Cluster Computing (2024) 27:2365–2379
https://doi.org/10.1007/s10586-023-04089-5

Machine learning-based intrusion detection: feature selection versus feature extraction

Vu-Duc Ngo[1,2] · Tuan-Cuong Vuong[3] · Thien Van Luong[3] · Hung Tran[3]

Received: 2 December 2022 / Revised: 5 June 2023 / Accepted: 11 June 2023 / Published online: 5 July 2023

Abstract
Internet of Things (IoTs) has been playing an important role in many sectors, such as smart cities, smart agriculture, smart healthcare, and smart manufacturing. However, IoT devices are highly vulnerable to cyber-attacks, which may result in security breaches and data leakages. To effectively prevent these attacks, a variety of machine learning-based network intrusion detection methods for IoT networks have been developed, which often rely on either feature extraction or feature selection techniques for reducing the dimension of input data before being fed into machine learning models. This aims to make the detection complexity low enough for real-time operations, which is particularly vital in any intrusion detection systems. This paper provides a comprehensive comparison between these two feature reduction methods of intrusion detection in terms of various performance metrics, namely, precision rate, recall rate, detection accuracy, as well as runtime complexity, in the presence of the modern UNSW-NB15 dataset as well as both binary and multiclass classification. For example, in general, the feature selection method not only provides better detection performance but also lower training and inference time compared to its feature extraction counterpart, especially when the number of reduced features K increases. However, the feature extraction method is much more reliable than its selection counterpart, particularly when K is very small, such as $K = 4$. Additionally, feature extraction is less sensitive to changing the number of reduced features K than feature selection, and this holds true for both binary and multiclass classifications. Based on this comparison, we provide a useful guideline for selecting a suitable intrusion detection type for each specific scenario, as detailed in Table 14 at the end of Sect. 4. Note that such the comparison between feature selection and feature extraction over UNSW-NB15 as well as theoretical guideline have been overlooked in the literature.

Keywords Intrusion detection · UNSW-NB15 · Feature selection · Feature extraction · PCA · Machine learning · Internet of Things · Runtime · Binary/multiclass classification · NIDS · IoT

1 Introduction

Internet of Things (IoTs) has recently witnessed an explosive expansion in a broad range of daily life and industrial applications [1–3], such as healthcare, smart homes, smart cities, smart energy, smart agriculture, and intelligent transportation. The IoT networks aim to provide internet connections for transferring data among massive IoT devices, such as interconnected sensors, drones, actuators, smart vehicles and smart home appliances [2], using

✉ Hung Tran
hung.tran@phenikaa-uni.edu.vn

Vu-Duc Ngo
duc.ngo@mobifone.vn

Tuan-Cuong Vuong
cngvng123@gmail.com

Thien Van Luong
thien.luongvan@phenikaa-uni.edu.vn

[1] Research and Development Center, MobiFone Corporation, Hanoi 11312, Vietnam

[2] School of Electronics and Electrical Engineering, Hanoi University of Science and Technology, Hanoi 11657, Vietnam

[3] Faculty of Computer Science, Phenikaa University, Hanoi 12116, Vietnam

either wired or wireless communications. However, most of these IoT devices are low-cost, low-power and limited-resource, making them highly vulnerable to cyber attacks as well as intrusive activities. Therefore, it is vital to develop network intrusion detection systems (NIDS) that can promptly and reliably identify and prevent malicious attacks to IoT networks. For this, a wide range of machine learning-based intrusion detection techniques have been designed for IoT, along with a number of public network traffic datasets [2, 4]. These datasets often contain a large number of features, in which many are irrelevant or redundant, which adversely affect both the complexity and accuracy of machine learning algorithms. Thus, many feature reduction methods have been developed for NIDS, in which feature selection and feature extraction are two of the most popular ones [2, 4], as discussed next.[1]

In NIDS, feature selection has been widely used for reducing the dimensionality of original traffic data. For example, in [8], a mutual information (MI)-based feature selection algorithm was proposed in combination with a classifier called least square support vector machine, which achieves higher accuracy and lower runtime complexity than the existing schemes, over three datasets, namely, KDD99 [9], NSLKDD [10] and Kyoto 2006+ [11]. Before that a MI-based scheme was also proposed for NIDS in [12], which however suffers from higher computational complexity than the approach in [8]. Additionally, several approaches that rely on genetic algorithm (GA) as a search strategy to select the best subset of features can be found in [13, 14]. These methods provide lower false alarm rates than the baselines, where UNSW-NB15 [15] and KDD99 [9] datasets are used. In [16], a hybrid feature selection approach, which relies on the association rule mining and the central points of attribute values, was developed, showing that UNSW-NB15 dataset achieves a better evaluation than NSLKDD. In [17], another hybrid feature selection method that comprises particle swarm optimization (PSO), ant colony algorithm, and GA was proposed, learning to better detection performance than the baselines such as GA [13], in the presence of both NSLKDD and UNSW-NB15 datasets. In [18], a Pigeon inspired optimizer was used for selecting features of NIDS, which achieves higher accuracy than the PSO [13] and hybrid association rules methods [16]. Note that the aforementioned feature selection schemes often suffer from high computational cost, especially for those relying on GA, PSO or machine learning-based classifiers. For this, a correlation-based feature selection method that offers low computational cost was investigated for NIDS over KDD99 and UNSW-NB15 datasets in [19], taking the correlation level among features into account. Recently, this correlation-based method was combined with ensemble-based machine learning classifiers to significantly improve the accuracy of NIDS [20], at the cost of higher complexity. Hence, aiming at a real-time and low-latency attack detection solutions, this work will more focus on the correlation-based feature selection method.[2]

Unlike feature selection, which retains a subset of the original features in NIDS, feature extraction attempts to compress a large amount of original features into a low-dimensional vector so that most of information is retained. There are a number of feature extraction techniques that have been applied for reducing data dimension in NIDS, such as principal component analysis (PCA), linear discriminant analysis (LDA), and neural network-based autoencoder (AE). For instance, in [23], PCA was applied to significantly reduce the dimension of KDD99 dataset, improving both accuracy and speed of NIDS, where support vector machine was used for attack classification. Then, several variants of PCA were adopted to intrusion detection, such hierarchical PCA neural networks [24] and kernel PCA with GA [25], which can enhance the detection precision for low-frequent attacks. Some of applications of PCA to recent network traffic datasets such as UNSW-NB15 and CICIDS2017 [26] can be found in [27, 28]. In addition to PCA, LDA was also employed as a feature reduction method for NIDS in [29], which remarkably reduces the computational complexity of NIDS. Then, in [30, 31], both PCA and LDA were combined into a two-layer dimension reduction, which is capable of reliably detecting low-frequency malicious activities, such as User to Root and Remote to Local, over NSLKDD dataset. To further improve the efficiency of feature extraction in NIDS, a AE-based neural network was used in a range of research works [32–37]. In particular, a stacked sparse AE approach was developed in [32] to conduct a non-linear mapping between high-dimensional data and low-dimensional data over NSLKDD dataset. In [33], a deep stacked AE was used to noticeably reduce the number of features to 5 and 10 for binary and multiclass classification, respectively, leading to better accuracy than the previous methods. Additionally, a number of AE architectures based on long short-term memory (LSTM) were developed for dimensionality reduction of NIDS, such as variational LSTM [35] and bidirectional LSTM [34], which can efficiently address imbalance and high-dimensional problems. Note that these AE-based methods suffer from a high computational cost compared to PCA and LDA, both in training and testing phases. To address this issue, a network

[1] Note that several recent works that apply deep learning and blockchain to secure IoT networks can be found in [5–7], in the fields of healthcare system, unmanned aerial vehicle and Android malware.

[2] Note that several matrix factorization-based dimensionality reduction methods were developed for gene expression analysis in [21, 22].

pruning algorithm has been recently proposed in [36] to considerably lower complexity of AE structures in extracting features of NIDS. In [37], a network architecture uses an AE based on convolutional and recurrent neural networks to extract spatial and temporal features without human engineering.

It is worth noting that most of the aforementioned papers have focused on either improving the detection accuracy or reducing the computational complexity of NIDS, by using machine learning-based classifications in combination with feature engineering methods such as feature selection and feature extraction for reducing data dimensionality. However, a comprehensive comparison between these two feature reduction methods has been overlooked in the literature. Our paper appears to address this gap. In particular, we first provide an overview of NIDS, with a focus on the phase of feature reduction, where feature extraction with PCA and feature selection with correlation matrix are the two promising candidates for realistic low-latency operations of NIDS. Then, using the modern UNSW-NB15 dataset, we thoroughly compare the detection performance (precision, recall, F1-score) as well as runtime complexity (training time and inference time) of these two methods, taking into account both binary and multiclass classifications as well as the same number of selected/extracted features denoted as K. Based on our extensive experiments, we found that feature selection generally achieves higher detection accuracy and requires less training and inference times when the number of reduced features K is large enough, while feature extraction outperforms feature selection when K gets smaller, such as $K = 4$ or less. Furthermore, in order to gain a deeper insight into detection behaviors of both methods, we investigate and compare their accuracy for each attack class when varying K, based on their best machine learning classifiers, which revealed that feature extraction is not only less sensitive to varying the number of reduced features but also capable of detecting more diverse attack types than feature selection. Additionally, both tend to be able to detect more attacks, i.e., Abnormal classes, when having more features selected or extracted. Relying on such comprehensive observations, we provide a theoretical guideline for selecting an appropriate intrusion detection type for each specific scenario, as detailed in Table 14 at the end of Sect. 4, which is, to the best of our knowledge, not available in the literature.

The rest of this paper is organized as follows. Section 2 discusses machine learning-based network intrusion detection methods for IoT networks. The overview of UNSW-NB15 dataset and data pre-processing are explained in Sect. 3. Section 4 provides the experimental results and discussion. Finally, Sect. 5 concludes this paper.

2 Machine learning-based network intrusion detection methods

In this section, we describe an overview of a NIDS based on machine learning, followed by details on the two major feature reduction methods, namely, feature selection and feature extraction.

2.1 Overview of NIDS

A NIDS consists of three major components, namely, data pre-processing, feature reduction, and attack classification, as illustrated in Fig. 1. In particular, in the first phase, the raw data is denoted as the dataframe $\mathbf{Z}$, whose features may include unexpected or non-numeric values, such as null or nominal. $\mathbf{Z}$ is pre-processed in order to either replace these unexpected values with valid ones or transform them to the numeric format using one-hot encoding. Several features that do not affect detection performance, such as the source IP address and the source port number, are dropped out. Furthermore, depending on the classifier we use for identifying attacks, we may use the normalization technique, for example, to constrain the values of all features, i.e., the elements of the output vector of the first phase $\mathbf{X}$ in Fig. 1, to range from 0 to 1. We will discuss this in detail in Sect. 3 when presenting UNSW-NB15 dataset.

As such, after the first phase, the pre-processed data $\mathbf{X} \in \mathbb{R}^{D \times N}$ is likely to have much more features than the original data $\mathbf{Z}$, particularly due to the use of one-hot encoding, where D is the number of dimensions, or equivalently, the number of features of $\mathbf{X}$, and N is the number of data samples. For example, when UNSW-NB15 dataset is used, the dimension of data increases from 45 to nearly 200, which is too large for classification techniques to quickly recognize the attack type. In order to address this fundamental issue, in the second phase, we need to reduce the number of features that will be used for the attack classification phase (the last phase in Fig. 1). For this, two feature reduction methods called feature selection and feature extraction are widely used to either select or extract a small number of most important features from pre-processed traffic data. This procedure also helps to remove a large amount of unnecessary features, which not also increase the complexity of NIDS, but also degrade its detection performance, as will be illustrated in experimental results in Sect. 4. Herein, the output data of the feature reduction block is denoted as vector $\mathbf{U} \in \mathbb{R}^{K \times N}$ in Fig. 1, which is expected to have a much lower dimension than $\mathbf{X}$, i.e., $K \ll D$, while retaining its most important information.

Finally, in the third phase of NIDS, a number of binary and multiclass classification approaches based on machine

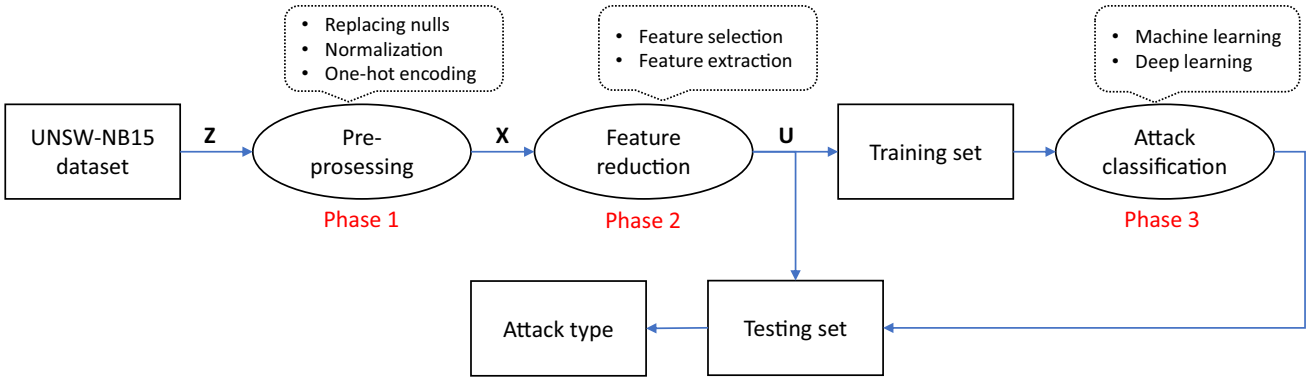


Fig. 1 Block diagram of a network intrusion detection system

learning, such as decision tree, random forest and multilayer perception neural networks, are employed to detect the attack type. Relying on attack detection results, the system administrators can promptly make a decision to prevent malicious activities, ensuring the security of IoT networks. Here, note that the detection performance and latency of a NIDS strongly depend on which classifier and which feature reduction method it employs. Therefore, in this contribution, we comprehensively investigate detection performance (in terms of recall, precision, F1-score) and latency (in terms of training time and inference time) of different detection methods in presence of both feature selection and feature extraction as well as different machine learning classifiers. We also focus more on the comparison between these two feature reduction methods, which will be described in detail in the following subsections.

2.2 Feature selection

There are a number of feature selection techniques used in intrusion detection, namely, information gain, IG [8] and feature correlation [19, 20, 38]. In this work, we focus on using feature correlation for selecting important features, since this method has been shown to achieve competitive detection accuracy and complexity compared to other selection counterparts. Using this correlation-based method, we aim to select features that are most correlated to other features based on the correlation matrix calculated from the training dataset. More specifically, the correlation coefficient between feature Ω_1 and feature Ω_2 is calculated based on the numeric pre-processed training dataset $\mathbf{X}$ as follows [38]:

$$\mathcal{C}_{\Omega_1,\Omega_2} = \frac{\sum_{i=1}^{N}(\alpha_i - E_{\Omega_1})(\beta_i - E_{\Omega_2})}{\sqrt{\sum_{i=1}^{N}(\alpha_i - E_{\Omega_1})^2}\cdot\sqrt{\sum_{i=1}^{N}(\beta_i - E_{\Omega_2})^2}}, \quad (1)$$

where α_i and β_i are the values of these two features, $E_{\Omega_1} = \sum_{i=1}^{N}\alpha_i/N$ and $E_{\Omega_2} = \sum_{i=1}^{N}\beta_i/N$ are their means over N training data samples. Note that after preprocessing the raw data $\mathbf{Z}$ to obtain $\mathbf{X}$, all features of $\mathbf{X}$ are now numeric, i.e., α_i and β_i are numeric, making (1) applicable to process. By doing this, we obtain a $D \times D$ correlation matrix $\mathbf{C}$, whose elements are given by $c_{ij} = \mathcal{C}_{\Omega_i,\Omega_j}$ for $i,j = 1,2,\ldots,D$. The average correlation of feature Ω_i to other features is computed as follows:

$$C_i = \frac{\sum_{j=1}^{D} c_{ij}}{D}, \quad (2)$$

where $c_{ii} = 1$ for $j = i$ and $c_{ij} \in [-1;1]$ for $j \neq i$. Note that the self-correlation coefficient c_{ii} does not affect selection results, since it contributes the same amount to all C_i for $i = 1,2,\ldots,D$. Then, using a suitable threshold, as will be detailed in Sect. 4, we are able to select K most important features corresponding to K largest elements C_i.

It is worth noting that we only need to calculate such feature correlation in the training phase, while in the testing phase, we simply pick up K features from the high-dimensional data $\mathbf{X}$ to form the reduced-dimensional data $\mathbf{U}$ in Fig. 1. This does not require much computational resource when compared with the feature extraction method, which is presented next.

2.3 Feature extraction

PCA [23] and AE [36] are the two major feature extraction methods used in the NIDS. Different from feature selection, whose selected features are identical to those appearing in the original data, these feature extraction techniques compress the high-dimensional data $\mathbf{X}$ into the low-dimensional data $\mathbf{U}$ using either a projection matrix or an AE-based neural network learned from training dataset. Note that the AE approach usually suffers from high computational complexity of a deep neural network (DNN), leading to higher latency than the PCA. Thus, in this work, we concentrate on the PCA-based feature

extraction approach in order to fulfill a strict requirement on the latency of the NIDS for promptly preventing severe cyber attacks.

In what follows, we introduce the procedure of producing the $D \times K$ projection matrix $\mathbf{W}$ in the training phase, and how to utilize this matrix in the testing phase. In particular, based on the pre-processed training data $\mathbf{X}$ of N samples, we normalize it by subtracting all samples of $\mathbf{X}$ by its mean over all training samples, i.e., the normalized data is given as follows: $\hat{\mathbf{X}} = \mathbf{X} - \bar{\mathbf{X}}$, where $\bar{\mathbf{X}}$ is the mean vector. Then, we compute the $D \times D$ covariance matrix of training data as follows: $\mathbf{R} = \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^T$. Based on this, we determine its eigenvalues and eigenvectors, from which, we select K eigenvectors corresponding to K largest eigenvalues for constructing the $D \times K$ projection matrix $\mathbf{W}$. Herein, these K eigenvectors are regarded as the principal components that create a subspace, which is expected to be significantly close to the normalized high-dimensional data $\hat{\mathbf{X}}$. Finally, the compressed data is determined by $\mathbf{U} = \mathbf{W}^T\hat{\mathbf{X}}$, which now has the size of $K \times N$ instead of $D \times N$ of the original data.

In the testing phase, for each new data point $\mathbf{x}_i \in \mathbb{R}^D$, its dimension is reduced using PCA according to $\mathbf{u}_i = \mathbf{W}^T(\mathbf{x}_i - \bar{\mathbf{X}})$. This indicates that the output of the training phase of PCA includes both the projection matrix $\mathbf{W}$ and the mean vector of all training samples $\bar{\mathbf{X}}$. It should be noted that such projection matrix calculation would be computationally expensive, particularly when D and K are large.

3 Overview of UNSW-NB15 dataset

We now present some key information about UNSW-NB15 dataset, which will be used in our experiments in Sect. 4 to compare between feature selection and feature extraction. Then, the data pre-processing for this dataset is also discussed.

3.1 Key information of UNSW-NB15 dataset

UNSW-NB15 dataset was first introduced in [15], which offers better real modern normal and abnormal synthetical network traffic compared with the previous NIDS datasets such as KDD99 [9] and NSLKDD [10]. A total of 2.5 million records of data are included in the UNSW-NB15 dataset, in which there are one normal class and nine attack classes: Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms. Flow features, basic features, content features, time features, additional generated features, and labeled features are 6 feature groups, which consist of a total of 49 features in the original data [15]. However, in this work, we use a 10% cleaned dataset of UNSW-NB15, which includes a training set of 175,341 records and a test set of 82,332 records. There are a few minority classes with proportions of less than 2%, including Analysis, Backdoor, Shellcode, and Worms (see Figs. 2, 3). In the 10% dataset, some unrelevant features were removed, such as *scrip* (source IP address), *sport* (source port number), *dstip* (destination IP address), and *dsport* (destination port number). Therefore, the number of features was reduced to 45, including 41 numerical features and 4 nominal features.

3.2 Pre-processing dataset

As mentioned above, the 10% dataset of UNSW-NB15 has 45 features, including 41 numerical features and 4 nominal features. We remove the *id* feature in numerical features, since it does not affect the detection performance. The *attack_cat* nominal feature that contains the names of attack categories is also removed. Thus, there are three remaining helpful nominal features, namely, *proto, service, state*. In addition, null values appearing in the *service* feature are treated as 'other' type of service.

One-hot encoding is used for transforming nominal features, i.e., *proto, service, state*, to numerical values. For example, assume that the *proto* feature has a total of three different values, namely, A, B, C, then its one-encoding will result in three numerical features, namely, *proto_A, proto_C, proto_C*, whose values are 0 or 1, as illustrated in Table 1. As a result, after pre-processing data, the number of features will increase from 45 features in $\mathbf{Z}$ to approximately 200 features in $\mathbf{U}$ (see Fig. 1), where many of them are not really helpful in classifying attacks. Therefore, it is necessary to reduce such a large number of features to a few of the most important features, which allows to reduce the complexity of machine learning models in the classification phase. Finally, we note that when feature extraction is used, we normalize the input feature with the min-max normalization method [39] to improve the classification accuracy, while we do not use that data normalization for feature selection, since it does not improve the performance.

4 Experimental results and discussion

We now present extensive experimental results for investigating the performance of the NIDS using both feature selection and feature extraction methods described in Sect. 2, in combination with a range of machine learning-based classification models. More particularly, the performance metrics used for comparison include recall (R), precision (P), F1-score, training time and inference time,

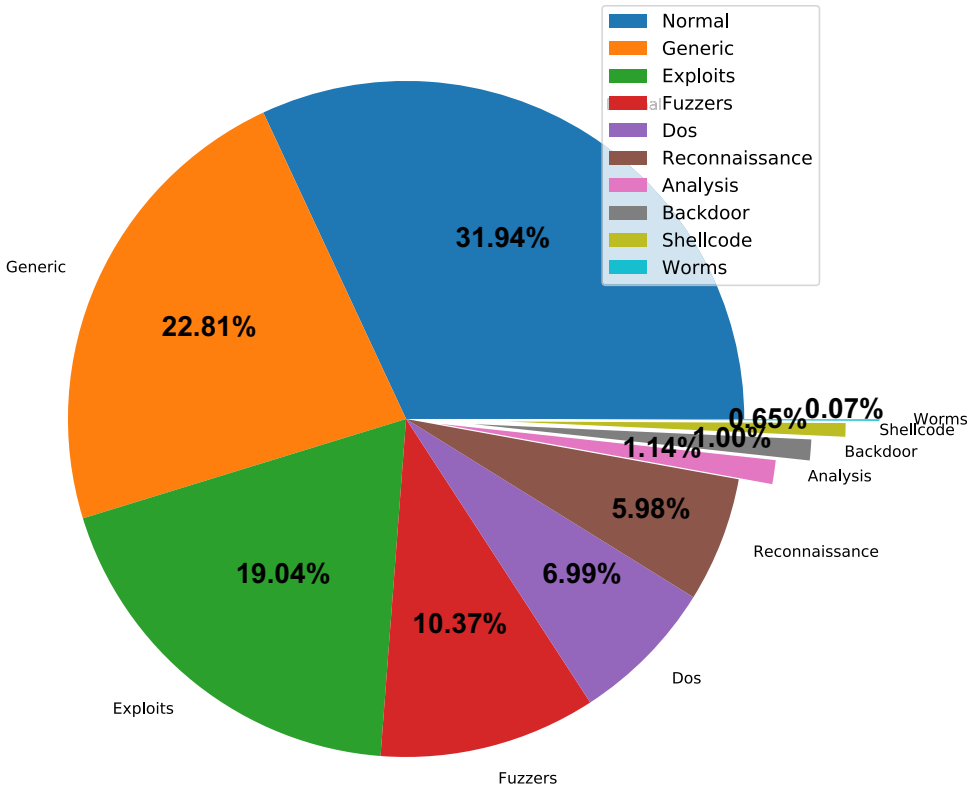


Fig. 2 Proportions of 10 classes in training dataset of UNSW-NB15

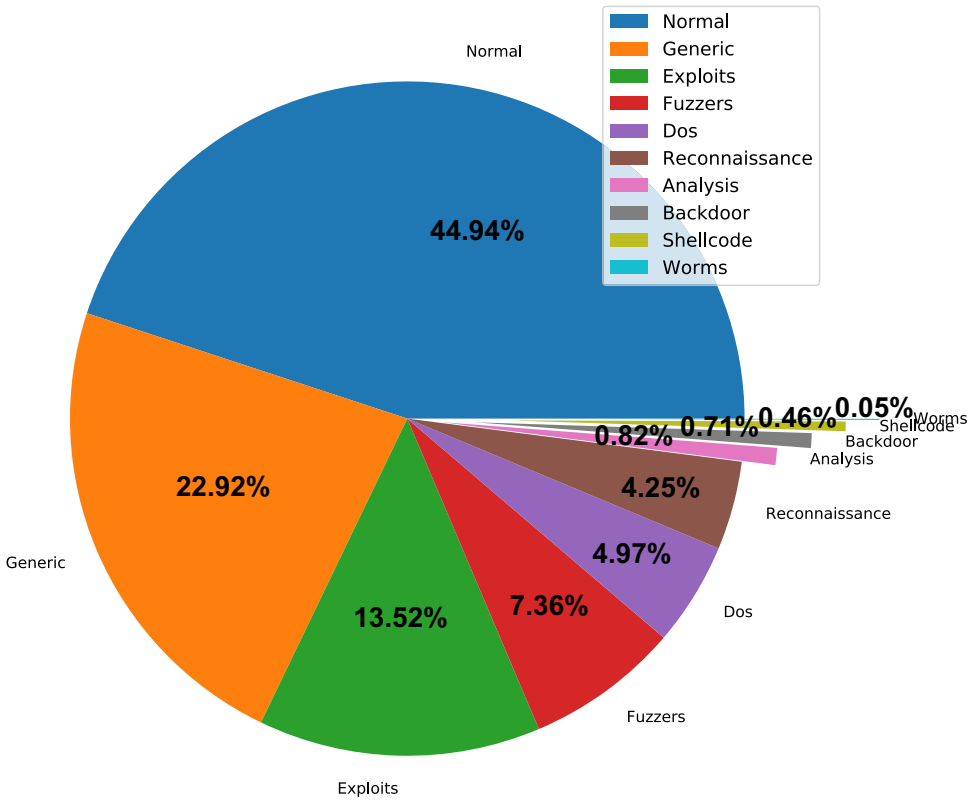


Fig. 3 Proportions of 10 classes in testing dataset of UNSW-NB15

Table 1 An example of one-hot encoding for a nominal feature

Proto	Proto_A	Proto_B	Proto_C
A	1	0	0
B	0	1	0
C	0	0	1

which will be explained in detail in Sect. 4.1. Both binary and multiclass classifications are considered. We also investigate the accuracy for each attack class to provide an insight into the behaviors of different detection methods. Last but not least, based on our extensive comparison between feature selection and feature extraction, we provide a helpful guideline on how to choose an appropriate detection technique for each specific scenario.

4.1 Implementation setting

4.1.1 Computer configuration

The configuration of our computer, its operation system as well as a range of Python packages used for implementing intrusion detection algorithms in this work are detailed in Table 2.

4.1.2 Evaluation metrics

We consider the following performance metrics: precision, recall, F1-score, as well as training time and inference time. In particularly, F1-score is calculated based on precision and recall as follows:

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \tag{3}$$

which is regarded as a harmonic mean of precision and recall.

As shown in Fig. 1, the two feature reduction methods considered in this work go through the same pre-processing data step, so we do not take the time required for this step into account when estimating their training and inference time. Particularly, the training time consists of the training time of classification models and the time duration consumed by feature reduction in training (FR_train), as follows:

$$\text{Training time} = \text{time}_{train} + \text{time}_{FR\_train}. \tag{4}$$

Meanwhile, the inference time consists of the prediction time of machine learning classifiers and the time duration required for feature reduction in the testing phase, given by

$$\text{Inference time} = \text{time}_{predict} + \text{time}_{FR\_test}. \tag{5}$$

4.1.3 Classification models

We use five machine learning models to do both binary and multiclass classification tasks, which are available in Python Scikit-learn library, namely, Decision Tree, Random Forest (max_depth = 5), K-nearest Neighbors (n_neighbors = 5), Multi-layer Perceptron (MLP) (max_{i}ter = 100, hidden_layer_sizes = 200), and Bernoulli Naive Bayes. Additionally, for a better insight of feature selection, we provide lists of 4, 8 and 16 selected features in Table 3, as well as the corresponding thresholds of the average correlation used to achieve those numbers of selected features.

4.2 Binary classification

We first investigate the detection performance and runtime of feature selection and feature extraction methods when using binary classification in Tables 4, 5 and 6 for 4, 8, 16 selected/extracted features, respectively. In these tables, the best values (i.e. the maximum values of precision, recall, and F1-score, and the minimum values of training and inference times at each column of the tables) are highlighted in bold, especially the best values for both feature selection and feature extraction are highlighted in italics. The training time is measured in second (s), while the inference time for each data sample is measured in millisecond (μs).

In terms of detection performance, it is shown from Tables 4, 5 and 6 that when the number of reduced

Table 2 Hardware and environment specification

Unit	Description
Processor	Intel Core i5-10400F (2.66 Hz, 6 cores 12 threats, 12 MB Cache, 65 W)
RAM	16 GB
GPU	Nvidia GTX 1650 OC-4G
Operating system	Ubuntu 20.04.4 LTS
Packages	Numpy, Matplotlib, Pandas, Scipy, Scikit-learn, Scikit-plot and Time

Table 3 Threshold setting and features selected

Threshold	Number	Features selected
0.011	4	'spkts', 'dpkts', 'dbytes', 'dloss'
0.0137	8	'dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'sloss', 'dloss', 'ct_state_ttl'
0.011	16	'dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'sloss', 'dloss', 'dinpkt', 'sjit', 'djit', 'tcprtt', 'synack', 'ackdat', 'response_body_len', 'ct_state_ttl', 'proto_{i}cmp'

Table 4 Feature Selection versus Feature Extraction: 4 selected/extracted features and binary classification

Models	Feature Extraction					Feature Selection				
	P	R	F1	Training (s)	Inference (μs)	P	R	F1	Training (s)	Inference (μs)
Decision Tree	85.33	84.35	84.84	22.73	3.73	**84.09**	**79.89**	**81.94**	14.53	*0.07*
Random Forest	85.76	81.22	83.42	32.32	18.12	77.86	75.11	76.46	17.13	3.29
KNeighbors	*86.19*	*84.67*	*85.42*	23.04	38.47	52.38	47.98	50.08	14.75	259.03
MLP	85.91	81.75	83.78	1011.31	39.37	75.76	74.75	75.25	1278.61	37.08
Naive Bayes	72.62	71.95	72.28	**20.37**	**3.62**	75.47	73.63	74.54	*14.3*	0.24

Table 5 Feature Selection versus Feature Extraction: 8 selected/extracted features and binary classification

Models	Feature Extraction					Feature Selection				
	P	R	F1	Training (s)	Inference (μs)	P	R	F1	Training (s)	Inference (μs)
Decision Tree	85.98	84.98	85.48	26.94	**3.95**	*87.87*	*87.07*	*87.47*	14.57	*0.11*
Random Forest	85.23	80.03	82.55	33.89	18.32	85.74	80.95	83.27	18.08	3.34
KNeighbors	**86.39**	**84.99**	**85.69**	23.41	44.02	87.08	85.98	86.53	14.99	51.16
MLP	86.14	82.42	84.24	1252.49	39.82	84.42	79.95	82.12	607.66	33.06
Naive Bayes	72.14	70.01	71.06	**21.76**	4.11	75.56	73.85	74.69	*14.33*	0.35

Table 6 Feature Selection versus Feature Extraction: 16 selected/extracted features and binary classification

Models	Feature Extraction					Feature Selection				
	P	R	F1	Training (s)	Inference (μs)	P	R	F1	Training (s)	Inference (μs)
Decision Tree	**86.43**	**85.47**	**85.95**	37.38	**5.38**	*87.41*	*86.61*	*87.01*	15.18	*0.17*
Random Forest	85.85	81.08	83.4	58.07	19.26	85.85	81.08	83.39	52.92	19.01
KNeighbors	86.09	84.5	85.29	38.74	1421.15	86.09	84.5	85.29	37.68	1426.22
MLP	86.36	84.67	83.04	1344.91	31.59	81.75	74.27	77.83	661.67	40.64
Naive Bayes	78.2	75.59	76.87	**36.64**	5.82	75.66	73.88	74.76	*14.46*	0.55

features (i.e. extracted or selected) K increases, the detection performance of feature extraction generally improves, while that of feature selection does not improve when we increase K from 8 to 16. In fact, the precision, recall and F1-score of feature selection even slightly degrade from Tables 5 and 6. This phenomenon is understandable due to the fact that if the number of selected features gets larger, it is likely to have more noisy or unimportant features appearing in the selected ones, which are expected to deteriorate the detection performance. Moreover,

comparing the two feature reduction methods, we find that when the number of reduced features is small, i.e., $K = 4$, the detection performance of feature extraction is much better than that of feature selection. For instance, in Table 4, the highest F1-score of feature extraction is 85.42% when the KNeighbors classifier is used, while that of feature selection is lower with 81.94% when the Decision Tree classifier is used. However, for larger K such as 8 and 16 in Tables 5 and 6, the feature selection method achieves better accuracy than its extraction counterpart, especially when using Decision Tree for classification. For example, when Decision Tree is employed in Table 5 to achieve the lowest inference time, the F1-score of feature selection is 87.47%, which is higher than that of feature extraction with 85.69%. It is also shown from Tables 4, 5 and 6 that when using feature selection, the Decision Tree classification method always provides the best precision, recall as well as F1-score. By contrast, the feature extraction method would enjoy the KNeighbors classifier when K are small, i.e., 4 or 8, while Decision Tree is only its best classifier when K becomes larger, such as $K = 16$.

In terms of the runtime performance, Tables 4, 5 and 6 demonstrate that both the training time and the inference time of feature selection is lower than that of feature extraction. This is because of the fact that the feature extraction method requires additional computational resources when compressing the high-dimensional data into low-dimensional data, as explained in Sect. 2, while the feature selection almost do not require any computing resources when just picking up K out of D features. More particularly, in Table 5, the best inference time of feature selection is 0.11 µs, which is 36 times lower than that of feature extraction being 3.95 µs, where the Decision Tree classifier is the best choice for both feature reduction methods for minimizing the inference time. Again, Decision Tree is one of the best classifiers for minimizing both training and inference times, in addition to the Naive Bayes classifier, which however does not achieve a good accuracy.

Finally, in order to better understand the attack detection performance of feature selection and feature extraction, in Table 7, we provide the accuracy comparison for each class in binary classification, namely, Normal and Abnormal. Similar to Tables 4, 5 And 6, we consider the number of reduced features K being 4, 8 and 16. Besides, based on the results obtained from these three tables, we only include the classifiers that offer the highest F1-scores for accuracy comparison for each class in Table 7, namely, MLP and KNeighbors for feature extraction and Decision Tree for feature selection. Herein, the highest accuracy for each class with respect to K is highlighted in bold, while the highest values in both feature selection and feature extraction are highlighted in italics. It is worth noting from this table that in both feature reduction methods, while the accuracy of detecting Normal class steadily improves when increasing K, that of detecting Abnormal class gradually degrades. This interestingly indicates that in order to detect more attacks, we should select small K rather than large K. In addition, Table 7 shows that for both feature reduction methods, the accuracy of Abnormal class is much higher than that of Normal class. Observe the average accuracy from this table, we find that the accuracy of feature extraction is less sensitive to varying K that that of feature selection, which varies significantly with respect to K.

4.3 Multiclass classification

We compare both the detection performance and runtime of feature selection and feature extraction in Tables 8, 9, and 10 for 4, 8, and 16 selected/extracted features, respectively, when multiclass classification is considered. Here, we still employ five machine learning models as in binary classification. As shown via these three tables, similar to the binary case, the precision, recall and F1-score of both methods generally improve when increasing the number of reduced features K. For example, the highest F1-scores of feature extraction are 74.11%, 75.39%, and 75.52%, while that of feature selection are 65.43%, 78.36% and 77.64%, when K = 4, 8, and 16 reduced features, respectively. As such, feature extraction outperforms its counterpart when K is small such as $K = 4$, however, this is no longer true when K gets larger such as $K = 8$ and 16, where feature selection performs much better than feature extraction. Again, akin to the binary classification, it is shown from Tables 9 and 10 that the detection performance of feature selection degrades when K increases from 8 to 16, mostly due to the impact of noisy or irrelevant features when having more features selected.

Table 7 Accuracy comparison for each class between feature selection and feature extraction using binary classification

Class	Feature Extraction (MLP/KNeighbors)			Feature Selection (Decision Tree)		
	$K = 4$	$K = 8$	$K = 16$	$K = 4$	$K = 8$	$K = 16$
Normal	70.64	71.55	**73.79**	57.21	*76.67*	76.09
Abnormal	**96.14**	96.02	94.99	*98.4*	95.55	95.21
Average	84.68	85.02	**85.47**	79.89	*87.07*	86.61

Table 8 Feature Selection versus Feature Extraction: 4 selected/extracted features and multiclass classification

Models	Feature Extraction					Feature Selection				
	P	R	F1	Training (s)	Inference (μs)	P	R	F1	Training (s)	Inference (μs)
Decision Tree	75.54	67.56	71.33	21.76	4.47	**69.71**	**61.65**	**65.43**	14.45	*0.09*
Random Forest	77.6	64.35	70.36	30.52	20.69	55.73	58.76	57.21	17.55	5.55
KNeighbors	76.98	*70.87*	73.8	20.49	38.57	50.28	45.87	47.97	14.67	258.84
MLP	*80.1*	68.96	*74.11*	1085.63	42.89	63.42	55.41	59.15	1519.35	43.67
Naive Bayes	62.82	50.81	56.18	**19.9**	**3.83**	41.55	59.68	48.99	*12.29*	0.59

Table 9 Feature Selection versus Feature Extraction: 8 selected/extracted features and multiclass classification

Models	Feature Extraction					Feature Selection				
	P	R	F1	Training (s)	Inference (μs)	P	R	F1	Training (s)	Inference (μs)
Decision Tree	76.43	69.36	72.72	27.49	**4.42**	80.18	*76.62*	*78.36*	14.65	*0.13*
Random Forest	78.32	66.74	72.07	34.13	20.66	78.82	68.65	73.38	18.38	5.48
KNeighbors	77.95	**72.86**	75.32	23.5	41.78	*80.27*	73.9	76.96	14.8	50.15
MLP	**79.90**	71.36	**75.39**	1180.75	47.77	65.14	68.96	66.99	591.4	42.79
Naive Bayes	65.73	51.77	57.92	**22.77**	4.52	43.47	59.67	50.3	*14.57*	1.08

Table 10 Feature Selection versus Feature Extraction: 16 selected/extracted features and multiclass classification

Models	Feature Extraction					Feature Selection				
	P	R	F1	Training (s)	Inference (μs)	P	R	F1	Training (s)	Inference (μs)
Decision Tree	77.33	70.11	73.55	37.68	**5.04**	79.59	*75.78*	*77.64*	15.17	*0.19*
Random Forest	78.36	66.71	72.07	53.12	21.27	*80.03*	68.02	73.54	22.17	5.69
KNeighbors	77.56	**72.03**	74.69	34.94	1405.43	78.79	63.91	70.58	14.72	1396.85
MLP	**79.44**	71.97	**75.52**	1428.97	47.51	64.42	69.01	66.63	895.37	49.51
Naive Bayes	74.57	60.59	66.87	**34.75**	6.34	47.28	59.75	52.79	*14.47*	1.09

Besides, unlike the binary case, where KNeighbors is the best classifier for feature extraction when K is small such as 4 and 8, with multiclass classification, MLP now provides the best detection performance of feature extraction for any values of K, as shown via Tables 8, 9, and 10. Meanwhile, feature selection still enjoys the Decision Tree classifier to achieve the highest detection performance, similar to the binary classification analyzed in the previous subsection, while MLP does not offer a good detection performance for feature selection. Additionally, the Naive Bayes classifier achieves the worst accuracy for both feature reduction methods.

With regard to the runtime comparison, again, Tables 8, 9, and 10 demonstrate that the training and inference times of feature selection are significantly lower than that of feature extraction. For example, using the same Decision Tree model for achieving the lowest runtime, in Table 9 when $K = 8$, the inference time of feature selection is 0.19 μs, which is 26 times lower than that of feature extraction with 5.04 μs. Similarly, it is shown from this table that the training time of feature selection is also 2 times lower than that of its extraction counterpart. In addition, the Decision Tree model provides the lowest inference time for both feature reduction methods, while the neural network-based MLP classifier exhibits both the highest inference and training times for them.

Finally, we compare the accuracy for detecting each attack type (including 9 attack classes and 1 normal class, as described in Sect. 3) between feature selection and feature extraction in Table 11, where the values of K are 4,

Table 11 Accuracy comparison for each class between feature selection and feature extraction using multiclass classification

Class	Feature Extraction (MLP)			Feature Selection (Decision Tree)		
	$K = 4$	$K = 8$	$K = 16$	$K = 4$	$K = 8$	$K = 16$
Analysis	0	0	0	1.03	*1.62*	0
Backdoor	0	0	0	6	*7.2*	6.17
DoS	0.81	10.61	**11.2**	8.12	*14.18*	12.96
Exploits	*86.79*	85.92	85.59	53.33	**84.65**	82.79
Fuzzers	*67.6*	66.46	58	43.1	**50.46**	49.64
Generic	96.22	96.24	**96.3**	*98.17*	98.04	97.38
Normal	62.47	62.5	**68.79**	59.19	*77.06*	76.51
Reconnaissance	50.69	60.55	**66.5**	39.99	*78.58*	78.4
Shellcode	0	7.67	**15.08**	0	*47.62*	42.86
Worms	0	2.27	**9.09**	11.36	*61.36*	34.09
Average	69.03	69.79	**72.29**	61.65	*76.62*	75.78

8, and 16 reduced features. Herein, we employ MLP and Decision Tree classifiers for feature extraction and feature selection, respectively, in order to achieve the best detection performance, as analyzed in the previous discussions. It is observed from Table 11 that feature selection performs better than feature extraction in most of classes, except for Exploits and Fuzzers classes. This table also shows that both methods are capable of achieving higher accuracy for Exploits, Generic, Normal and Reconnaissance classes than the remaining ones. Additionally, similar to the binary classification discussed in Table 7, the multiclass classification accuracy of feature extraction is less sensitive to the number of reduced features K than that of its selection counterpart. More importantly, feature selection with MLP is unable to correctly detect any samples of Analysis and Backdoor, even for all three values of K. By contrast, feature selection with Decision Tree classifier is capable of correctly detecting samples from all classes. We found that this is mainly due to the machine learning classifier rather than the feature reduction method we choose. In order to clarify this issue, we compare the accuracy for each class between the two feature reduction methods using the same Decision Tree and MLP classifiers in Tables 12 and 13, respectively. It is shown via Table 13 that using the same MLP, similar to feature extraction, feature selection is unable to detect any samples of Analysis and Backdoor correctly. Observe these two tables, we found that if the same classifier is employed, feature extraction tends to be able to detect more diverse attack types than feature selection. This is due to the fact that feature extraction can extract key information from all available features, leading to more diverse attack types, instead of relying solely on a subset of selected features as in the feature selection approach. In other words, feature selection tends to detect only attack types, which are highly correlated to the features it selects.

In summary, considering both binary and multiclass classification for the NIDS, the feature selection method not only provides better detection performance but also lower training and inference time compared to its feature extraction counterpart, especially when the number of reduced features K increases. However, the feature extraction method is much more reliable than its selection counterpart, particularly when K is very small, such as $K = 4$. Additionally, among five considered classifiers, while Decision Tree is the best classifier for improving the accuracy of feature selection, a neural network-based MLP is the best one for feature extraction. Last but not least, feature extraction is less sensitive to changing the number of reduced features K than feature selection, and this holds true for both binary and multiclass classifications. For more details, we provide a comprehensive comparison between feature selection and feature extraction in intrusion detection systems in Table 14.

5 Conclusions

We have compared two typical machine learning-based intrusion detection methods, namely, feature selection and feature extraction, in the presence of the modern UNSW-NB15 dataset, where both binary and multiclass classifications were considered. Our extensive comparison showed that when the number of reduced features is large enough, such as 8 or 16, feature selection not only achieves higher detection accuracy, but also requires less training and inference times than feature extraction. However, when the number of reduced features is very small, such as 4 or less, feature extraction notably outperforms its selection counterpart. Besides, the detection performance of feature selection tends to degrade when the number of selected features becomes too large, while that of feature

Table 12 Accuracy comparison for each class between feature selection and feature extraction using multiclass classification and the same Decision Tree classifier

Class	Feature Extraction (Decision Tree)			Feature Selection (Decision Tree)		
	$K = 4$	$K = 8$	$K = 16$	$K = 4$	$K = 8$	$K = 16$
Analysis	10.78	*11.82*	11.52	1.03	**1.62**	0
Backdoor	2.92	2.74	**5.83**	6	*7.2*	6.17
DoS	19.93	19.81	*24.75*	8.12	**14.18**	12.96
Exploits	59.07	61.77	**63.94**	53.33	*84.65*	82.79
Fuzzers	38.16	42.08	**44.46**	43.1	*50.46*	49.64
Generic	93.58	**96.22**	95.82	*98.17*	98.04	97.38
Normal	72.48	**73.13**	72.92	59.19	*77.06*	76.51
Reconnaissance	36.87	42.68	**45.68**	39.99	*78.58*	78.4
Shellcode	23.54	28.04	**32.8**	0	*47.62*	42.86
Worms	**15.91**	13.64	11.36	11.36	*61.36*	34.09
Average	67.6	69.42	**70.11**	61.65	*76.62*	75.78

Table 13 Accuracy comparison for each class between feature selection and feature extraction using multiclass classification and the same MLP classifier

Class	Feature Extraction (MLP)			Feature Selection (MLP)		
	$K = 4$	$K = 8$	$K = 16$	$K = 4$	$K = 8$	$K = 16$
Analysis	0	0	0	0	0	0
Backdoor	0	0	0	0	0	0
DoS	0.81	10.61	*11.2*	**6.6**	0.83	0.42
Exploits	*86.79*	85.92	85.59	19.16	25	**26.23**
Fuzzers	*67.6*	66.46	58	**30.86**	18.51	11.86
Generic	96.22	96.24	**96.3**	*96.96*	96.2	96.21
Normal	62.47	62.5	**68.79**	68.66	75.5	*93.08*
Reconnaissance	50.69	60.55	**66.5**	0.11	*73.31*	36.61
Shellcode	0	7.67	*15.08*	0	**1.06**	0
Worms	0	2.27	*9.09*	0	0	0
Average	69.03	69.79	*72.29*	58.38	63.88	**69.88**

Table 14 A summary of comparison between feature extraction and feature selection

No.	Content	Extraction	Selection
1	Higher accuracy when K is very small, such as $K = 4$	✔	
2	Higher accuracy when K gets large, such as $K = 8$ or 16		✔
3	Lower training time		✔
4	Lower inference time		✔
5	Detect more diverse attack types when using the same classifier	✔	
6	Less sensitive to the number of selected/extracted features K	✔	
7	MLP is the best classifier	✔	
8	Decision Tree is the best classifier		✔
9	Detection performance degrades when K is too large		✔
10	Detection performance steadily improves when K increases	✔	
12	Higher accuracy in detecting Exploits and Fuzzers classes	✔	
13	Higher accuracy in detecting 8 remaining classes, except for Exploits and Fuzzers		✔
14	Accuracy of Abnormal class is much higher than that of Normal class (Binary)	✔	✔
15	Accuracy of detecting Abnormal class degrades when K increases (Binary)	✔	✔
16	Higher accuracy for Exploits, Generic, Normal, Reconnaissance than remaining classes	✔	✔

extraction steadily improves. We also found that while MLP is the best classifier for feature extraction, Decision Tree is the best one for feature selection for achieving the highest attack detection accuracy. Finally, our accuracy analysis for each attack class demonstrated that feature extraction is not only less sensitive to varying the number of reduced features but also capable of detecting more diverse attack types than feature selection. Both tend to be able to detect more attacks, i.e., Abnormal classes, when having more features selected or extracted. We believe that such insightful observations about the performance comparison between two feature reduction methods give us a helpful guideline on choosing a suitable intrusion detection method for each specific scenario. Finally, note that our study evaluated the effectiveness of feature reduction methods only on the UNSW-NB15 dataset. In the future, we intend to explore whether our observations with UNSW-NB15 are applicable to other intrusion detection datasets, such as, NSL-KDD, KDD99, CICIDS2017, and DARPA1998. We also plan to thoroughly investigate the performance of various deep learning classification models for NIDS, and compare with existing machine learning models.

Acknowledgements This work was supported in part by the SSF Framework Grant Serendipity and R &D Project of Brighter Gates AB, Sweden.

Author contributions V-DN and T-CV wrote the main manuscript. T VL and HT reviewed and corrected the manuscript.

Funding Not applicable.

Data availability The paper does not include any supporting data.

Declarations

Conflict of interest All authors declare that they do not have any conflict of interest.

Ethical Approval Not applicable.

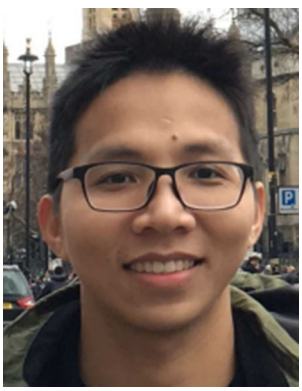
References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutor. **17**(4), 2347–2376 (2015)
2. Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., Faruki, P.: Network intrusion detection for IoT security based on learning techniques. IEEE Commun. Surv. Tutor. **21**(3), 2671–2701 (2019)
3. Kumar, P., Kumar, R., Garg, S., Kaur, K., Zhang, Y., Guizani, M.: A secure data dissemination scheme for IoT-based e-health systems using AI and blockchain. In: GLOBECOM 2022—2022 IEEE Global Communications Conference, 2022, pp. 1397–1403. IEEE (2022)
4. Mishra, P., Varadharajan, V., Tupakula, U., Pilli, E.S.: A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Commun. Surv. Tutor. **21**(1), 686–728 (2019)
5. Kumar, R., Kumar, P., Aloqaily, M., Aljuhani, A.: Deep learning-based blockchain for secure zero touch networks. IEEE Commun. Mag. **61**(2), 96–102 (2022)
6. Kumar, P., Kumar, R., Gupta, G.P., Tripathi, R., Jolfaei, A., Islam, A.N.: A blockchain-orchestrated deep learning approach for secure data transmission in IoT-enabled healthcare system. J. Parallel Distrib. Comput. **172**, 69–83 (2023)
7. D'Angelo, G., Palmieri, F., Robustelli, A., Castiglione, A.: Effective classification of Android Malware families through dynamic features and neural networks. Connect. Sci. **33**(3), 786–801 (2021)
8. Ambusaidi, M.A., He, X., Nanda, P., Tan, Z.: Building an intrusion detection system using a filter-based feature selection algorithm. IEEE Trans. Comput. **65**(10), 2986–2998 (2016)
9. KDD Cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed 10 Oct 2022
10. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A detailed analysis of the KDD CUP 99 data set. In: IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1–6 (2009)
11. Song, J., Takakura, H., Okabe, Y., Eto, M., Inoue, D., Nakao, K.: Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation, pp. 29–36. Association for Computing Machinery, New York (2011)
12. Amiri, F., Yousefi, M.R., Lucas, C., Shakery, A., Yazdani, N.: Mutual information-based feature selection for intrusion detection systems. J. Netw. Comput. Appl. **34**(4), 1184–1199 (2011)
13. Khammassi, C., Krichen, S.: A GA-LR wrapper approach for feature selection in network intrusion detection. Comput. Secur. **70**, 255–277 (2017)
14. Aslahi-Shahri, B.M., Rahmani, R., Chizari, M., Maralani, A., Eslami, M., Golkar, M.J., Ebrahimi, A.: A hybrid method consisting of GA and SVM for intrusion detection system. Neural Comput. Appl. **27**(6), 1669–1676 (2016)
15. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: Military Communications and Information Systems Conference (MilCIS), 2015, pp. 1–6 (2015)
16. Moustafa, N., Slay, J.: A hybrid feature selection for network intrusion detection systems: central points. arXiv e-prints (2017). arXiv:1707.05505
17. Tama, B.A., Comuzzi, M., Rhee, K.-H.: TSE-IDS: a two-stage classifier ensemble for intelligent anomaly-based intrusion detection system. IEEE Access **7**, 94497–94507 (2019)
18. Alazzam, H., Sharieh, A., Sabri, K.E.: A feature selection algorithm for intrusion detection system based on Pigeon inspired optimizer. Expert Syst. Appl. **148**, 113249 (2020)
19. Moustafa, N., Slay, J.: The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. Inf. Sec. J. Glob. Perspect. **25**(1–3), 18–31 (2016)
20. Moustafa, N., Turnbull, B., Choo, K.-K.R.: An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of Internet of Things. IEEE Internet Things J. **6**(3), 4815–4830 (2019)
21. Saberi-Movahed, F., Rostami, M., Berahmand, K., Karami, S., Tiwari, P., Oussalah, M., Band, S.S.: Dual regularized unsupervised feature selection based on matrix factorization and minimum redundancy with application in gene selection. Knowl. Based Syst. **256**, 109884 (2022)

22. Azadifar, S., Rostami, M., Berahmand, K., Moradi, P., Oussalah, M.: Graph-based relevancy–redundancy gene selection method for cancer diagnosis. Comput. Biol. Med. **147**, 105766 (2022)
23. Xu, X., Wang, X.: An adaptive network intrusion detection method based on PCA and support vector machines. In: Proceedings of the First International Conference on Advanced Data Mining and Applications, 2005, pp. 696–703 (2005)
24. Liu, G., Yi, Z., Yang, S.: A hierarchical intrusion detection model based on the PCA neural networks. Neurocomputing **70**(7–9), 1561–1568 (2007)
25. Kuang, F., Xu, W., Zhang, S.: A novel hybrid KPCA and SVM with GA model for intrusion detection. Appl. Soft Comput. **18**, 178–184 (2014)
26. Sharafaldin, I., Habibi Lashkari, A., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy—ICISSP, 2018, pp. 108–116 (2018)
27. Abdulhammed, R., Faezipour, M., Musafer, H., Abuzneid, A.: Efficient network intrusion detection using PCA-based dimensionality reduction of features. In: International Symposium on Networks, Computers and Communications (ISNCC), 2019, pp. 1–6 (2019)
28. Qi, L., Yang, Y., Zhou, X., Rafique, W., Ma, J.: Fast anomaly identification based on multiaspect data streams for intelligent intrusion detection toward secure Industry 4.0. IEEE Trans. Ind. Inform. **18**(9), 6503–6511 (2022)
29. Tan, Z., Jamdagni, A., He, X., Nanda, P.: Network intrusion detection based on LDA for payload feature selection. In: IEEE GLOBECOM Workshops, 2010, pp. 1545–1549 (2010)
30. Pajouh, H.H., Javidan, R., Khayami, R., Dehghantanha, A., Choo, K.-K.R.: A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in IoT backbone networks. IEEE Trans. Emerg. Top. Comput. **7**(2), 314–323 (2019)
31. Pajouh, H.H., Dastghaibyfard, G., Hashemi, S.: Two-tier network anomaly detection model: a machine learning approach. J. Intell. Inf. Syst. **48**(1), 61–74 (2017)
32. Yan, B., Han, G.: Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system. IEEE Access **6**, 41238–41248 (2018)
33. Khan, F.A., Gumaei, A., Derhab, A., Hussain, A.: A novel two-stage deep learning model for efficient network intrusion detection. IEEE Access **7**, 30373–30385 (2019)
34. Popoola, S.I., Adebisi, B., Hammoudeh, M., Gui, G., Gacanin, H.: Hybrid deep learning for botnet attack detection in the Internet-of-Things networks. IEEE Internet Things J. **8**(6), 4944–4956 (2021)
35. Zhou, X., Hu, Y., Liang, W., Ma, J., Jin, Q.: Variational LSTM enhanced anomaly detection for industrial big data. IEEE Trans. Ind. Inform. **17**(5), 3469–3477 (2021)
36. Dao, T.-N., Lee, H.: Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection. IEEE Internet Things J. **9**(16), 14438–14451 (2022)
37. D'Angelo, G., Palmieri, F.: Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. J. Netw. Comput. Appl. **173**, 102890 (2021)
38. Hall, M.A.: Correlation-based feature selection for machine learning. PhD Dissertation, The University of Waikato (1999)
39. Kotsiantis, S.B., et al.: Data preprocessing for supervised learning. Int. J. Comput. Electr. Autom. Control Inf. Eng (2006). https://doi.org/10.5281/zenodo.1082415

Vu-Duc Ngo Vu-Duc Ngo received the Ph.D. Degree from the Korea Advanced Institute of Science and Technology in 2011. From 2007 to 2009, he was a Co-founder and the CTO of Wichip Technologies, Inc., USA. Since 2009, he has been a Co-founder and the Director of uVision Jsc, Vietnam. Since 2012, he has been serving as a BoM Member of the National Program on Research, Training, and Construction, High-Tech Engineering Infrastructure of Vietnam. He is currently a Researcher with the MobiFone Research and Development Center, MobiFone Corporation, and also a Lecturer with the School of Electrical and Electronics Engineering, Hanoi University of Science and Technology, Hanoi, Vietnam. His research interests are in the fields of SoC, NoC design and verification, VLSI design for multimedia codecs, and wireless communications PHY layer. He was a Recipient of the IEEE 2006 ICCES, the IEEE 2012 ATC and the NICS 2021 Best Paper Awards.

Tuan-Cuong Vuong Tuan-Cuong Vuong is currently a second year Bachelor Student, working at AIoT Lab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam. His research interests include applied machine learning and deep learning in cyber security and computer vision.



Thien Van Luong Thien Van Luong is currently a Lecturer with the Faculty of Computer Science, and a Leader of AIoT Lab (https://aiot.phenikaa-uni.edu.vn/), Phenikaa University, Vietnam. He was a Research Fellow with University of Southampton, UK Prior to that he obtained the Ph.D. Degree at Queen's University Belfast, UK, and the B.S. Degree at Hanoi University of Science and Technology, Vietnam. His research interests include

applied machine learning in signal processing and wireless communications.

Hung Tran Hung Tran received the B.S. and M.S. Degrees in Information Technology from Vietnam National University, Hanoi, Vietnam, in 2002 and 2006, respectively, and the Ph.D. Degree from the Blekinge Institute of Technology, Sweden, in March 2013. In 2014, he was with the Electrical Engineering Department, ETS, Montreal, Canada. From 2015 to 2020, he was a Researcher at Malardalen University, Sweden. He is currently working as a Lecturer at the Computer Science Department, Phenikaa University, Vietnam. Besides doing research in the areas of wireless communication, he is also interested in topics of natural language processing and artificial intelligence, which have been applied to develop core engines for the academic gates platform (https://www.academicgates.com).

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com