoding parameters $\theta = \{W, b\}$. $W$and b refer to weight and bias, respectively. Second, after the encoding stage, the hidden representation y is then mapped to the "reconstructed" vector $z = g_{\theta'}(y) = s\left(W'y + b'\right)$ using decoding parameters $\theta' = \left\{W', b'\right\}$ [15]. In sum, each training datum $x^{(i)}$ is mapped to the corresponding $y^{(i)}$, and $y^{(i)}$ is again mapped to its reconstructed vector $z^{(i)}$. The parameters of a basic autoencoder model are optimized in such a way that it minimizes the *average reconstruction error* [Formula (1)] .

$$\begin{aligned}\theta^*, \theta'^* &= \underset{\theta,\theta'}{arg\ min}\, \frac{1}{n} \sum_{i=1}^{n} L\left(x^{(i)}, z^{(i)}\right) \\ &= \underset{\theta,\theta'}{arg\ min}\, \frac{1}{n} \sum_{i=1}^{n} L\left(x^{(i)}, g_{\theta'}\left(f_{\theta\left(x^{(i)}\right)}\right)\right)\end{aligned} \tag{1}$$

In Formula (1), L is a traditional loss function, such as $L1$ ($L(x, z) = |x - z|$) or $L2$ $\left(L(x, z) = |x - z|^2\right)$ loss. In addition, the loss function can be replaced by the *reconstruction cross-entropy* [Formula (2)] considering x, z as a bit vector or a bit probability (Bernoulli), where x is a binary vector and $L_{\mathbb{H}}(x, z)$ is the negative log-likelihood for x given the Bernoulli parameter z.

$$\begin{aligned}L_{\mathbb{H}}(x, z) &= \mathbb{H}\left(\mathbb{B}_{\boldsymbol{x}} \parallel \mathbb{B}_z\right) \\ &= -\sum_{k=1}^{d} \left[x_k \log z_k + \left(1 - x_k\right) \log\left(1 - z_k\right)\right]\end{aligned} \tag{2}$$

For $L = L_{\mathbb{H}}$, Formula (1) can be represented as Formula (2).

Stacked autoencoder [12, 16–18] refers to an autoencoder algorithm consisting of multiple hidden layers, in contrast to basic autoencoder which has a single hidden layer. Figure 2 illustrates the structure of stacked autoencoder. Just as basic autoencoder, stacked autoencoder is divided into two steps: encoding and decoding. During the encoding step, the input variable x is mapped to the hidden representation $y = f_\theta(x) = f_\theta(Wx + b)$, and during the decoding step, the hidden representation y is converted to the reconstructed input vector $z = g_{\theta'}(y) = s\left(W'y + b'\right)$. At this time, several hidden layers are stacked together to form an artificial neural network and the hidden representation y calculated in the previous hidden layer is used as an input variable for the next hidden layer. Each layer of the stacked autoencoder is trained using optimization algorithms such as a stochastic gradient descent [19].

2.2.3 Denoising autoencoder

The basic idea of a denoising autoencoder is to train an autoencoder algorithm to produce a robust input feature representation by reconstructing the original input after partially destroying the original model input [15]. Figure 3 illustrates the structure of denoising autoencoder. A denoising autoencoder learns through the

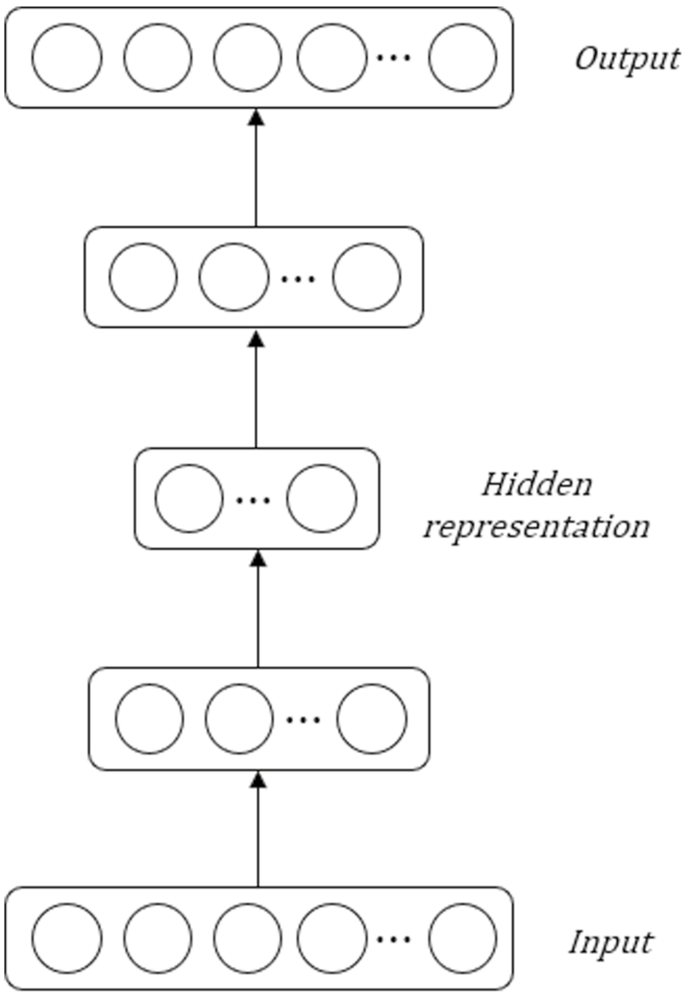

Fig. 2 Stacked autoencoder

following process. Through stochastic mapping $\tilde{x} \sim q_D(\tilde{x}|x)$, the initial input vector x is destroyed to produce a partially destroyed input $\tilde{x}$. Using d dimensions of x and a ratio parameter v, the corrupting process works in such a way that the fixed vd number of components of the input x is randomly selected and replaced with zeros. In other words, the selected number of components is removed from the input pattern and the autoencoder is trained to "fill in" the artificially introduced "blanks" [15]. This corruption process can also be replaced by introducing Gaussian noises instead. The destroyed input $\tilde{x}$ is mapped to the hidden representation of the autoencoder, y $= f_\theta(\tilde{x}) = s(W\tilde{x} + b)$, and then mapped to reconstruct the original vector in the form of $z = g_{\theta'}(y) = s\left(W'y + b'\right)$. As in basic autoencoder, the parameters of a denoising autoencoder are trained to minimize the *average reconstruction*

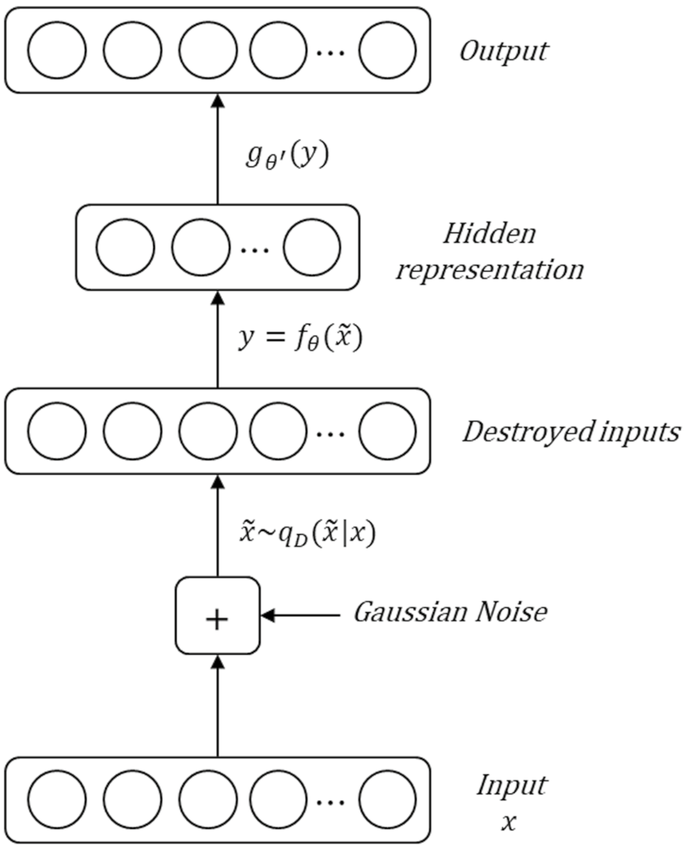

Fig. 3 Denoising autoencoder

error $L_{\mathbb{H}}(x, z) = \mathbb{H}\left(\mathbb{B}_x \parallel \mathbb{B}_z\right)$. The key difference from basic autoencoder is that z is a deterministic function of $\tilde{x}$ (not x), where $\tilde{x}$ is the output of the stochastic mapping from x. Formula (3) represents the objective function of a denoising autoencoder. X denotes the input and the output of the model, $\tilde{X}$ denotes the destroyed input X, and $q^0\left(X, \tilde{X}\right)$ denotes the empirical joint distribution of X and $\tilde{X}$. The objective function [Formula (3)] can be trained with the same optimizers used in other autoencoders.

$$\underset{\theta,\theta'}{arg\ min}\, \mathbb{E}_{q^0(X,\tilde{X})}\left[L_{\mathbb{H}}\left(X, g_{\theta'}\left(f_\theta\left(\tilde{X}\right)\right)\right)\right] \tag{3}$$

2.2.4 Variational autoencoder

Variational autoencoder [20] differs from other autoencoders in that the model forces the distribution of the hidden representation to follow a parameterized probability distribution. Figure 4 illustrates the structure of variational autoencoder. Basically, in variational autoencoders, the encoder generates a hidden representation just as in stacked autoencoders, and the decoder performs the

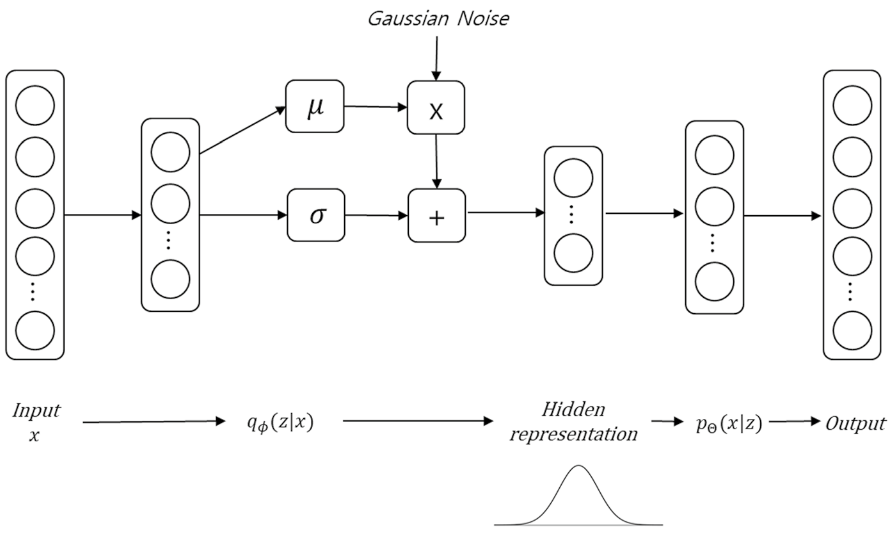

Fig. 4 Variational autoencoder

process of reconstructing input data based on the hidden representation. Unlike stacked autoencoders that generate hidden representations directly from the previous hidden layer, variational autoencoders randomly extract the mean μ and standard deviation σ of the previous hidden layers and use it to derive its hidden representation to follow a normal distribution. In the decoding stage, just as other autoencoders do, the hidden representation is used to reconstruct the input data.

Formula (4) represents the loss function for a variational autoencoder (θ, ϕ), where $x^{(i)}$ is the input variable, $z^{(i)}$ is the hidden representation, and $y^{(i)}$ is the output (or the reconstructed input). $q_\phi(z|x)$ refers to the probability distribution of the encoder, and $p_\theta(x|z)$ is the probability distribution of the decoder, where θ and ϕ are the parameters of the encoder and the decoder, respectively.

$$L(\theta, \phi) = -\mathbb{E}_{z \sim q_\phi(z|x^{(i)})}\left[\log p_\theta\left(x^{(i)}|z\right)\right] + \mathbb{D}_{KL}\left(q_\phi\left(z|x^{(i)}\right) \parallel p(z)\right) \tag{4}$$

In Formula (4), the first right term is the reconstruction loss of the model and the second right term is a Kullback–Leibler divergence term for fitting the distribution of the hidden representation to a desired probability distribution, generally a standard Gaussian distribution with a mean of 0 and a variance of 1. In short, a variational autoencoder is trained to best reconstruct its original input while keeping the distribution of its hidden representation to a parameterized probability distribution.

Table 2 Attack type expansion of NSL-KDD data set

No.	Attack name	Attack type	No.	Attack name	Attack type
1	Back	Dos	12	Perl	U2r
2	Buffer_overflow	U2r	13	Phf	R2l
3	Ftp_write	R2l	14	Pod	Dos
4	Huess_passwd	R2l	15	Portsweep	Probe
5	Imap	R2l	16	Rootkit	U2r
6	Ipsweep	Probe	17	Satan	Probe
7	Land	Dos	18	Smurf	Dos
8	Loadmodule	U2r	19	Spy	R2l
9	Multihop	R2l	20	Teardrop	Dos
10	Meptune	Dos	21	Warezclient	R2l
11	Nmap	Probe	22	Warezmaster	R2l

2.3 NSL-KDD data set

The traditional KDD Cup data set[2] was developed by the Defense Advanced Research Projects Agency (DARPA) and has been utilized as benchmark data to evaluate the performance of NIDS models. However, the KDD Cup data set has a problem in that NIDS models trained on the KDD Cup data set tend to have a bias toward frequent types of attacks. Due to the high redundancies in the data set, it is difficult for the model to detect low-frequency fatal attacks [4, 21]. Thus, to circumvent the limitations of the KDD Cup data set, the new NSL-KDD[3] data set was proposed.

The NSL-KDD data set resolves two issues: Duplicate data were removed from the KDD Cup data set, and the level of difficulty in anomaly detection was improved by categorizing the detailed attack types in the abnormal data. As a result, the NSL-KDD data set has been widely used in a number of studies [4, 7–11] as a benchmark data set in the development of NIDSs for real-world applications. Therefore, we utilized the NSL-KDD data set in our NIDS development.

Each record in the NSL-KDD data set states whether it is normal or abnormal, and each item of abnormal data is individually labeled with attack types such as Dos, U2r (user-to-root), R2l (remote-to-local). Moreover, the attack type is further divided into detailed attacks (e.g., Dos attack type are divided into detailed attacks Back, Land, Neptune, Pod, Smurf, and Teardrop). The forty-one features of the NSL-KDD data set include features extracted directly through TCP/IP connections, such as connection duration and protocol type as well as accumulated traffic characteristics in each interval. The forty-one features are composed of four binary, three nominal, and thirty-four numeric variables. "Appendix" summarizes the detailed

[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[3] http://www.unb.ca/cic/datasets/nsl.html.

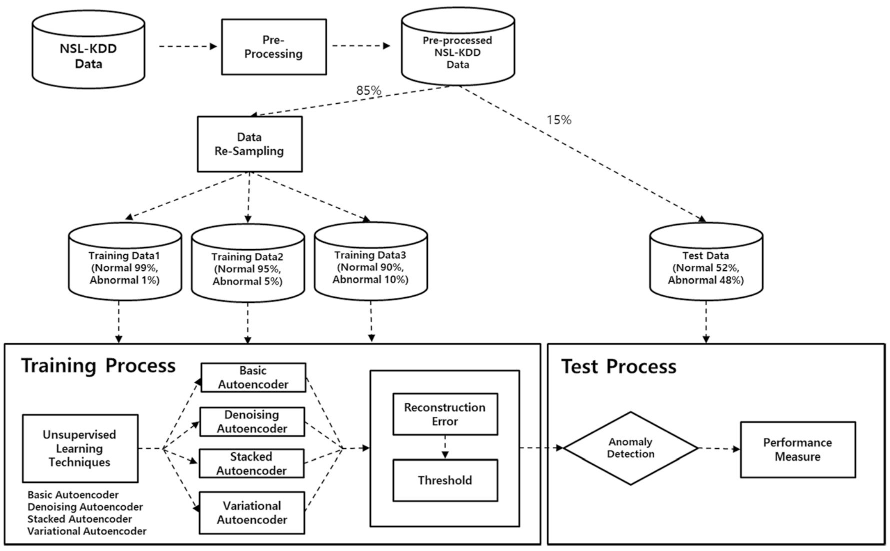

Fig. 5 Research method

explanation of each feature in the NSL-KDD data set. More information about the abnormal data labels is listed in Table 2.

3 Research methodology

The main purpose of this study is to develop a NIDS using an autoencoder. Our research methodology is divided into the data preparation process and the data analysis process. A detailed schematic diagram of the process is summarized in Fig. 5. The data preparation process is comprised of the data preprocessing stage and the training and test data set partitioning stage. The data analysis process involves training our autoencoder model, devising reconstruction error thresholds, and measuring the performance of our model using test data.

3.1 Data preparation process

First, data preprocessing was performed on the NSL-KDD data set. Numerical data were normalized using min–max normalization, and nominal data were converted into one-hot vectors, representing categorical features as binary vectors, before inputting into our autoencoder models.

In the data partitioning process, 85% of the data set was used for training and the remaining 15% was used for testing. To cope with various situations and patterns in the real network data, we further divided the training data sets into three subsets that differed in label ratios: (1) training data 1: 99% normal and 1% abnormal, (2) training data 2: 95% normal and 5% abnormal, and (3) training data 3: 90% normal and 10% abnormal. The data resampling process was performed using resampling operations so that the proportion of abnormal data was exactly 1%, 5%, and 10%, respectively. Through this, we constructed three different training data sets reflecting the real network environment where most of the data were normal, and there was only a small percentage of abnormal data.

3.2 Data analysis process

In the training process, each three training data set was used to train the four different unsupervised learning models: basic autoencoder, denoising autoencoder, stacked autoencoder and variational autoencoder, resulting in twelve cases (3 training data sets times 4 models). With this approach, all models were able to produce dimensionally reduced key features for each of the three training data sets with different levels of abnormality, which later could be used to reconstruct the original data. We then used the value of the reconstruction error θ_α that corresponds to the upper α% of the reconstruction error distribution to devise a threshold that classified the data sets with α% abnormality.

In the testing process, the input data including both normal and abnormal labels were reconstructed, and the difference between the input data and the reconstructed data was measured. If the reconstruction error was greater than or equal to the value of threshold θ_α, then the corresponding data were classified as abnormal data; otherwise, it was normal. After classifying the normal and abnormal data, we evaluated the final performance of the model.

4 Results

4.1 Data preparation result

Table 3 summarizes the frequency and ratio of the normal and abnormal classes for the training data and test data constructed through data preparation. In this study, the training data set and test data set were constructed using the preprocessed NSL-KDD data set with 85% used for the training data set and 15% used for the test data set. In addition, the training data set was resampled so that the proportion of abnormal data was 1% (training data 1), 5% (training data 2), and 10% (training data 3) to reflect real network conditions, where most of the data are normal and only a small proportion of the data is abnormal.

As a result, all training data sets had 65,444 normal data points, and training data 1, training data 2, and training data 3 data sets had 661, 3,444, and 7,271 abnormal data points, respectively. The test data set consists of 11,609 normal and 10,669 abnormal data points, which was almost class-balanced.

4.2 Data analysis results

In general, hyperparameter and parameter tuning are an essential step to yield the best performance out of a model. However, since our approach was totally

Table 3 Data set description

Data set	Statistics	Total records	Normal class	Dos class	Probe class	U2r class	R2l class	Etc.
Training data 1	Count	66,105	65,444	490	128	3	37	3
(normal 99%; abnormal 1%)	Ratio	100.00%	99.00%	0.74%	0.19%	0.00%	0.06%	0.00%
Training data 2	Count	68,888	65,444	2,564	663	5	193	19
(normal 95%; abnormal 5%)	Ratio	100.00%	95.00%	3.72%	0.96%	0.00%	0.28%	0.03%
Training data 3	Count	72,715	65,444	5,436	1,390	9	402	34
(normal 90%; abnormal 10%)	Ratio	100.00%	90.00%	7.48%	1.91%	0.01%	0.55%	0.05%
Test data	Count	22,278	11,609	7,955	2,079	20	568	47
	Ratio	100.00%	52.11%	35.71%	9.33%	0.90%	2.54%	0.21%

Table 4 Model's threshold (training data 1: normal 99%; abnormal 1%)

Metric	Basic autoencoder	Denoising autoencoder	Stacked autoencoder	Variational autoencoder
Mean of reconstruction error μ_1	0.7945	0.5663	1.4071	1.0116
Standard deviation of reconstruction error σ_1	0.4343	0.3036	0.4089	0.5668
$\theta_1 = \mu_1 + 2.33 * \sigma_1$	1.8063	1.2737	2.3599	2.3321

Table 5 Model's threshold (training data 2: normal 95%; abnormal 5%)

Metric	Basic autoencoder	Denoising autoencoder	Stacked autoencoder	Variational autoencoder
Mean of reconstruction error μ_5	0.7456	0.5226	1.3659	1.0823
Standard deviation of reconstruction error σ_5	0.4441	0.2910	0.4917	0.6070
$\theta_5 = \mu_5 + 1.645 * \sigma_5$	1.4761	1.0014	2.1747	2.0809

Table 6 Model's threshold (training data 3: normal 90%; abnormal 10%)

Metric	Basic autoencoder	Denoising autoencoder	Stacked autoencoder	Variational autoencoder
Mean of reconstruction error μ_{10}	0.7948	0.5105	1.5533	1.1016
Standard deviation of reconstruction error σ_{10}	0.4257	0.4223	0.4223	0.6267
$\theta_{10} = \mu_{10} + 1.28 * \sigma_{10}$	1.3396	2.0939	2.0939	1.9038

unsupervised to show the potential usage of the unsupervised method, we simply followed the hyperparameter and parameter settings from previous research and kept the rest as naïve as possible. Our models were implemented using Google TensorFlow. The basic autoencoder and denoising autoencoder models consisted of a single hidden layer with 32 units with the rectified linear unit (ReLU) activation function. The stacked autoencoder and variational autoencoder models consisted of three hidden layers with 32, 16, and 32 units with the ReLU activation function. We used the Adam optimizer with a 0.0001 learning rate [22], L2 regularization with 0.01, train epochs of 10, and the batch size of 256 for the parameters without excessive parameter tuning. Specifically, the same noise parameter of 0.1 was used in our denoising autoencoder model, as used by Sakurada and Yairi's study [13].

After training the models, we calculated the value of the threshold $\theta_\alpha = \mu_\alpha + Z_\alpha * \sigma_\alpha$ for the three training data sets to classify normal and abnormal data points during the test phase (μ_α means the average value of the reconstruction error, Z_α means a Z value for the upper α% of the standard normal distribution, and σ_α means the standard deviation value of the reconstruction error, in all training data

Table 7 Model's test performance (training data 1: normal 99%; abnormal 1%)

Metric	Basic autoencoder	Denoising autoencoder	Stacked autoencoder	Variational autoencoder
Accuracy	0.9170	0.8802	0.8782	0.8766
F1_score	0.9071	0.8613	0.8606	0.8559
Specificity	0.9815	0.9754	0.9642	0.9784
Precision	0.9768	0.9666	0.9527	0.9702
Recall	0.8468	0.7766	0.7847	0.7658

Table 8 Model's test performance (training data 2: normal 95%; abnormal 5%)

Metric	Basic autoencoder	Denoising autoencoder	Stacked autoencoder	Variational autoencoder
Accuracy	0.6971	0.6845	0.9012	0.8468
F1_score	0.5771	0.5573	0.8905	0.8599
Specificity	0.9412	0.9324	0.9577	0.9571
Precision	0.8708	0.8493	0.9480	0.9442
Recall	0.4315	0.4148	0.8396	0.7895

Table 9 Model's test performance (training data 3: normal 90%; abnormal 10%)

Metric	Basic autoencoder	Denoising autoencoder	Stacked autoencoder	Variational autoencoder
Accuracy	0.6220	0.6689	0.9038	0.8626
F1_score	0.4557	0.5383	0.8947	0.8446
Specificity	0.8901	0.9131	0.9493	0.9393
Precision	0.7342	0.8100	0.9394	0.9218
Recall	0.3304	0.4031	0.8542	0.7793

sets). Calculated thresholds for each three training data sets are presented in Table 4 (for training data 1), Table 5 (for training data 2), and Table 6 (for training data 3).

With the threshold calculated in the training process, we then used our test data set to evaluate the reconstruction errors for 12 different cases (3 training data sets times 4 models). The results are presented in Table 7 (for training data 1), Table 8 (for training data 2), and Table 9 (for training data 3). The performance of the basic autoencoder, denoising autoencoder, stacked autoencoder, and variational autoencoder models trained on 99% normal and 1% abnormal data showed 91.70%, 88.02%, 87.82%, and 87.66% accuracy, respectively (Table 7). In addition, the performance of the models trained using training data 2 (normal 95%, abnormal 5%) showed 69.71%, 68.45%, 90.12%, and 84.68% accuracy, respectively (Table 8). Finally, the performance of the models trained using training data 3 (normal 90%, abnormal 10%) showed 62.20%, 66.89%, 90.38%, and 86.26% accuracy, respectively (Table 9). Considering the performance of previous study using unsupervised

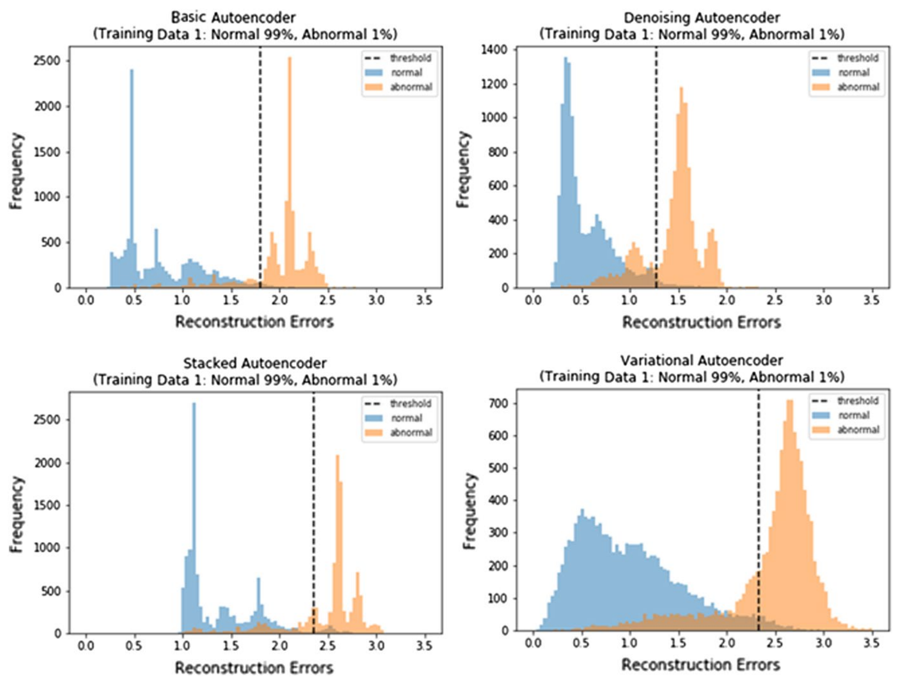

Fig. 6 Visualization of the results of the models trained using training data 1 (normal 99%; abnormal 1%): The value (x-axis) and the frequency (y-axis) of the reconstruction errors are plotted

learning-based NIDSs, which ranged from 57.81 to 80.15% accuracy [10], our proposed unsupervised learning-based models showed satisfactory performance, with the best accuracy of 91.70%.

The visualizations of the models evaluated on the test data are presented in Fig. 6 (training data 1), Fig. 7 (training data 2), and Fig. 8 (training data 3). The x-axis represents the amount of reconstruction error, and the y-axis represents the frequency of the corresponding reconstruction error. The normal and abnormal data points are colored in blue and yellow, respectively, and thresholds θ_α for classifying normal and abnormal data are drawn in vertical dotted lines. In this way, it is easier to visualize the difference between the reconstruction error distribution for both normal and abnormal data at once. In addition, it aids in checking proper values for the threshold.

Overall, the visualization showed that our proposed models successfully performed anomaly detection, revealing that the reconstruction error generated from normal data points was low and the error generated from abnormal data was high. As shown in Table 7, the four models trained using training data 1 (Fig. 6) properly performed anomaly detection based on the suggested threshold and distributed the normal and abnormal reconstruction errors in a separable way. In addition, as shown in Table 8, among the models trained using training data 2 (Fig. 7), the stacked autoencoder and variational autoencoder showed high performance based on the suggested threshold and the separability of the normal and abnormal

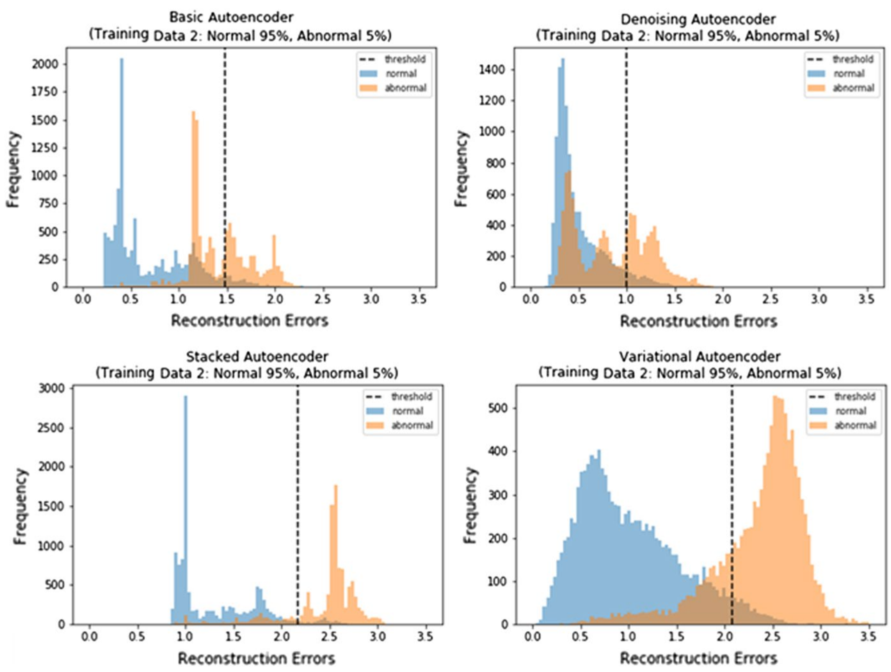

Fig. 7 Visualization of the results of the models trained using training data 2 (normal 95%; abnormal 5%): The value (*x*-axis) and the frequency (*y*-axis) of the reconstruction errors are plotted

data distribution. Finally, as shown in Table 9, among the models trained using training data 3 (Fig. 7), the stacked autoencoder and variational autoencoder had the highest performance.

The anomaly detection results of the basic autoencoder and denoising autoencoder show low performances as the proportion of abnormality data of training data increase. However, the stacked autoencoder and variational autoencoder properly performed anomaly detection and showed robust performance against the changes in the percentage of abnormality in training data. We argue that the reason why the basic autoencoder and the denoising autoencoder show lower performances on training data 2 and 3 over training data 1 is that as abnormal data increase, the data being generated become more heterogeneous, causing the reconstruction process more difficult.

On the other hand, the stacked autoencoder and the variational autoencoder show robust performances on all training data 1, 2, and 3. We argue that it is due to higher model complexity that the models can successfully capture more heterogeneous patterns, rather than simply averaging out between normal and abnormal patterns. As a result, we have found that, under the real-world situation where mixed normal and abnormal data coexist, models with higher complexity such as stacked autoencoders and variational autoencoders perform better than

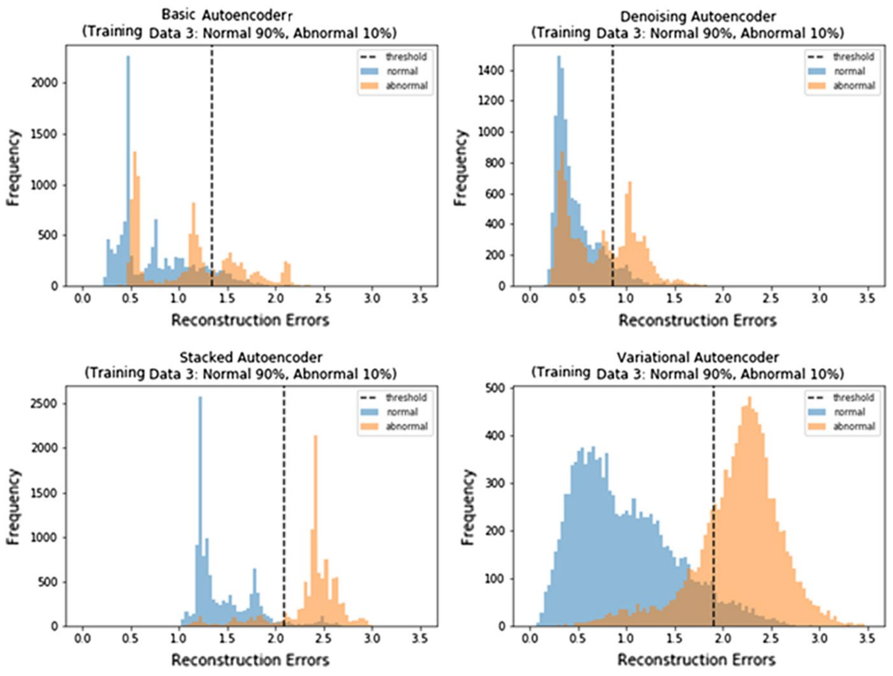

Fig. 8 Visualization of the results of the models trained using training data 3 (normal 90%; abnormal 10%): The value (x-axis) and the frequency (y-axis) of the reconstruction errors are plotted

basic autoencoders and denoising autoencoders for network intrusion detection scenarios.

5 Conclusion

While previous studies related to the development of NIDSs have mainly focused on the classification of attack types based on supervised learning algorithms, we have developed a NIDS based on unsupervised learning algorithm autoencoders capable of learning models without labeled data. First, we constructed three training data sets reflecting real-world network conditions in which most of the data are normal. Then, by using autoencoder algorithms, we trained our models to extract the core features from the input data. Based on the core features, we reconstructed the original input data to classify normal and abnormal data using the reconstruction error threshold. For training data that have $\alpha\%$ of abnormality, we set the value of reconstruction error θ_α as a threshold heuristic. This threshold heuristic is used to cut off the upper $\alpha\%$ of an assumed normal distribution of reconstructed errors to classify as abnormal data. As a result, we confirmed that the performance accuracy was higher than previous unsupervised learning-based NIDS studies using clustering algorithms [10].

The contribution of our study is to apply autoencoder-based anomaly detection algorithms to the domain of unsupervised learning-based network intrusion detection and to suggest a heuristic method of setting a reconstruction loss threshold based on the percentage of abnormal data in training data. Specifically, we utilize the percentage of abnormal data $\alpha\%$ in training data which later become a threshold heuristic to cut off the upper $\alpha\%$ of an assumed normal distribution of reconstructed errors to be classified as abnormal data. With our approach, one could easily detect abnormality without actual labels but only with a simple threshold from the previously observed percentage of abnormal data. Our approach is distinguished from supervised learning-based network intrusion detection systems that require manual labeling by network experts. We believe that our study is a good example for network managers who try to build their network intrusion detection systems using unsupervised learning with only having prior knowledge on the percentage of abnormality.

The limitations of the study and future research directions are as follows. First, there is the problem of our data set. Since our models are only trained using the NSL-KDD data set, the validity of our models is confined to the NSL-KDD data set. To justify the generality of our models, it would be advisable to train and test our models on different data sets including other open data sets and actual industry network data. Second, there is the problem of the use of a single kind of unsupervised learning algorithm. In this study, we developed a NIDS only using an autoencoder. Therefore, the applicability of other unsupervised learning models has yet to be verified. In the future, we expect to be able to develop a NIDS based on various unsupervised learning algorithms and heuristic theory and further improve the performance using ensemble strategies.

Acknowledgements This research was supported by the project "Efficient Operation of LTE Device and IT infrastructure Assets Self-diagnosis Integrated Control System" of the Korea Evaluation Institute of Industrial Technology.

Appendix

See Table 10.

Table 10 Feature description of NSL-KDD data set

No.	Feature name	Description	Data type
1	Duration	Length of time duration of the connection	Numeric
2	Protocol_type	Protocol used in the connection	Nominal
3	Service	Destination network service used	Nominal
4	Flag	Status of the connection—normal or error	Nominal
5	Src_byte	Number of data bytes transferred from source to destination in single connection	Numeric
6	Dst_bytes	Number of data bytes transferred from destination to source in single connection	Numeric
7	Land	If source and destination IP addresses and port numbers are equal, then this variable takes value 1 else 0	Binary
8	Wrong_fragment	Total number of wrong fragments in this connection	Numeric
9	Urgent	Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated	Numeric
10	Hot	Number of "hot" indicators in the content such as entering a system directory, creating programs, and executing programs	Numeric
11	Num_failed_logins	Count of failed log-in attempts	Numeric
12	Logged_in	Log-in status: 1 if successfully logged in; 0 otherwise	Binary
13	Num_compromised	Number of "compromised" conditions	Numeric
14	Root_shell	1 if root shell is obtained; 0 otherwise	Numeric
15	Su_attempted	1 if "su root" command attempted or used; 0 otherwise	Numeric
16	Num_root	Number of "root" accesses or number of operations performed as a root in the connection	Numeric
17	Num_file_creations	Number of file creation operations in the connection	Numeric
18	Num_shells	Number of shell prompts	Numeric
19	Num_access_files	Number of operations on access control files	Numeric
20	Num_outbound_cmds	Number of outbound commands in an ftp session	Numeric
21	Is_hot_login	1 if the log-in belongs to the "hot" list, i.e., root or admin; else 0	Binary
22	Is_guest_login	1 if the log-in is a "guest" log-in; 0 otherwise	Binary
23	Count	Number of connections to the same destination host as the current connection in the past two 2 s	Numeric
24	Srv_count	Number of connections to the same service (port number) as the current connection in the past 2 s	Numeric

Table 10 (continued)

No.	Feature name	Description	Data type
25	Serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2, or s3, among the connections aggregated in count (23)	Numeric
26	Srv_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2, or s3, among the connections aggregated in srv_count (24)	Numeric
27	Rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)	Numeric
28	Srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)	Numeric
29	Same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in count (23)	Numeric
30	Diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in count (23)	Numeric
31	Srv_diff_host_rate	The percentage of connections that were to different destination machines among the connections aggregated in srv_count (24)	Numeric
32	Dst_host_count	Number of connections having the same destination host IP address	Numeric
33	Dst_host_srv_count	Number of connections having the same port number	Numeric
34	Dst_host_same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)	Numeric
35	Dst_host_diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)	Numeric
36	Dst_host_same_src_port_rate	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)	Numeric
37	Dst_host_srv_diff_host_rate	The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33)	Numeric
38	Dst_host_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2, or s3, among the connections aggregated in dst_host_count (32)	Numeric
39	Dst_host_srv_serror_rate	The percent of connections that have activated the flag (4) s0, s1, s2, or s3, among the connections aggregated in dst_host_srv_count (33)	Numeric

Table 10 (continued)

No.	Feature name	Description	Data type
40	Dst_host_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)	Numeric
41	Dst_host_srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)	Numeric

References

1. Akhgar B, Saathoff GB, Arabnia HR, Hill R, Staniforth A, Bayerl PS (2015) Application of big data for national security: a practitioner's guide to emerging technologies. Butterworth-Heinemann, Oxford, p 320. https://doi.org/10.1016/B978-0-12-801967-2.01002-8
2. Deligiannidis L, Arabnia HR (2015) Security surveillance applications utilizing parallel video-processing techniques in the spatial domain. In: Deligiannidis L, Arabnia HR (eds) Emerging trends in image processing, computer vision and pattern recognition. Morgan Kaufmann, Boston, pp 117–130. https://doi.org/10.1016/B978-0-12-802045-6.00008-9
3. Choche A, Arabnia H (2011) A methodology to conceal qr codes for security applications. In: Proceedings of the 10th International Conference on Information and Knowledge Engineering. pp 151–157
4. Javaid A, Niyaz Q, Sun W, Alam M (2016) 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), 2016. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp 21–26. https://doi.org/10.4108/eai.3-12-2015.2262516
5. Bace R, Mell P (2001) NIST special publication on intrusion detection systems. National Institute of Standards and Technology, Gaithersburg
6. Gogoi P, Borah B, Bhattacharyya DK (2010) Anomaly detection analysis of intrusion data using supervised & unsupervised approach. J Converg Inf Technol 5(1):95–110
7. Panda M, Abraham A, Patra MR (2010) Discriminative multinomial Naïve Bayes for network intrusion detection. In: 2010 Sixth International Conference on Information Assurance and Security. pp 5–10. https://doi.org/10.1109/ISIAS.2010.5604193
8. Naoum RS, Abid NA, Al-Sultani ZN (2012) An enhanced resilient backpropagation artificial neural network for intrusion detection system. Int J Comput Sci Netw Secur 12(3):11–16
9. Thaseen S, Kumar CA (2013) An analysis of supervised tree based classifiers for intrusion detection system. In: 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering. pp 294–299. https://doi.org/10.1109/ICPRIME.2013.6496489
10. Syarif I, Prugel-Bennett A, Wills G (2012) Unsupervised clustering approach for network anomaly detection. 2012 networked digital technologies. Springer, Berlin, pp 135–145
11. Heba FE, Darwish A, Hassanien AE, Abraham (2010) A principle components analysis and support vector machine based intrusion detection system. In: 2010 10th International Conference on Intelligent Systems Design and Applications. pp 363–367. https://doi.org/10.1109/ISDA.2010.5687239
12. Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layer-wise training of deep networks. In: Proceedings of the 19th International Conference on Neural Information Processing Systems, Canada. pp 153–160
13. Sakurada M, Yairi T (2014) Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis. ACM, pp 4–11. https://doi.org/10.1145/2689746.2689747
14. Mirsky Y, Doitshman T, Elovici Y, Shabtai A (2018) Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv Preprint. arXiv:1802.09089
15. Vincent P, Larochelle H, Bengio Y, Manzagol P-A (2008) Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland. ACM, pp 1096–1103. https://doi.org/10.1145/1390156.1390294
16. Bengio Y (2009) Learning Deep Architectures for AI. Found Trends Mach Learn 2(1):1–127. https://doi.org/10.1561/2200000006
17. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507. https://doi.org/10.1126/science.1127647
18. Cao L, Huang W, Sun F (2016) Building feature space of extreme learning machine with sparse denoising stacked-autoencoder. Neurocomput 174:60–71. https://doi.org/10.1016/j.neucom.2015.02.096
19. Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 22(3):400–407
20. Kingma DP, Welling M (2013) Auto-encoding variational bayes. arXiv Preprint. arXiv:1312.6114
21. Tavallaee M, Bagheri E, Lu W, Ghorbani AA (2009) A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications. pp 1–6. https://doi.org/10.1109/CISDA.2009.5356528

22. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv Preprint. arXiv:1412.6980