Journal of Big Data

RESEARCH Open Access

Optimizing IoT intrusion detection system: feature selection versus feature extraction in machine learning

Jing Li[1], Mohd Shahizan Othman[1], Hewan Chen[2*] and Lizawati Mi Yusuf[1]

*Correspondence:
chw@cjlu.edu.cn

[1] University of Technology Malaysia, Johor Bahru, Malaysia
[2] China Jiliang University, Hangzhou, China

Abstract

Internet of Things (IoT) devices are widely used but also vulnerable to cyberattacks that can cause security issues. To protect against this, machine learning approaches have been developed for network intrusion detection in IoT. These often use feature reduction techniques like feature selection or extraction before feeding data to models. This helps make detection efficient for real-time needs. This paper thoroughly compares feature extraction and selection for IoT network intrusion detection in machine learning-based attack classification framework. It looks at performance metrics like accuracy, f1-score, and runtime, etc. on the heterogenous IoT dataset named Network TON-IoT using binary and multiclass classification. Overall, feature extraction gives better detection performance than feature selection as the number of features is small. Moreover, extraction shows less feature reduction compared with that of selection, and is less sensitive to changes in the number of features. However, feature selection achieves less model training and inference time compared with its counterpart. Also, more space to improve the accuracy for selection than extraction when the number of features changes. This holds for both binary and multiclass classification. The study provides guidelines for selecting appropriate intrusion detection methods for particular scenarios. Before, the TON-IoT heterogeneous IoT dataset comparison and recommendations were overlooked. Overall, the research presents a thorough comparison of feature reduction techniques for machine learning-driven intrusion detection in IoT networks.

Keywords: Internet of Things, IoT, Intrusion detection, Feature selection, Feature extraction, Machine learning, Attack classification

Introduction

The Internet of Things (IoT) refers to the technology of connecting everyday life and devices to the internet. IoT is growing and changing quickly, with the goal of linking things like wireless sensors, smart cameras, televisions, and other smart home devices online [1]. The number of Internet-connected IoT devices is rising rapidly, with over 2 billion connected in 2017. Experts predict there will be over 7.5 billion IoT devices generating 73.1 zettabytes of data by 2025 [2]. While IoT devices are becoming widespread and assisting people in many areas, they often have very limited security capabilities.

This is despite the huge growth of IoT and the large amounts of data it creates. In summary, IoT adoption is surging, connecting billions of devices and generating massive data. However, IoT devices typically lack strong security protections even as their use proliferates.

Due to the security limitations of IoT devices, it is crucial to create network intrusion detection systems (NIDS) that can quickly and dependably detect and prevent attacks on IoT networks [3]. For this purpose, many machine learning techniques have been developed for intrusion detection in IoT, along with public datasets of network traffic [4]. However, these datasets frequently contain numerous irrelevant or redundant features, which negatively impacts the complexity and accuracy of machine learning models [5]. A common approach to develop efficient NIDS is through feature reduction, which decreases the dimensionality of network traffic data fed into the machine learning model. This helps lower computational costs and latency while enhancing model generalization.

Two of the most common are feature selection and feature extraction, which help address the issues caused by excessive features. Feature selection selects a subset of the most informative features from the original set [6]. It reduces dimensionality while retaining the semantic interpretability of the selected features. In contrast, feature extraction transforms the original features into a new low-dimensional space via mathematical projection [7]. While it can effectively reduce dimensionality, however, the extracted features lose intuitive meanings. In the realm of IoT security, feature selection enables the creation of lightweight and efficient IDS by judiciously choosing a subset of the most relevant original features. On the other hand, feature extraction techniques offer a valuable means to transform and distill the essence of the original feature set, reducing overall data dimensionality while retaining critical information. By optimizing the efficiency and interpretability of intrusion detection models, both feature selection and feature extraction become indispensable tools for enhancing the cybersecurity posture of IoT ecosystems, ensuring effective threat detection in a manner tailored to the limitations and intricacies of IoT devices and networks.

While existing works have focused on using either feature selection, feature extraction or hybrid method of the two, to improve certain performance metrics for NIDS [8, 9], there remains a research gap in comprehensively comparing these two methods, especially on modern IoT datasets [10]. Very few studies have evaluated the trade-offs between detection accuracy and computational complexity under the same experimental settings. However, such a comparison is essential to provide guidelines for choosing the appropriate feature reduction technique based on the IoT system constraints and intrusion detection requirements.

Therefore, this research aims to conduct an in-depth investigation of feature selection and feature extraction for building lightweight NIDS tailored to IoT environments. We focus on comparing the two techniques because they take contrasting approaches to reducing dimensionality, and may have different advantages and limitations in the context of IoT-based NIDS [11]. The findings can provide data-driven insights to guide the selection of feature reduction methods for optimal efficiency and detection performance in IoT network protection systems. In summary, our work addresses the gap in comparative studies on feature reduction techniques for machine learning-driven NIDS on IoT data. By benchmarking feature selection and extraction head-to-head, we derive valuable guidelines for striking the right balance between detection accuracy and complexity in IoT environments.

This comparative study reveals that feature selection and feature extraction have different strengths and weaknesses for building lightweight NIDS on IoT data. Our experiments demonstrate that when a substantial number of features are reduced, feature selection generally achieves higher detection accuracy, demanding less training and inference time. Conversely, as the number of features decreases, feature extraction excels over feature selection. Additionally, examining the F1-scores for different attack classes under various feature quantities using various machine learning classifiers provides a deeper insight into the detection capabilities of both methods. This analysis reveals that while feature extraction shows less sensitivity to changes in the number of reduced features, it also demonstrates the ability to detect a wider array of attack types compared to feature selection. Moreover, both methods favorite Decision Tree classifier considering both classification metrics and run time performance, which is more suitable for NIDS in IoT network. Based on these observations, we present a detailed theoretical guide, elaborated in Table 20 within "Result verification statistically" section, to aid in the selection of the most appropriate intrusion detection method for distinct scenarios.

The key contributions in this paper are provided as follows.

1) A comprehensive performance evaluation between feature selection and feature extraction, involving performance metrics and run-time using the IoT data set, is conducted and evaluated.
2) The 3-phase machine learning pipeline framework, involving data preprocessing, feature reduction, and classification with multiple machine learning classifiers, is created for performance evaluation.
3) The NIDS for IoT is tested using public IoT datasets, named Network TON-IoT [10], to build models and compare performance between two feature reduction methods.

The subsequent sections are structured as follows: "Related works" section explores previous studies linked to this research, "Methodology" section explains the proposed methodology, "Experimental setup and analysis" section details the experimental setup and analysis, "Result and analysis" section displays the outcomes and discussions of two feature reduction techniques, and lastly, "Conclusion" section concludes this paper.

Related works

In this section, related studies on NIDSs that were implemented using feature reduction methods are discussed.

In the realm of NIDS, there has been widespread use of feature selection to reduce the complexity of the original traffic data. Many studies employ filter-based feature selection method to select the discriminate feature towards target class. For instance, in study [8], Mutual Information (MI)-based approach was proposed to select the features for NIDS, the study compared both linear and non-linear, specifically correlation-based and MI-based feature selection techniques, while MI-based outperform correlation-based approach on accuracy of attack detection. After that, Ambusaidi et al. [12] introduced a feature selection algorithm that utilized MI in combination with an variant support vector machine classifier. This approach exhibited enhanced accuracy and decreased model complexity compared to prior methods, on datasets such as KDD Cup 99, NSL-KDD [10] and Kyoto 2006+ [13].

In study [14], the authors conducted an analysis of a dataset named UNSW-NB15 [15] for NIDS. The filter-based feature reduction technique using machine learning algorithm such as XGBoost algorithm was applied to select features. In the same way, Disha and Waheed [16] designed feature ranking based on Gini Impurity by Random Forest (RF) to analyze the classification performance for NIDS using the latest TON-IoT dataset, while did not consider too much on computational cost for feature reduction process. However, most of the datasets as networking dataset are outdated as the benchmark data sets to evaluate classification models in NIDS for IoT security.

Furthermore, many studies use wrapper-based feature selection to find out the best feature subsets to improve the classification performance. Shafiq et al. [17] introduced a feature selection method called CorrAUC and a wrapper-based FS algorithm that employs the area under the curve (AUC) metric to choose effective features for machine learning (ML) algorithms. The method was tested on the Bot-IoT dataset [18] with four ML algorithms and the approach effectively selected informative features, however, it had lower precision for certain attacks like keylogging attack.

In addition, various techniques employing heuristic optimization algorithms, such as genetic algorithms (GA) as a search strategy to identify optimal feature subsets are detailed in [19–21]. These methods demonstrated lower false alarm rates compared to baseline approaches, using datasets like UNSW-NB15 and KDD99. In study [22], The researchers utilized the Pigeon Inspired Optimizer (PIO) for the feature selection process, binarizing the continuous pigeon inspired optimizer and contrasting it with the conventional approach for binarizing continuous swarm intelligent algorithms. The evaluation was conducted on datasets including KDDCUP99, NLS-KDD, and UNSW-NB15, showcasing outcomes that demonstrated a high detection rate and accuracy while minimizing false alarms. In addition, some studies designed lightweight models to meet the characteristic of IoT network, Liu et al. [23] proposed Particle Swam Optimization (PSO) with one-class Support Vector Machine (SVM) [24] optimized PSO for feature selection with light GBM to build lightweight models for detecting attack. However, it is worth noting that these feature selection strategies often come at a high computational cost, especially when relying on GA, PSO, or machine learning-based classifiers, as a result, which have negative impact on resource-constraint IoT system and networks.

Moreover, many studies employed hybrid feature selection methods to improve the performance of the attack classifiers while reducing overfitting in model training task. In study [25], the authors utilized association rule mining and central attribute values outperformed NSLKDD when tested on the UNSW-NB15 dataset. In addition, some studies employed ensemble feature selection techniques to find out the significant features, for example, Moustafa et al. [26] employed an ensemble Intrusion detection technique, which combined DT, ANN and NB as the base learners to learn the optimal features from statistic flow features, while Leevy et al. [27] employed information gain, information gain ratio, and Chi-squared (Chi^2) feature ranking techniques for feature selection. However, the cost for computation is overlooked with the purpose of improving the performance metrics.

As a response to this challenge, researchers investigated a correlation-based feature selection method that offers a more computationally efficient solution for NIDS, considering the correlation among features [6]. This approach was initially applied to the KDD99 and UNSW-NB15 datasets in [28]. More recently, Moustafa et al. [26] proposed correlation-based method which was improved for multivariate correlation-based network anomaly detection systems, moreover, Gavel et al. [29] employed correlation-based fitness function using the ant lion optimization to select features using AWID dataset for wireless network. Zhou et al. [30] chose the optimal features by removing the redundant features and selecting the most informative features based on the threshold of correlation. These works lead to a substantial improvement in NIDS accuracy, albeit with increased complexity. In light of the need for real-time and low-latency attack detection solutions, this study will place greater emphasis on the correlation-based feature selection method.

Unlike feature selection, which maintains a subset of initial features in Network Intrusion Detection Systems (NIDS), feature extraction focuses on condensing the original features into a lower-dimensional vector while preserving much of the data and applied in various research domains. In the research domain of image processing and pattern recognition, feature extraction involves transforming raw data, such as images, into a reduced and more meaningful representation [31]. The primary goal is to capture essential information that is relevant for subsequent analysis, classification, or recognition tasks. For example, Miseikis et al. [32] employed a multi-objective convolutional neural network to extract features, identify and precisely localize the robot in 2D camera images, allowing flexibility in camera movement and providing accurate 3D position estimates for the robot base and joints. Aggarwal [33] explored the use of the Grey-level Co-occurrence Matrix (GLCM) feature extractor in classifying brain tumor MRI images with a random forest classifier. The results indicate that GLCM features with optimal parameters can achieve promising accuracy in capturing significant texture components. Various methods, such as principal component analysis (PCA), linear discriminant analysis (LDA), and autoencoders (AE) based on neural networks, have been utilized for reducing dimensions in NIDS.

For example, in [34], the KDD99 dataset's dimensionality was greatly reduced by PCA, improving NIDS performance and accuracy while handling attack classification via support vector machines. Various PCA variants, such as hierarchical PCA neural networks using 1998 DARPA dataset [35] and kernel PCA with genetic algorithms [36] have been adopted for intrusion detection to improve precision for less common attacks. PCA is also employed to recent network traffic datasets like UNSW-NB15 and CICIDS2017 can be found in [37, 38]. Additionally, LDA has been utilized as a feature reduction method in NIDS to notably decrease computational complexity, as seen in [39]. In [40, 41] the combination of PCA and LDA were employed to build a two-layer dimension reduction approach, effectively reducing dimensionality, and detecting low-frequency malicious activities over the NSLKDD dataset.

To improve efficiency of feature extraction in NIDS, various research works have applied AE-based neural networks. In particular, Yan and Han [7] introduced a stacked sparse AE approach to build non-linear mapping of high-dimensional to low-dimensional data over the NSLKDD dataset. Khan et al. [42] employed a deep stacked AE to reduce the number of features for both binary and multiclass classification, achieving higher accuracy than previous methods. Several AE-based networks on long short-term memory (LSTM), including variational LSTM [43] and bidirectional LSTM [44], have been developed for dimensionality reduction in NIDS, addressing imbalances and high-dimensional problems effectively. However, it's worth noting that AE-based methods, derived from deep neural networks, entail higher computational costs in both training and testing compared to statistical-based PCA and LDA algorithm.

To mitigate the computational costs issue, a network pruning algorithm was recently proposed to build lightweight detection model in [45] to significantly reduce the complexity of AE structures for feature extraction in NIDS, using UNSW-NB15 and CICIDS data sets. Moreover, in [46], a network design integrates an autoencoder (AE) network using convolutional and recurrent neural networks to extract spatial and temporal features without human intervention.

Since there is a wide range of studies that employed various feature reduction or dimensionality reduction techniques, which can be classified into two methods, namely feature selection and feature extraction, to build lightweight detection models for NIDS. However, few studies conduct comprehensive comparison for the performance and efficiency between the two methods, particularly for IoT data. For example, Aminanto et al. [9] combined AE-based feature extraction and supervised machine learning feature selection to learning representations of the original features, without performance comparison between them, while [47] only conducted comparison for two methods using traditional networking data set UNSW-NB15.

It's important to highlight that most of the previously mentioned studies have concentrated on enhancing either the accuracy of detection or reducing the computational complexity of Network Intrusion Detection Systems (NIDS). They accomplished this by utilizing machine learning classifications and feature engineering methods such as FS and FE to minimize data complexity. Nonetheless, the existing literature lacks a comprehensive comparison between these two feature reduction methods with current datasets in IoT networks. Our study endeavors to fill this gap.

In particular, we initiate the creation of a machine learning-driven NIDS framework utilizing diverse IoT data, emphasizing the feature reduction evaluation phase. Within this context, we identify feature selection through the correlation matrix and feature extraction using PCA as promising approaches for practical low-latency NIDS operations. We then perform an extensive assessment using the contemporary TON-IoT dataset derived from a heterogenous IoT network, comparing performance measures for detection. This includes accuracy, precision, recall, F1-score, and runtime intricacies such as feature reduction time, model training time, and inference time for these methodologies. Our evaluation encompasses both binary and multiclass classifications while maintaining consistency in the quantity of selected or extracted features.

Methodology

In this section, we put FS or FE technique as module factor into the pipeline of machine learning-based network intrusion detection system (NIDS) respectively, according to the final performance metrics of the classification models. Here is the framework of the methodology according to Fig. 1, which can be divided into three phases, data pre-processing, feature reduction, and classification. A detailed explanation of the three workflow of the proposed model is provided as following, particularly for two feature reduction methods.

(1) Data preprocessing

During this phase, the data is processed by cleansing, partitioning, and normalization to standardize the data format. The dataset is divided into two sets, training for feature reduction and testing for final model prediction. A detailed description is presented in "Phase 1 data preprocessing" section.

(2) Feature reduction

This critical stage employs FS or FE techniques to identify the most crucial attributes, thereby reducing data dimensionality. The transformed data through both methods is then utilized in subsequent classification tasks. "Phase 2 feature reduction" section offers an in-depth description of feature reduction methods.

(3) Classification modeling

Various machine learning models, involving Decision Tree, Random Forest, k-Nearest Neighbors, Naive Bayes, and Multiple Layer Perception, are employed to validate the impact of the two feature reduction methods. These models perform binary and multiple classifications, offering a comprehensive comparison based on multiple performance metrics.

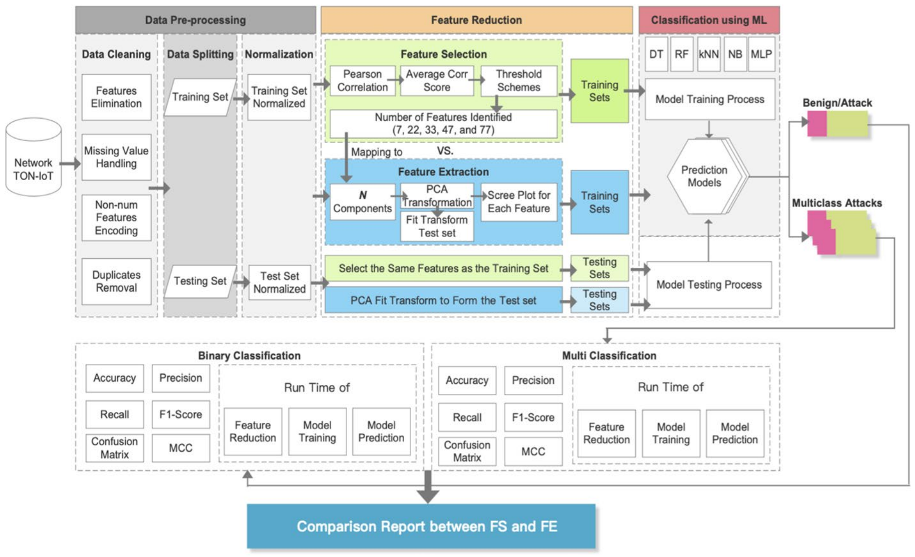


Fig. 1 Framework of proposed NIDS for comparison of feature reduction methods

Dataset

Below is the key information about the TON-IoT Network dataset, which will be employed in our experiments detailed in "Experimental setup and analysis" section. Subsequently, a comprehensive discussion on data preprocessing for this dataset will be provided.

TON-IoT dataset was generated from heterogeneous data sources collected from Telemetry datasets of IoT and IIoT sensors, operating systems datasets of Windows as well as Ubuntu network traffic datasets. It was first introduced in [48], and this dataset comprises 22,339,021 instances of data and includes two target classes: the "label" class, containing normal and attack data, and another class with ten categories—normal and nine attack types, such as Backdoor, DDoS, DoS, Injection, Password, Ransomware, Scanning, XSS, and MITM. There are six feature groups: Connection, Statistical, DNS, SSL, HTTP, Violation, and Labeling, holding a total of 45 features in the original data. However, in this research, our data analysis involves the "Train_Test_Network.csv" dataset, comprising both training and testing sets, totaling 461,043 records. Table 1 displays the distribution of labels for the binary class and types for the multiple-class, while Table 2 shows the dataset's respective features.

Phase 1 data preprocessing

Data preprocessing refers to the process of transforming raw data into a clean, consistent, and meaningful format that can be used for analysis. It plays a vital role in ensuring the quality and suitability of data for machine learning based classification models in IoT security [49]. Thus, to achieve accurate and reliable results, proper data preprocessing of IoT datasets is crucial. As described in the methodology framework, feature elimination, missing value handling, duplicates removal, non-numerical features encoding, and normalization, after that, data splitting is implemented to split the original data into training set and test set, in which, the training set is used for following normalization, feature reduction, and model training process, while the test set will be set aside for final model prediction for both binary and multi-class classification.

Table 1 Classes description in network TON_IoT

No.	Type of classes	Amount	No.	Type of sub-classes	Amount
1	Normal	300,000	1	Normal	300,000
2	Attack	161,043	2	Backdoor	20,000
			3	ddos	20,000
			4	dos	20,000
			5	Injection	20,000
			6	Password	20,000
			7	Ransomware	20,000
			8	Scanning	20,000
			9	xss	20,000
			10	mitm	1043

Table 2 Features description in network TON_IoT

No.	Feature	Description	No.	Feature	Description
1	ts	Timestamp of connection between flow identifiers	24	dns_rejected	DNS rejection, where the DNS queries are rejected by the server
2	src_ip	Source IP addresses which originate endpoints' IP addresses	25	ssl_version	SSL version which is offered by the server
3	Src_port	Source ports which Originate endpoint's TCP/UDP ports	26	ssl_cipher	SSL cipher suite which the server chose
4	Dst_ip	Destination IP addresses which respond to end-point's IP addresses	27	ssl_resumed	SSL flag indicates the session that can be used to initiate new connections, where T refers to the SSL connection is initiated
5	Dst_port	Destination ports which respond to endpoint's TCP/UDP ports	28	ssl_established	SSL flag indicates establish-ing connections between two parties, where T refers to establishing the connection
6	proto	Transport layer protocols of flow connections	29	ssl_subject	Subject of the X.509 cert offered by the server
7	Service	Dynamically detected protocols, such as DNS, HTTP and SSL	30	ssl_issuer	Trusted owner/originator of SLL and digital certificate (certificate authority)
8	Duration	The time of the packet connections, which is estimated by subtracting 'time of the last packet seen' and 'time of the first packet seen'	31	http_trans_depth	Pipelined depth into the HTTP connection
9	src_bytes	Source bytes which are originated from payload bytes of TCP sequence number	32	http_method	HTTP request methods such as GET, POST and HEAD
10	dst_bytes	Destination bytes which are responded payload bytes from TCP sequence numbers	33	http_uri	URIs used in the HTTP request
11	conn_state	Various connection states, such as S0 (connection without replay), S1 (con-nection established), and REJ (connection attempt rejected)	34	http_version	The HTTP versions utilized such as V1.1
12	missed_bytes	Number of missing bytes in content gaps	35	http_request_body_len	Actual uncompressed content sizes of the data transferred from the HTTP client
13	src_pkts	Number of original packets which is estimated from source systems	36	http_response_body_len	Actual uncompressed content sizes of the data transferred from the HTTP server
14	src_ip_bytes	Number of original IP bytes which is the total length of IP header field of source systems	37	http_status_code	Status codes returned by the HTTP server
15	dst_pkts	Number of destination packets which is estimated from destination systems	38	http_user_agent	Values of the UserAgent header in the HTTP protocol

Table 2 (continued)

No.	Feature	Description	No.	Feature	Description
16	dst_ip_bytes	Number of destination IP bytes which is the total length of IP header field of destination systems	39	http_orig_mime_types	Ordered vectors of mime types from source system in the HTTP protocol
17	dns_query	Domain name subjects of the DNS queries	40	http_resp_mime _types	Ordered vectors of mime types from destination system in the HTTP protocol
18	dns_qclass	Values which specifie the DNS query classes	41	weird_name	Names of anomalies/violations related to protocols that happened
19	dns_qtype	Value which specifies the DNS query types	42	weird_addl	Additional information is associated to protocol anomalies/violations
20	dns_rcode	Response code values in the DNS responses	43	weird_notice	It indicates if the violation/anomaly was turned into a notice
21	dns_AA	Authoritative answers of DNS, where T denotes server is authoritative for query	44	Label	Tag normal and attack records, where 0 indicates normal and 1 indicates attacks
22	dns_RD	Recursion desired of DNS, where T denotes request recursive lookup of query	45	Type	Tag attack categories, such as normal, DoS, DDoS and backdoor attacks, and normal records
23	dns_RA	Recursion available of DNS, where T denotes server supports recursive queries			

Feature elimination

To maintain the generalization of the models that can be used for real-scenario classification models in IoT networks, the features that represent the identifiers of the test environment in which the data was generated are eliminated. The "ts" feature represents the timestamp of each connection, while 'src_ip', 'src_port', 'dst_ip', 'dst_port' stands for the identifier of each instance, all these features are not significant as the predictors for following model training [50], therefore, after eliminating the unnecessary features in this stage, 38 features are left in the data set, with the exception of the two labels.

Missing value handling

Since all the "–" values among the features means not available from the perspective of networking domain knowledge, for example, the connection feature named "service" that has "–" value, which means the instance does not have the service value. Similarly, the instances that have "–" value in DNS features means that the instances are not DNS-capable instances. In the same way, the remaining features involving SSL, HTTP and Violation features that contain "–" value, means these instances do not support the SSL, HTTP and Violation capability. Thus, we replace it with the value "n/a," which means it is not available for this feature, and will create a corresponding new feature, named "<feature_name>_n/a," as detailed in "Non-numerical features encoding" step.

Duplicates removal

After investigate the dataset, there are 11,071 rows duplicated in the data set, thus, we need consider the mechanism to handle. Because duplicate instances cannot contribute to meaningful data to the model building process, we directly drop the duplicate instances. Here we remove the duplicated instances and leave the unique ones in the dataset. Now the remaining the dataset of 449,972 rows with unique instances is generated.

Non-numerical features encoding

Since non-numerical data can be used for model training process, while part of the original dataset of Network TON-IoT has 38 features, including 15 numerical features and 23 non-numerical features. Since there are many categorical features in the dataset, so we need to convert the non-numerical features into numerical ones, so that the followed reduction and machine learning algorithms can process the data. Label encoding and one-hot encoding are methods for handling categorical variables in machine learning. The choice between them depends on the specific dataset and the ML algorithm we use.

Label encoding is simpler and more space-efficient, but it may introduce an arbitrary order to categorical values. One-hot encoding avoids this issue by creating binary columns for each category, but it can lead to high-dimensional data [51]. In our work, we implement different encoding scheme considering the characteristics of various features in the dataset.

We employ the one-hot encoding method for the connection features "proto," "service," and "conn_state" because they all have distinct and finite values. The new features will be encoded as "proto_icmp," "proto_tcp," and "proto_udp" features with binary values like 0 or 1. For instance, the "proto" feature has the values "icmp," "tcp," and "udp." The only difference is that "service_n/a" and "conn_state_n/a" will be generated since the original features contain "n/a," which is not available in the original features. Otherwise, the same scheme will be applied to features "service" and "conn_state."

Regarding the DNS features, such "dns_query," one-hot encoding or directly applying the label may not be the best course of action due to the feature's numerous possible values. As a result, we employ a binary encoding approach to classify this feature as either DNS request available or not, indicating whether or not the instance has DNS requests. Regarding the characteristics "dns_AA", "dns_RD", "dns_RA", and "dns_rejected", we convert the non-numerical features into numerical ones using a one-hot encoder.

The non-numerical SSL features are "ssl_version," "ssl_cipher," "ssl_resumed," "ssl_established," "ssl_subject," and "ssl_issuer." Binary encoding is used in these features because the majority of them have SSL functionality disabled. One-hot encoder is used to convert the "ssl_resumed" and "ssl_established" features into numerical ones.

Regarding HTTP features, the non-numerical http features are "http_trans_depth", "http_method", "http_uri", "http_version", "http_user_agent", "http_orig_mime_types", and "http_resp_mime_types". For "http_uri", "http_user_agent", "http_orig_mime_types", and "http_resp_mime_types", binary encoding will be applied, while one-hot encoding will be applied to "http_trans_depth", "http_method", and "http_version".

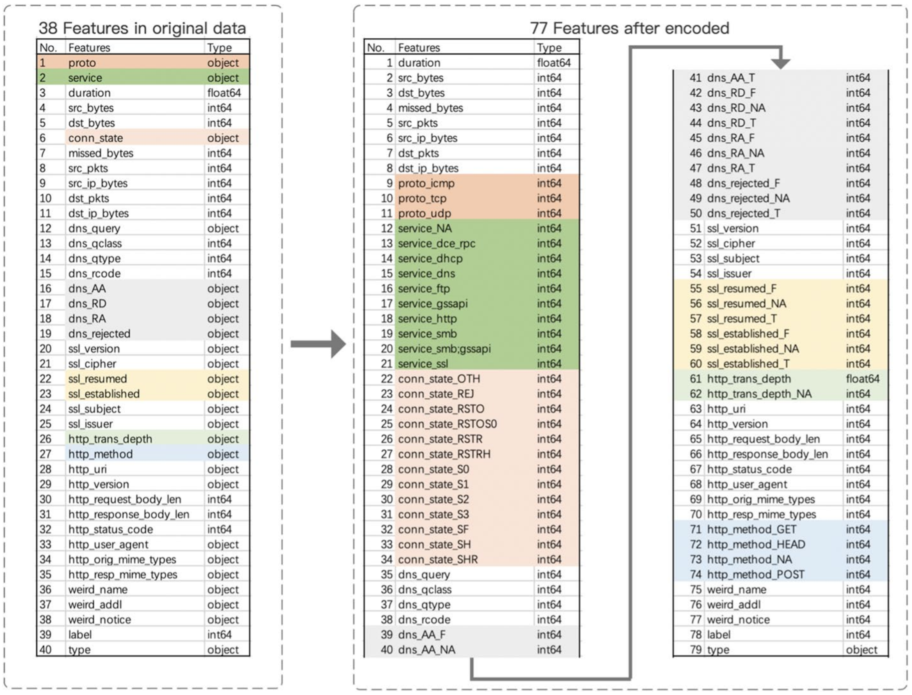
Regarding the weird features, the non-numerical weird features are “weird_name” “weird_addl” and “weird_notice”. For “weird_name” binary encoding will be used, and for “weird_addl” and “weird_notice” one-hot encoding will be used.

Consequently, as presented in Fig. 2, the number of features will increase from the initial 38 to 77 in this step once the aforementioned features are encoded, many of which are not particularly useful in classifying attacks. To reduce the complexity of machine learning models during the classification stage, it is therefore required to condense such a vast number of attributes into a small number. Different encoding schemes are used for the non-numerical features above based on the features’ qualities in the dataset, enabling the transformed data to proceed to the next phase.

Data splitting

Data splitting is to split the original data into two sets, one is training set which is used for model training, while the other sets named test sets is used for model test, or the final performance evaluation for the trained model. However, to avoid data leakage in the following step of data transformation, such as normalization and feature reduction, and following machine learning process, data splitting was implemented before that [51].

Moreover, in order to verify the effectiveness of the trained model, the proportion of the classes of the test data set keeps nearly the same class distribution as the training set to simulate the real scenario of IoT networks. Thus, we use stratified splitting scheme to split the dataset into training and test data with the proportion of 80:20, in which the 80% of the dataset will be used for model training to improve the performance of the final model, while the remaining percent will be used for model evaluation. As a result of the data splitting, the



38 Features in original data

No.	Features	Type
1	proto	object
2	service	object
3	duration	float64
4	src_bytes	int64
5	dst_bytes	int64
6	conn_state	object
7	missed_bytes	int64
8	src_pkts	int64
9	src_ip_bytes	int64
10	dst_pkts	int64
11	dst_ip_bytes	int64
12	dns_query	object
13	dns_qclass	int64
14	dns_qtype	int64
15	dns_rcode	int64
16	dns_AA	object
17	dns_RD	object
18	dns_RA	object
19	dns_rejected	object
20	ssl_version	object
21	ssl_cipher	object
22	ssl_resumed	object
23	ssl_established	object
24	ssl_subject	object
25	ssl_issuer	object
26	http_trans_depth	object
27	http_method	object
28	http_uri	object
29	http_version	object
30	http_request_body_len	int64
31	http_response_body_len	int64
32	http_status_code	int64
33	http_user_agent	object
34	http_orig_mime_types	object
35	http_resp_mime_types	object
36	weird_name	object
37	weird_addl	object
38	weird_notice	object
39	label	int64
40	type	object

77 Features after encoded

No.	Features	Type
1	duration	float64
2	src_bytes	int64
3	dst_bytes	int64
4	missed_bytes	int64
5	src_pkts	int64
6	src_ip_bytes	int64
7	dst_pkts	int64
8	dst_ip_bytes	int64
9	proto_icmp	int64
10	proto_tcp	int64
11	proto_udp	int64
12	service_NA	int64
13	service_dce_rpc	int64
14	service_dhcp	int64
15	service_dns	int64
16	service_ftp	int64
17	service_gssapi	int64
18	service_http	int64
19	service_smb	int64
20	service_smb;gssapi	int64
21	service_ssl	int64
22	conn_state_OTH	int64
23	conn_state_REJ	int64
24	conn_state_RSTO	int64
25	conn_state_RSTOS0	int64
26	conn_state_RSTR	int64
27	conn_state_RSTRH	int64
28	conn_state_S0	int64
29	conn_state_S1	int64
30	conn_state_S2	int64
31	conn_state_S3	int64
32	conn_state_SF	int64
33	conn_state_SH	int64
34	conn_state_SHR	int64
35	dns_query	int64
36	dns_qclass	int64
37	dns_qtype	int64
38	dns_rcode	int64
39	dns_AA_F	int64
40	dns_AA_NA	int64
41	dns_AA_T	int64
42	dns_RD_F	int64
43	dns_RD_NA	int64
44	dns_RD_T	int64
45	dns_RA_F	int64
46	dns_RA_NA	int64
47	dns_RA_T	int64
48	dns_rejected_F	int64
49	dns_rejected_NA	int64
50	dns_rejected_T	int64
51	ssl_version	int64
52	ssl_cipher	int64
53	ssl_subject	int64
54	ssl_issuer	int64
55	ssl_resumed_F	int64
56	ssl_resumed_NA	int64
57	ssl_resumed_T	int64
58	ssl_established_F	int64
59	ssl_established_NA	int64
60	ssl_established_T	int64
61	http_trans_depth	float64
62	http_trans_depth_NA	int64
63	http_uri	int64
64	http_version	int64
65	http_request_body_len	int64
66	http_response_body_len	int64
67	http_status_code	int64
68	http_user_agent	int64
69	http_orig_mime_types	int64
70	http_resp_mime_types	int64
71	http_method_GET	int64
72	http_method_HEAD	int64
73	http_method_NA	int64
74	http_method_POST	int64
75	weird_name	int64
76	weird_addl	int64
77	weird_notice	int64
78	label	int64
79	type	object

Fig. 2 The features of network TON_IoT before and after numerically encoded

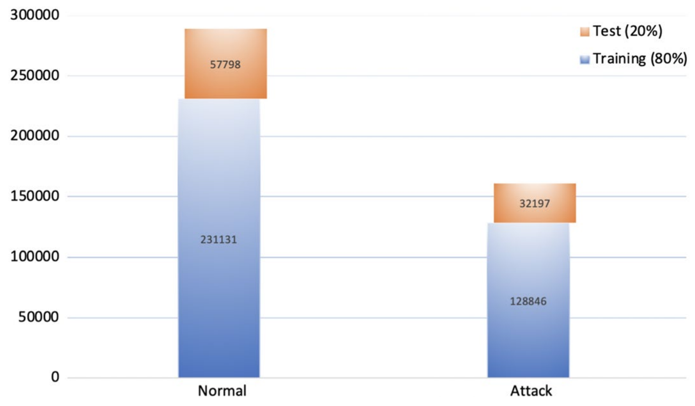


Fig. 3 Proportions of the normal/attack classes in training and test set of TON-IoT

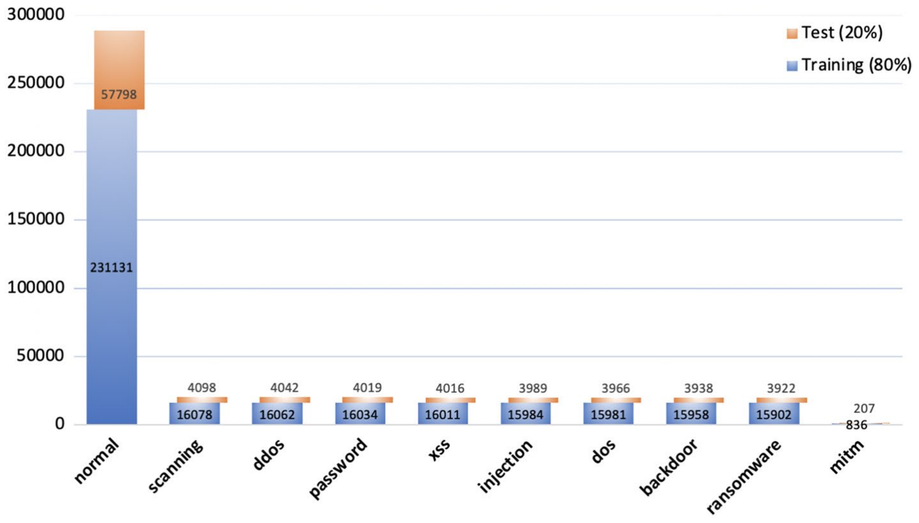


Fig. 4 Proportions of the 10 classes in training and test set of TON-IoT

distribution and the specific number of instances of the normal/attack class and 10 classes for binary classification and multi-classification purposes, respectively, are shown in Figs. 3, 4.

Normalization

Normalization is used to keep the scale of the feature without bias to the features with large values. In machine learning, two commonly used feature scaling techniques are normalization and standardization. The studies [25, 47] used normalization technique to scale the features, thus, we use min–max scaling to normalize the data. The data in this experiment were normalized between the range of 0 and 1 using min–max scaling. The normalization formula is shown as Eq. (1):

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{1}$$

where X is the original value of the data point, $X_{normalized}$ is the normalized value of the data point. X_{min} is the minimum value of the variable, while X_{max} is the maximum value of the variable of the data set.

As demonstrated in Algorithm 1, we developed our preprocessing technique based on the preceding procedures discussed above in the phase 1 data pro-processing.

Algorithm 1 Data preprocessing in phase 1

```
Input: Original Dataset (461,043 instances, 44 features)
Output: Numeric training and test dataset with no duplicate and missing data
        (449,972 instances with 77 features, in which, training set: 359,977 and test set: 89,995)
1. Array ← RawDataset
2. In [Array] remove 'label', 'type' label
3. In [Array] remove 'ts', 'src_ip', 'src_port', 'dst_ip', 'dst_port' features
4. In [Array] replace '-' with "n/a"
5. In [Array] remove 11071 duplicates
6. In [Array] One-hot encoding on 'proto', 'service', 'conn_state', 'dns_AA', 'dns_RD', 'dns_RA',
'dns_rejected', 'ssl_resumed', 'ssl_established', 'http_trans_depth', 'http_method', 'http_version',
'weird_addl', 'weird_notice' features
7. In [Array] Binary encoding on 'dns_query', 'ssl_version', 'ssl_cipher', 'ssl_resumed',
'ssl_established', 'ssl_subject', 'ssl_issuer', 'http_uri', 'http_user_agent', 'http_orig_mime_types',
'http_resp_mime_types', 'weird_name'
8. X_train, X_test, y_train, y_test ←Train_test_split [Array]
9. X_train_normalized, X_test_normalized ← Min-max scaling X_train, X_test
10. X_train_p1, X_test_p1 ←X_train_normalized, X_test_normalized
```

Phase 2 feature reduction

Feature selection

There are a series of feature selection techniques implemented in NIDS for IoT security, such as Gini-impurity [3], Chi-square [4], Information Gain [5], Mutual Information [25] and Feature Correlation [29, 33, 50]. In this work, we focus on employing feature correlation to pick informative features based on the range of the given threshold because it has been found to attain competitive detection accuracy and complexity when compared to other selection equivalents. The correlation between each feature and the target variable is typically calculated in a correlation-based feature selection approach. In this methodology, we implement correlation-based feature selection based on Pearson correlation coefficient technique [46], by selecting features that are not correlated with each other to reduce multicollinearity. The defined correlation score threshold values based on the correlation score are set iteratively till final full number of features of the dataset, to build classifiers using five machine learning models, which will be explained in phase 3.

The Pearson's correlation coefficient (PCC) represents a straightforward linear correlation approach used to evaluate feature interdependencies. Employing this correlation-based technique, our objective is to select features highly correlated to other features. This selection process relies on the correlation matrix computed from the preprocessed training set following the steps outlined in "Phase 1 data preprocessing" section. To calculate the correlation coefficient between feature f1 and f2, the PCC is derived from the formulated features f1 and f2 as follows:

$$PCC(f_1, f_2) = \frac{cov(f_1, f_2)}{\sigma_{f_1} \times \sigma_{f_2}}, \tag{2}$$

$$\frac{cov(f_1, f_2)}{\sigma_{f_1} \times \sigma_{f_2}} = \frac{\sum_{i=1}^{N} (x_i - M_{f_1})(y_i - M_{f_2})}{\sqrt{\sum_{i=1}^{N} (x_i - M_{f_1})^2} \times \sqrt{\sum_{i=1}^{N} (y_i - M_{f_2})^2}}, \tag{3}$$

$$\boldsymbol{M_{f_1} = 1/N \sum_{i}^{N} x_i}, \tag{4}$$

$$\boldsymbol{M_{f_2} = 1/N \sum_{i}^{N} y_i}, \tag{5}$$

where *cov* is the covariance and σ is the standard deviation, while M_{f_1} and M_{f_2} indicate the means of f1 and f2 respectively.

The Pearson correlation coefficient is a measure of the linear correlation between two variables, ranging from − 1 to 1. A coefficient of 1 indicates a perfect positive correlation, a coefficient of − 1 indicates a perfect negative correlation, and a coefficient of 0 indicates no correlation. The closer the coefficient is to 1 or − 1, the stronger the correlation between the variables.

The average correlation score for each feature with others is then calculated based on algorithm 2. The average correlation scores provide a summary measure of the overall correlation tendency of each feature with respect to all other features in the dataset. A higher average score indicates a feature that, on average, tends to be positively correlated with other features, while a lower average score suggests a feature with weaker or more varied correlations. The assumption of the features to be selected is based on the independence of each feature, in which features with weak or no correlation might be more independent, potentially contributing unique information to the model [46].

Algorithm 2 Calculating average correlation score for each feature in phase 2

```
Input: correlation_matrix (a square matrix containing correlation coefficients between features.)
Output: average_scores (an array containing average correlation scores for each feature.)
1. average_scores = [] # Initialize an empty array to store average correlation scores for each feature.
2. For each feature in correlation_matrix.columns:
3.     average_score = mean(correlation_matrix[feature]) # Calculate the average of correlation coefficients for the current feature.
4.     average_scores.append(average_score) # Append the average score to the array.
5. return average_scores
```

Since the average score of each feature is calculated, the next step is to define the threshold or range of the average score in order to select a different range of features for the benchmark number of features for comparison with feature extraction, followed by model training and validation. The criteria of the range are based on two aspects: one is to select the features from a small size with increasing features until a full set of features is covered; the other aspect is that the number of selected features is based on the threshold or range of average scores based on the overall average scores of the features, which will be detailed and visualized in "Features selected based on correlation thresholds" section.

Furthermore, we only need to consider such feature correlation during the training phase, while during the testing phase, we directly select the selected features from the original high-dimensional training data set to generate the reduced-dimensional test data in Fig. 1's feature reduction module. In contrast, in feature extraction, the PCA technique is applied on both the training and test data sets to reduce dimensionality, which will be considered as the run duration of the feature reduction operation.

Feature extraction

There is a series of feature extraction methods used in IoT security domain, involving PCA [39], LDA [52], and AE [53], while PCA and AE stand out as the mostly used extraction methods applied in NIDS for IoT security. Unlike feature selection, which uses chosen features to map those in the original dataset, these feature extraction techniques use a projection matrix or an Autoencoder-based neural network learned from a training dataset to condense the high-dimensional data into lower-dimensional data. It should be noted that the AE approach often deals with the higher computational complexity associated with deep neural networks (DNN), resulting in greater latency as compared to PCA. Consequently, this study exclusively focuses on the PCA-based feature extraction approach, a choice driven by the imperative need for resource-constraint IoT devices and low latency NIDS to protect IoT network from cyber threats.

Principal Component Analysis (PCA) is a powerful dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional representation, while retaining the most important information. The mechanism of PCA is based on the calculation of eigenvectors and eigenvalues of the data's covariance matrix. The equation that underlies PCA is as follows:

$$C = \frac{1}{N} X * X^T, \tag{6}$$

Here C represents the covariance matrix of the standardized data X consisting of N samples, and X^T is the transpose of X. The matrix C captures the relationship between features in the data.

The projection matrix W_k is a key component in PCA and represents the transformation matrix that projects the original data onto the first k principal components. Each column of W_k corresponds to a principal component.

$$W_k = [V_1, V_2, V_3, \ldots V_k], \tag{7}$$

Here $V_1, V_2, V_3, \ldots V_k$ are the eigenvectors corresponding to the top k eigenvalues of the covariance matrix.

The matrix that performs the projection of the data onto the first ***k*** principal components is often represented as W_k, where each column is a principal component. The projection is as followings:

$$X_{proj} = X * W_k, \tag{8}$$

Here X_{proj} is the projected data onto the first k principal components, W_k is the transpose of the projection matrix.

Then reconstruction of data can be implemented by the from the first ***k*** principal components as:

$$X_{reconstructed} = X_{proj} \times W_k^T, \tag{9}$$

which illustrates how well the original data can be approximated using the reduced set of principal components.

After that, Scree Plot Calculation for each extracted feature is calculated as following:

$$Scree\ plot_i = \frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j}, \tag{10}$$

Here is the i-th eigenvalue of the covariance matrix, $\sum_{j=1}^{d} \lambda_j$ is the sum of all eigenvalues, the Scree Plot values indicate the proportion of total variance explained by each principal component. It shows the explained variance for each principal component, can be calculated by arranging the eigenvalues in decreasing order.

In the training phase, we commence by preparing the dataset for PCA. This step involves fitting the PCA model to the training data, capturing the principal components, and simultaneously transforming the data accordingly. The algorithm operates by first standardizing the data by subtracting the mean and dividing by the standard deviation for each feature to ensure that all features have the same scale, and then computing the covariance matrix C. Eigenvalues and eigenvectors are derived from this matrix C, and by sorting the eigenvalues in descending order, the most significant components are selected. A projection matrix is constructed from these eigenvectors, enabling the data to be projected onto a lower-dimensional subspace. The result is a compact representation of the data, capturing the essential information while reducing dimensionality, making it useful for simplifying machine learning models. For comparison purpose, the defined number of components in PCA are exactly the same as the number of features selected by feature selection for each iteration.

In the testing phase, it's a common practice in machine learning workflows to fit the PCA model on the training data and then use the learned transformation to transform both the training and test datasets. This approach ensures that the same transformation is applied consistently to both sets of data, maintaining the relationship between the principal components [39]. When dealing with test data, the pre-fitted PCA transformation is applied using the transform method, ensuring consistency in the application of learned transformations across both training and test datasets. In addition, different from the run time of feature selection for test set ignored in performance evaluation, that of PCA for test data is calculated as the whole run time in feature reduction in phase 3.

As demonstrated in Algorithm 2, we developed and analyzed our feature reduction algorithm based on the preceding procedures discussed above in the phase 2 feature reduction.

Algorithm 3 Feature reduction in phase 2

```
Input: X_train_p1, X_test_p1 (Datasets with overall 449,972 instances 77 features)
Output: X_train_p2, X_test_p2 (Datasets with reduced 9, 22, 33, 47, 77 selected/extracted features)
1. [Array_train] ← X_train_p1, [Array_test] ← X_test_p1
2. Get the feature_correlation_list in descending order based on the average_scores
3. Set threshold ranges: threshold_ranges = [[-0.01, 0.01], [-0.015, 0.015], [-0.02, 0.02], [-0.03,
0.03], [min(average_scores), max(average_scores)]]
4. For each threshold range in threshold_ranges:
5.        get the reduced features based on threshold_ranges: reduced_features = []
6.        get the number of reduced features: No_Feature = length(reduced_features)
7.        get X_train_p2, X_test_p2 with reduced_features applied by feature selection
8.        store the feature selection runtime
9. End for (feature selection)
10. set the same number of PCA components: PCA_components = No_Feature
11. For PCA components in PCA_components:
12.        start a timer to record the feature extraction runtime
13.        apply PCA algorithm on [Array_train]
14.        apply the pre-fitted PCA on [Array_test]
15.        get X_train_p2, X_test_p2 with PCA_components applied by feature extraction
16.        store the feature extraction runtime
17. End for (feature extraction)
```

Phase 3 attack classification

In this phase, for classification tasks, we choose following five classic machine learning models mostly utilized in recent works for NIDS, to implement comprehensive comparison among different classifiers between two feature reduction methods. The specific hyperparameters of the models will be explained in "Experimental setup and analysis" section.

Decision tree (DT)

The decision tree classifier is a widely used machine learning model that aims to create a tree-like structure of decisions based on the features of the data [54]. It works by recursively splitting the data into subsets based on the feature that best separates them, typically using measures like Gini impurity or information gain. The main benefit of decision trees lies in their interpretability and ability to handle both numerical and categorical data. The primary goal of the algorithm for decision tree classification is to use a cost function to find the optimal splits. Decision trees can be applicable to IoT network intrusion detection due to their transparency and ease of understanding which features are critical for detecting attacks. The Gini impurity is used in this work as the splitting criterion, selecting a feature for splitting at each stage of the tree training, as Eq. (11) illustrates:

$$G(D) = \sum_{I=1}^{C} (P(i) + (1 - P(i))), \tag{11}$$

where D is the training dataset, C is a set of class labels, and P (i) is the percentage of samples that have class label I in C. In C, the Gini impurity is 0 when there is just one class.

Random forest (RF)

The random forest classifier is an ensemble method that builds multiple decision trees and combines their outputs to make predictions [55]. Each tree is trained on a random subset of the data with replacement (bootstrap samples) and a random subset of features. This ensemble approach reduces overfitting and improves prediction accuracy. The main benefit of random forests is their robustness and ability to handle high-dimensional data. However, they may not provide as much interpretability as single decision trees. The algorithm behind random forests aggregates the results from multiple decision trees. Random forests can be particularly useful for IoT intrusion detection, as they offer a good balance between accuracy and interpretability. The Gini impurity as presented in Eq. (11) is also used as a split criterion.

k-Nearest neighbors (kNN)

The k-nearest neighbors classifier is a simple instance-based learning model that classifies data points based on the majority class among their k nearest neighbors in feature space [56]. It operates on the assumption that similar data points share the same class label. The main benefit of kNN is its simplicity and effectiveness for non-linear data. However, it can be sensitive to the choice of the distance metric and the value of k. The algorithm calculates distances (e.g., Euclidean distance) between data points to find the k-nearest neighbors. In IoT intrusion detection, kNN can be useful when there is a need to adapt quickly to new attack patterns and anomalies. Due to its widespread usage as a distance metric, the Euclidean Distance was selected. Equation (12) defines the Euclidean Distance Equation as follows:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)}, \quad (12)$$

where the Euclidean distance function between the two samples is represented by *d (x, y), x_i* is the first observation, y_i denotes the second sampling of the data, and n denotes the number of observations.

Naive Bayes (NB)

The Naive Bayes classifier is a probabilistic model based on Bayes' theorem, which calculates the probability of a data point belonging to a specific class given its feature values [57]. It assumes that features are conditionally independent, which is a simplifying, albeit "naive," assumption. Naive Bayes is computationally efficient, particularly for text classification tasks, and can handle high-dimensional data. However, its performance may suffer if the independence assumption is violated. The algorithm calculates class probabilities using Bayes' theorem. In IoT intrusion detection, Naive Bayes can be useful when computational resources are limited and there is a need for quick training and classification. Bayes' theorem is expressed in Eq. (13):

$$P(L|X) = \frac{P(X|L)P(L)}{P(X)}, \quad (13)$$

where $P(L|X)$ is the posterior probability of class L, $P(L)$ is the prior probability, P (X | L) is the likelihood function, and $P(X)$ is the probability, these parameters are estimated using the training set.

Multi-layer perceptron (MLP)

The Multi-Layer Perceptron classifier is a type of artificial neural network that consists of multiple layers of interconnected nodes (neurons) [58]. It can learn complex non-linear relationships in data through a process called backpropagation. MLPs are highly flexible and can approximate any continuous function, making them suitable for various tasks. However, they require a larger amount of data for training and careful tuning of hyperparameters to prevent overfitting. The algorithm involves feedforward and back-propagation steps, where weights are updated to minimize the error between predicted and actual outputs. In IoT intrusion detection, MLPs can be applied when the data is highly complex, and feature engineering has been performed effectively.

As shown in Algorithm 4, we created above five classifiers and trained it with 80% of the dataset samples, and tested it with the remaining 20% samples for performance evaluation between feature selection and feature extraction.

Algorithm 4 Normal/attack classification in phase 3

```
Input: X_train_p2, X_test_p2 (Datasets with reduced 9, 22, 33, 47, 77 selected/extracted features)
Output: Performance metrics and corresponding run time for each case
1. Initialize sets of datasets: X_train_p2, X_test_p2, with reduced features [] and PCA_components
   [] respectively, and y_train, y_test
2. Initialize a list of machine learning models: [DT, RF, kNN, NB, MLP]
3. Initialize classification functions: [binary, multiclass]
4. For each dataset in datasets:
5.     For each model in models:
6.         For each classification in classification functions:
7.             start a timer to record the model training and testing runtime
8.             train the model on the training set
9.             make predictions on the test set
10.            stop the timer to calculate the runtime
11.            calculate performance metrics: accuracy, precision, recall, f1_score
12.            store the performance metrics and runtime
13.        End for (classification tasks)
14.    End for (models)
15. End for (datasets)
```

Experimental setup and analysis

We present an extensive set of experiments examining the performance of the NIDS using feature selection and extraction methods outlined in "Methodology" section. This evaluation involves a variety of machine learning-based classification models. Our comparison entails performance metrics such as accuracy, precision, recall, F1-score, and MCC elaborated in "Performance evaluation" section. Both binary and multiclass classifications are evaluated, and we also explore model training and inference times to evaluate detection method efficiency. Additionally, our thorough comparison of FS and FE methods offers valuable insights into their impact on performance metrics. This includes a comparison with and without feature reduction, providing guidance on selecting the appropriate detection techniques for specific IoT network scenarios.

Experimental setup

Table 3 details the setup of the computing platform, hardware, its operating system, and a variety of software information utilized for constructing the NIDS framework in this work.

Performance evaluation

We analyze the following metrics in order to evaluate the performance comprehensively: accuracy, precision, recall, F1-score, MCC, model training time, and inference time. True positive (TP), true negative (TN), false negative (FN), and false positive (FP) are the four words used to describe these measurements. For the purpose of assessing the capacity for particular class classification, confusion matrices based on the four factors are also proposed in this study. F1-score is determined specifically based on precision and memory as

Table 3 Hardware and software specifications of the implementation environment

Hardware	**Description**
Computing platform	Google colab
Process	2-core Xeon 2.2 GHz
RAM	16 GB
Disk usage	100 GB
Software	**Description**
Operating system	Linux
Machine	x86_64
Python	3.10.12
Other packages	Pandas, Numpy, Scikit-learn, Matplotlib, Scipy, Scikit-plot, and time

follows, which is regarded as a harmonic mean of precision and recall. The Matthews Correlation Coefficient (MCC) is a metric that takes into account true and false positives and negatives, providing a balanced measure of classification performance. It ranges from −1 to 1, where 1 indicates perfect prediction, 0 indicates no better than random chance, and −1 indicates total disagreement between prediction and observation. All the performance metrics Eqs. (14)–(18) are shown as follows:

$$Accuracy = \frac{(TP + TN)}{(TP + FN + FP + TN)}, \tag{14}$$

$$Precision = \frac{TP}{(TP + FP)}, \tag{15}$$

$$Recall = \frac{TP}{(TP + FN)}, \tag{16}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Presicion + Recall}. \tag{17}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}. \tag{18}$$

As to the evaluation of model efficiency, since either feature selection or feature reduction must go through the data-preprocessing stage, so we do not take this step into account, and focus on the evaluation of run time of feature reduction, model training using training set, as well as model prediction using test set. In particular, feature reduction time consists of the amount of time needed to compute (*Feature Calcuation*) and choose the reduced features (*Feature Selection*) until the data set containing the reduced features is updated, which is then fed into machine learning models using the following formula, Eq. (19):

$$Reduction\ Time = Time_{Feature\ Calcuation} + Time_{Feature\ Selection}. \tag{19}$$

The model training time refers to the training time of each classification model $(Model\ Training)$, as following, Eq. (20):

$$Training\ Time = Time_{Model\ Training}. \tag{20}$$

Meanwhile, the inference time means the prediction time of machine learning classifiers $(Model\ Testing)$ in the testing phase, as follow Eq. (21):

$$Inference\ Time = Time_{Model\ Testing}. \tag{21}$$

In particularly, the run time of feature reduction involves transformation of training set and test set using corresponding feature selection or feature extraction algorithm, respectively.

Hyperparameter settings of classifiers

To perform binary and multiclass classification tasks, we employ five machine learning models from the Python Scikit-learn library: Decision Tree (DT), Random Forest (RF), K-nearest Neighbours (kNN), Gaussian Naive Bayes (NB), and Multi-layer Perceptron (MLP). The hyperparameter settings for each model are described in Table 4.

Features selected based on correlation thresholds

Correlation scores matrix is implemented using Pearson correlation algorithm and the result is presented as Fig. 5. It provides a comprehensive view of the relationships and dependencies among different variables. The accompanying heatmap visually enhances the interpretability of these correlations, using a color spectrum to emphasize the strength and direction of the relationships. The Average correlation score of each feature will be calculated in the next step based on the scores in this matrix.

As we can see from Fig. 6, which displays the average correlation score among the features, we manually define the range of the thresholds based on the result of the average score in the figure. In order to cover all ranges of the size of feature subsets, we manually select the features with the least average correlation score until the

Table 4 Hyperparameter settings of each model

Machine learning model	Hyperparameter settings
DT	Criterion: Gini Splitter: best max_depth: None random_state: 42
RF	n_estimators: 100 criterion: Gini max_depth: 5 random_state: 42
kNN	n_neighbors: 3
NB	Default parameters
MLP	hidden_layer_sizes: 100 activation: relu alpha: 0.0001 batch_size: auto learning_rate: constant max_iter: 200

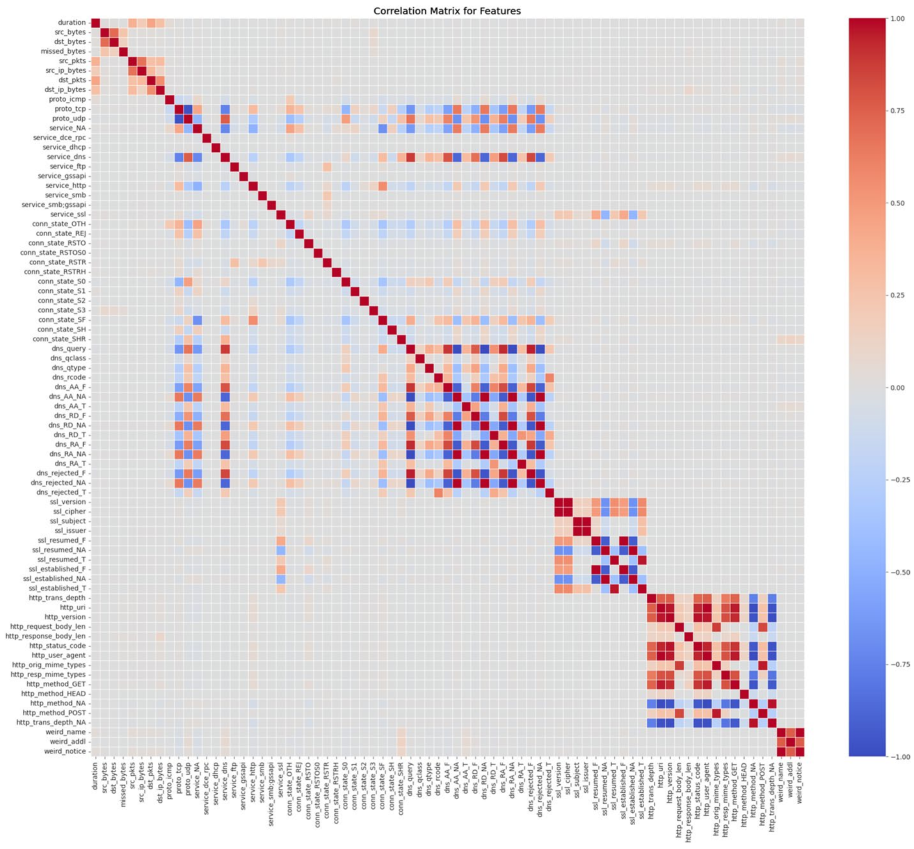


Fig. 5 Pearson correlation score matrix of the features in network ToN-IoT

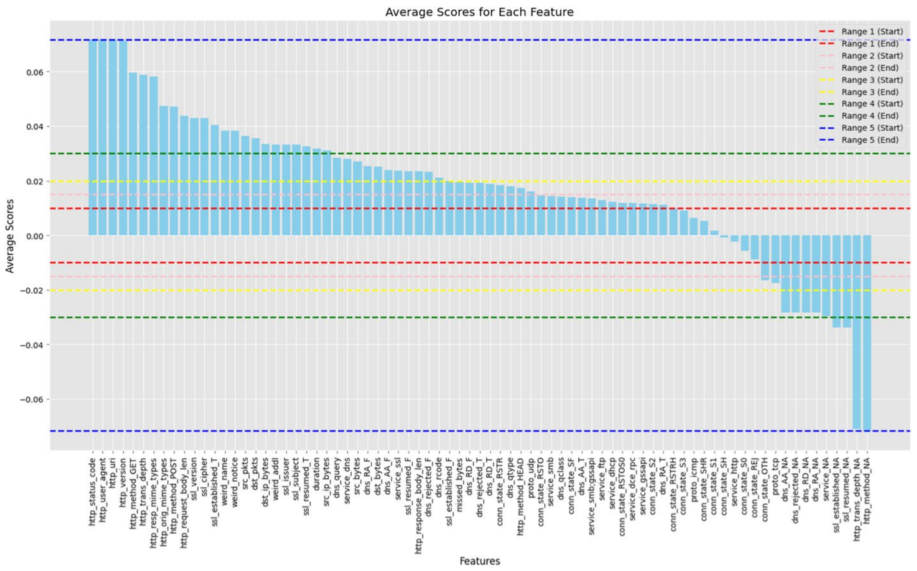


Fig. 6 Average correlation score for each feature in pre-processed TON-IoT

maximum score, ranging from [− 0.01 0.01] to [− 0.015 0.015], [− 0.02 0.02], [− 0.03 0.03] to [− 0.1, 0.1], which is divided based on scores representative of purpose in the figure with the different colors of boundary lines. To be specific, we select the 9 features in [− 0.01 0.01] in order to start the evaluation and comparison in the lightweight model, while selecting 22 in [− 0.015 0.015], 33 in [− 0.02 0.02], 47 in [− 0.03 0.03] for evaluation based on increasing number of features, and selecting 77 full features in [− 0.1, 0.1] for evaluation of the effect between reduced and full features. Thus, we choose the features based on the proposed range of the average correlation score. Following the selection of a certain number of features, the same number of components will be computed and chosen for the performance comparison analysis with feature extraction using PCA.

As a result, in Table 5, we present lists of 9, 22, 33, 47, and 77 (full features) selected features, as well as the corresponding average correlation thresholds used to achieve those numbers of selected features, in order to implement comprehensive comparison for a better understanding of two feature reduction methods.

Features extracted based on PCA

We extract the same number of features as the features selected by feature selection for evaluation and comparison. In our study, we presented the explained variance score of each extracted feature under different schemes (7, 22, 33, 47, and 77 extracted

Table 5 Correlation threshold with the features selected

Correlation threshold	Number of features	Selected features
[− 0.01 0.01]	9	‘conn_state_RSTRH’, ‘conn_state_S3’, ‘proto_icmp’, ‘conn_state_SHR’, ‘conn_state_S1’, ‘conn_state_SH’, ‘service_http’, ‘conn_state_S0’, ‘conn_state_REJ’
[− 0.015 0.015]	22	‘conn_state_RSTO’, ‘service_smb’, ‘dns_qclass’, ‘conn_state_SF’, ‘dns_AA_T’, ‘service_smb;gssapi’, ‘service_ftp’, ‘service_dhcp’, ‘conn_state_RSTOS0’, ‘service_dce_rpc’, ‘service_gssapi’, ‘conn_state_S2’, ‘dns_RA_T’, ‘conn_state_RSTRH’, ‘conn_state_S3’, ‘proto_icmp’, ‘conn_state_SHR’, ‘conn_state_S1’, ‘conn_state_SH’, ‘service_http’, ‘conn_state_S0’, ‘conn_state_REJ’
[− 0.02 0.02]	33	‘ssl_established_F’, ‘missed_bytes’, ‘dns_RD_F’, ‘dns_rejected_T’, ‘dns_RD_T’, ‘conn_state_RSTR’, ‘dns_qtype’, ‘http_method_HEAD’, ‘proto_udp’, ‘conn_state_RSTO’, ‘service_smb’, ‘dns_qclass’, ‘conn_state_SF’, ‘dns_AA_T’, ‘service_smb;gssapi’, ‘service_ftp’, ‘service_dhcp’, ‘conn_state_RSTOS0’, ‘service_dce_rpc’, ‘service_gssapi’, ‘conn_state_S2’, ‘dns_RA_T’, ‘conn_state_RSTRH’, ‘conn_state_S3’, ‘proto_icmp’, ‘conn_state_SHR’, ‘conn_state_S1’, ‘conn_state_SH’, ‘service_http’, ‘conn_state_S0’, ‘conn_state_REJ’, ‘conn_state_OTH’, ‘proto_tcp’
[− 0.03 0.03]	47	‘dns_query’, ‘service_dns’, ‘src_bytes’, ‘dns_RA_F’, ‘dst_bytes’, ‘dns_AA_F’, ‘service_ssl’, ‘ssl_resumed_F’, ‘http_response_body_len’, ‘dns_rejected_F’, ‘dns_rcode’, ‘ssl_established_F’, ‘missed_bytes’, ‘dns_RD_F’, ‘dns_rejected_T’, ‘dns_RD_T’, ‘conn_state_RSTR’, ‘dns_qtype’, ‘http_method_HEAD’, ‘proto_udp’, ‘conn_state_RSTO’, ‘service_smb’, ‘dns_qclass’, ‘conn_state_SF’, ‘dns_AA_T’, ‘service_smb;gssapi’, ‘service_ftp’, ‘service_dhcp’, ‘conn_state_RSTOS0’, ‘service_dce_rpc’, ‘service_gssapi’, ‘conn_state_S2’, ‘dns_RA_T’, ‘conn_state_RSTRH’, ‘conn_state_S3’, ‘proto_icmp’, ‘conn_state_SHR’, ‘conn_state_S1’, ‘conn_state_SH’, ‘service_http’, ‘conn_state_S0’, ‘conn_state_REJ’, ‘conn_state_OTH’, ‘proto_tcp’, ‘dns_AA_NA’, ‘dns_rejected_NA’, ‘dns_RD_NA’, ‘dns_RA_NA’, ‘service_NA’
N/A	77	All the features of the dataset transformed by pre-processing stage

features), and the percentage of the total variance in the original dataset that is captured by each principal component. In other words, it quantifies the amount of information that each principal component retains from the original features.

In Fig. 7, after performing PCA to the dataset, the principal components are ordered by the amount of variance they explain. For each scree plot, the first principal component explains the most variance, the second explains the second most, and so on, with the bar chart presented for 7, 22, 33, 47, and 77 extracted features, respectively, which is the same as the number of the features selected for evaluation and comparison purposes. In addition, the explained variance is expressed as a ratio or percentage of the total variance. Thus, higher explained variance ratios indicate more significant contributions of principal components in capturing the dataset's variability. It helps in making informed decisions about the number of components to retain for following tasks, such as model training and validation.

Result and analysis

Binary classification

Initially, we explore the performance and runtime of feature selection and extraction methods in binary classification, presented in Tables 6, 7, 8, 9, and 10. For every feature number scheme, five iterations are carried out in order to get an affirmative conclusion. The average result is then computed using the outcomes of each iteration. These tables showcase the performance metrics and times for 9, 22, 33, 47, and 77 features (full features) selected or extracted, respectively. The highlighted values, in bold and red, denote the superior outcomes for both feature selection and extraction. These best

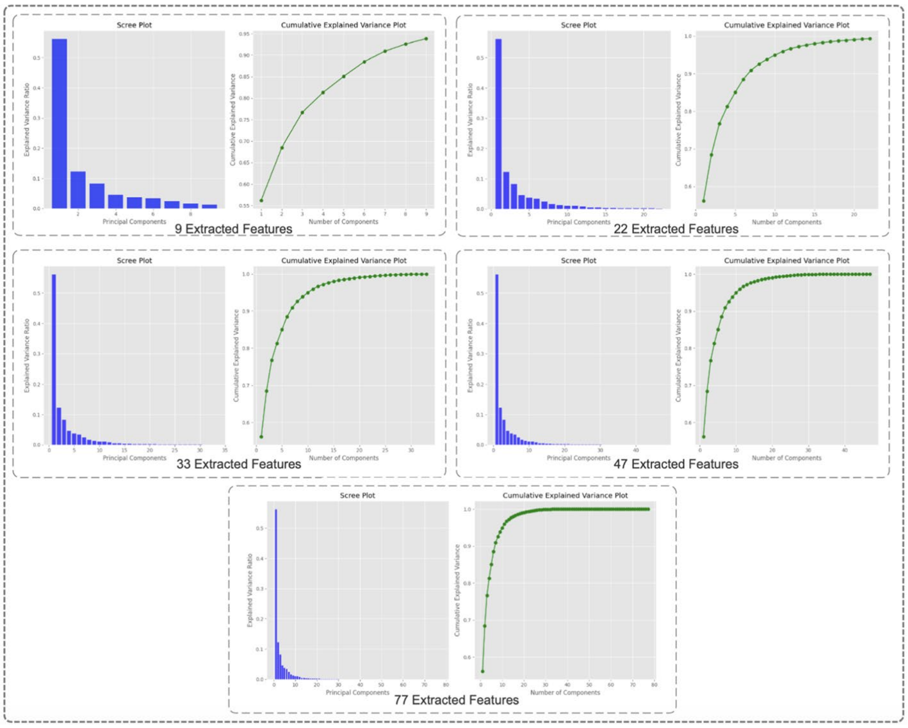


Fig. 7 The explained variance and cumulative total variance for extracted feature schemes

Table 6 FS vs. FE for binary classification with 9 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	**80.73**	**79.50**	**77.74**	**78.44**	**0.5721**	**8.38**	**0.22**	**12.11**
RF	79.07	78.53	74.55	75.73	0.5293		7.56	801.32
kNN	77.44	82.90	69.38	70.66	0.5050		1.09	709,996.65
NB	78.99	82.82	71.94	73.58	0.5366		***0.11***	12.50
MLP	80.73	79.50	77.74	78.44	0.5721		37.3	136.17
Feature extraction								
DT	***86.54***	***85.12***	***86.33***	***85.62***	***0.7128***	***5.27***	**1.44**	***8.10***
RF	86.45	85.02	86.30	85.54	0.7127		18.61	838.21
kNN	71.00	76.33	76.79	70.99	0.3409		1.55	10,172.81
NB	83.35	81.76	82.68	82.15	0.6443		**0.13**	18.52
MLP	86.30	84.85	86.26	85.41	0.7111		81.58	139.20

Table 7 FS vs. FE for binary classification with 22 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	**81.27**	**80.00**	**78.55**	**79.15**	**0.5853**	**7.82**	0.46	***11.85***
RF	77.72	84.92	69.30	70.57	0.5192		8.99	776.08
kNN	78.65	**85.26**	70.66	72.19	0.5398		**0.07**	196,772.36
NB	78.34	85.02	70.24	71.69	0.5324		0.17	28.58
MLP	81.27	80.00	78.55	79.15	0.5853		56.56	174.12
Feature extraction								
DT	85.94	84.49	85.55	84.94	0.7119	***4.92***	1.84	**12.71**
RF	86.54	85.11	86.37	85.63	0.7147		26.44	631
kNN	64.29	62.85	63.80	62.82	0.7287		***0.05***	193,070.46
NB	84.77	83.26	84.75	83.83	0.6799		0.19	37.25
MLP	***86.53***	***85.11***	***86.42***	***85.64***	***0.7151***		128.43	478.01

Table 8 FS vs. FE for binary classification with 33 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	86.40	84.96	86.19	85.47	0.7114	**8.28**	0.64	**17.69**
RF	85.90	84.45	86.17	85.07	0.7059		11.74	848.32
kNN	83.75	**86.96**	78.30	80.30	0.6469		**0.13**	231,367.82
NB	79.92	85.77	72.51	74.33	0.5675		0.27	40.52
MLP	**86.45**	**85.01**	**86.29**	**85.54**	**0.7129**		75.38	184.73
Feature extraction								
DT	86.83	85.42	86.59	85.91	0.7201	***6.13***	3.13	***11.58***
RF	86.58	85.15	86.40	85.67	0.7154		38.67	657.02
kNN	***89.10***	***87.78***	***89.28***	***88.39***	***0.7669***		***0.06***	227,237.45
NB	83.37	83.56	79.55	80.89	0.6299		0.27	45.47
MLP	86.54	85.11	86.35	85.62	0.7151		45.43	84.14

Table 9 FS vs. FE for binary classification with 47 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	84.23	83.44	81.68	82.40	0.6509	**5.47**	0.86	**30.00**
RF	86.23	84.82	85.76	85.23	0.7057		15.96	524.71
kNN	82.82	82.28	79.54	80.55	0.6176		**0.09**	148,293.32
NB	81.20	84.15	75.15	77.02	0.5861		0.28	44.95
MLP	**86.52**	**85.09**	**86.34**	**85.61**	**0.7142**		67.58	72.34
Feature extraction								
DT	83.81	82.92	85.61	83.26	0.6848	***5.01***	6.50	***10.47***
RF	***86.94***	***85.54***	***86.72***	***86.04***	***0.7225***		35.72	569.59
kNN	86.76	85.34	87.16	86.00	0.7129		***0.05***	147,798.15
NB	69.74	70.52	59.96	58.85	0.2859		0.21	43.87
MLP	86.59	85.16	86.39	85.67	0.7152		59.03	105.07

Table 10 FS vs. FE for binary classification with 77 (full) features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	78.28	76.59	75.24	75.79	0.5128	***0***	1.65	**24.21**
RF	***88.22***	***86.99***	***89.56***	***87.69***	***0.7651***		12.94	553.13
kNN	80.55	80.74	83.44	80.19	0.6413		**0.09**	188,417.64
NB	59.57	71.04	67.75	59.20	0.3865		0.36	55.02
MLP	86.58	85.15	86.38	85.66	0.7153		70.78	83.49
Feature extraction								
DT	74.68	73.37	75.10	73.64	0.4845	**3.98**	10.01	***12.25***
RF	**87.04**	**85.65**	**86.78**	**86.14**	**0.7243**		47.68	579.27
kNN	80.56	80.75	83.45	80.19	0.6414		***0.08***	186,251.17
NB	79.76	81.24	73.97	75.58	0.5473		0.28	63.89
MLP	86.59	85.16	86.39	85.67	0.7153		86.44	152.35

values encompass the highest accuracy, precision, recall, F1-score, MCC, and the lowest feature reduction, training, and inference times within each table column. Time values for feature reduction and training are measured in seconds (s), while inference time per data sample is measured in milliseconds (ms).

Regarding the classification performance, we initially explore the impact of an increasing number of features on the performance of both FS and FE methods. Expanding the number of features appears to enhance the performance of the FS model, while this increase shows no obvious effect on the FE model. Figure 8 illustrates that as the number of features increases, the performance of FS models generally improves from 9 features to 77 full features. In contrast, the performance of FE models remains nearly consistent, with the exception of the kNN model, which displays optimal performance with 33 features, as indicated in Fig. 9. While the performance of the best models in FS improves as the number of features increases from Tables 6, 7, 8, 9, and 10, the performance of certain models, like the decision tree, significantly decreases from Tables 8, 9, and 10. This trend aligns with the expectation

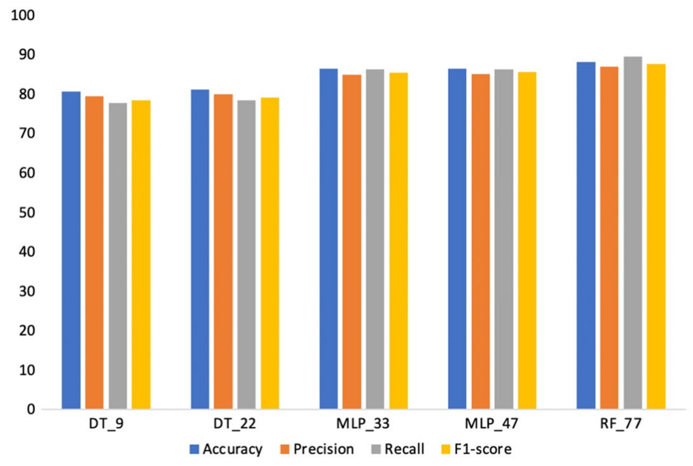


Fig. 8 The best performance of FS models for binary classification

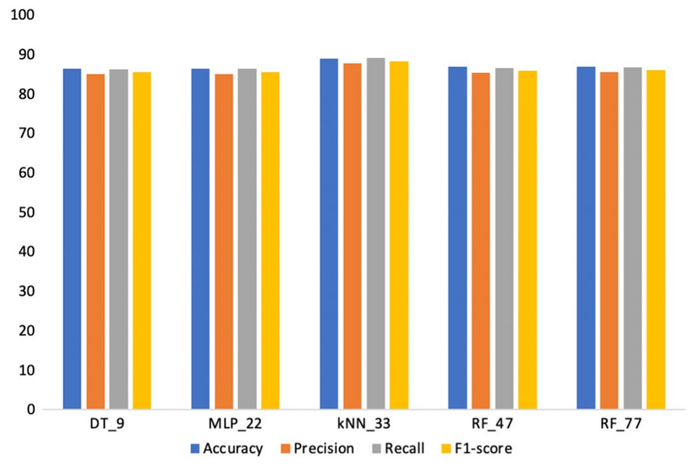


Fig. 9 The best performance of FE models for binary classification

that as the number of selected features increases, more irrelevant or noisy features might emerge, potentially impacting the detection performance negatively.

Furthermore, the performance of FE in classification surpasses that of FS, particularly when considering a small number of features. The comparison between the two methods in Fig. 10 reveals that when the number of reduced features is relatively small—9, 22, 33, and 47—the classification performance of FE notably outperforms that of FS. This advantage is particularly pronounced for the cases involving 9 and 22 features. For instance, as demonstrated in Table 6, using the DT classifier, the highest accuracy and F1-score of FE are 86.54% and 85.62%, respectively, while FS exhibits lower performance with 80.73% accuracy and 78.44% F1-score using the same DT classifier. However, as the number of features increases, for instance, to 47 and 77 full features in Tables 9 and 10, the effectiveness of FE gradually diminishes relative to FS. In the case of full features, both FS and FE favorite RF classifier to achieve the best performance metrics compared with other classifiers, Furthermore, FS exceeds FE in

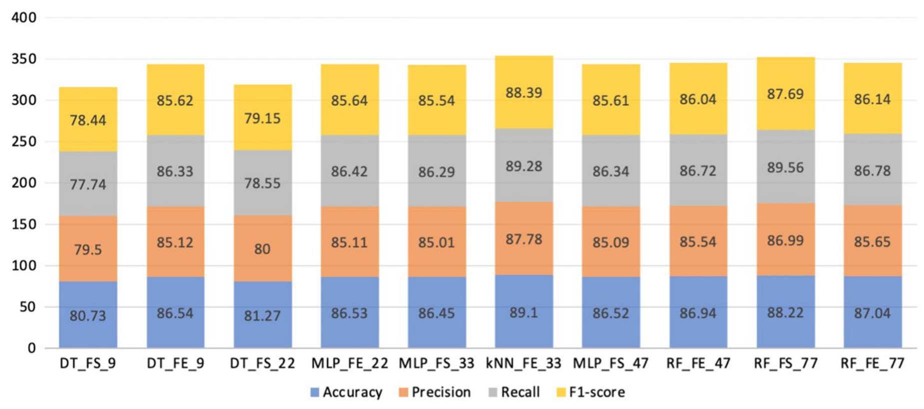


Fig. 10 The performance comparison of FS and FE models for binary classification

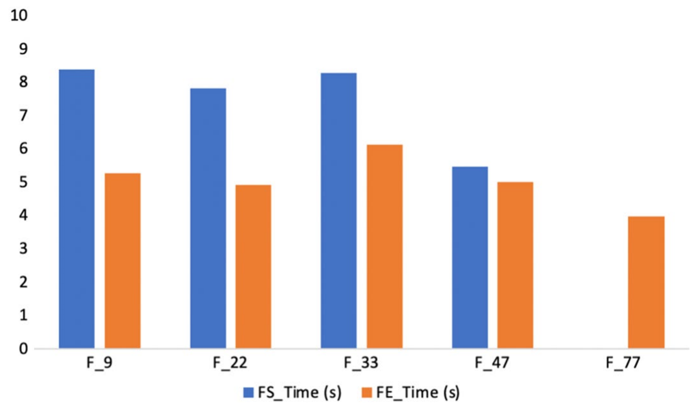


Fig. 11 The run time of FS and FE with reduced features in binary classification

terms of accuracy and F1-score, with 88.22% and 87.69%, respectively, surpassing FE with 87.04% and 86.14% under the same RF classifier.

As for which model two feature reduction methods prefer, the favorite models of FS and FE are different when the number of reduced features are different. Tables 6 and 7 demonstrate that with FS, the DT classification method consistently delivers the highest accuracy, precision, recall, and F1-score. This is followed by the MLP model, which becomes more favorable in Tables 8 and 9, and ultimately, the RF model emerges as the optimal choice with the full feature set in Table 10. In contrast, the FE method exhibits a different pattern, initially favoring DT with 9 features in Table 6, transitioning to MLP with 22 features in Table 7, and then showing stronger performance with the kNN classifier at 33 features, which marks the peak performance point. Subsequently, the RF classifier becomes the preferred choice for the FE method in Tables 9 and 10.

As for run time performance, we firstly investigate the run time of the two feature reduction methods. It is shown from Fig. 11. The runtime efficiency of FE surpasses that of FS, especially with a smaller number of features—specifically, 9, 22, and 33. However, the disparity in runtime between the two feature reduction methods narrows as the reduced features increase. This occurs because the FS algorithm demands extra computational resources to compute the average correlation score for every feature, making it more time-consuming compared to the PCA-based feature extraction, which compresses high-dimensional data into a lower-dimensional format, as detailed in "Methodology" section.

Moreover, as for the model training time, FS takes less time than FE with small number of features, such as 9 and 22 for all the models according to the Tables 6 and 7, particularly for the model training time of DT and RF for all feature settings based on Tables 6, 7, 8, 9, and 10. However, the training time of kNN, NB, and MLP in feature selection exceeds that in feature extraction when reduced features are increasing, such as 33, 47 and 77, except for the case of MLP in 77 full features, in which the training time under FS is 70.78 s, while that with features under feature extraction is 86.44 s.

The inference time comparison between models using FS and FE shows a consistent superiority in favor of the FE model for all feature settings except the case with 22 features. Notably, the DT classifier remains the optimal choice for both feature reduction methods in minimizing inference time. The DT classifier stands out among other classifiers for reducing both training and inference times. Similarly, the kNN classifier exhibits the shortest training time but considerably prolonged inference time, while the NB classifier, despite its weaker accuracy performance, demonstrates modest computational efficiency.

Finally, in order to better understand the attack detection performance of FS and FE, we evaluate and compare the class-wise f1-score, namely normal and attack, using the best model of FS and FE respectively in each feature setting, involving 9, 22, 33, 47 and 77 full features. We can refer to the Fig. 12, the F1-score of both normal and attack traffic improve little with the increasing number of features for both feature selection and extraction, which demonstrate the effect of feature reduction method to achieve good performance with less run time of created models. Moreover, the F1-score of normal traffic is obviously higher than that of attack traffic for all feature settings, that's because of the class imbalance before normal and attack case in training set, which makes sense since class imbalance handling, which is not the focus of our work, is not implemented.

In addition, we find that the performance of FE can achieve the highest performance result with more fewer features, for example, FS for both DT and MLP need 33 features to achieve highest F1-score, while, FE can achieve the same performance value using 9 features on DT classifier, and 22 features on MLP classifier. Moreover, FE is less sensitive

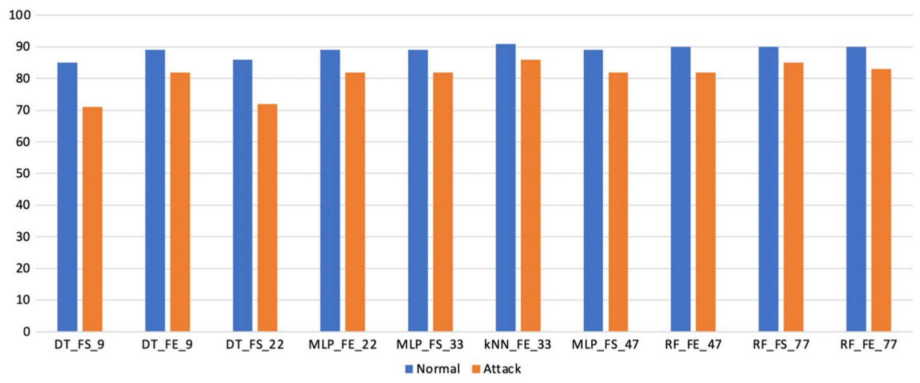


Fig. 12 The class-wise F1-score between FS and FE methods for reduced features

Table 11 Class level F1-score analysis between FS and FE in binary classification

Class	Feature selection (DT)					Feature extraction (DT)				
	Number of features					Number of features				
	9	22	33	47	77	9	22	33	47	77
Normal	0.85	0.86	**0.89**	0.88	0.84	**0.89**	0.89	0.89	0.86	0.79
Attack	0.71	0.72	**0.82**	0.77	0.68	**0.82**	0.81	0.82	0.80	0.68
Average	0.78	0.79	0.85	0.82	0.76	0.85	0.85	0.86	0.83	0.74
Class	Feature selection (MLP)					Feature extraction (MLP)				
	Number of features					Number of features				
	9	22	33	47	77	9	22	33	47	77
Normal	0.85	0.86	**0.89**	0.89	0.89	0.89	**0.89**	0.89	0.89	0.89
Attack	0.71	0.72	**0.82**	0.82	0.82	0.82	**0.82**	0.82	0.82	0.82
Average	0.78	0.79	0.86	0.86	0.86	0.85	0.86	0.86	0.86	0.86

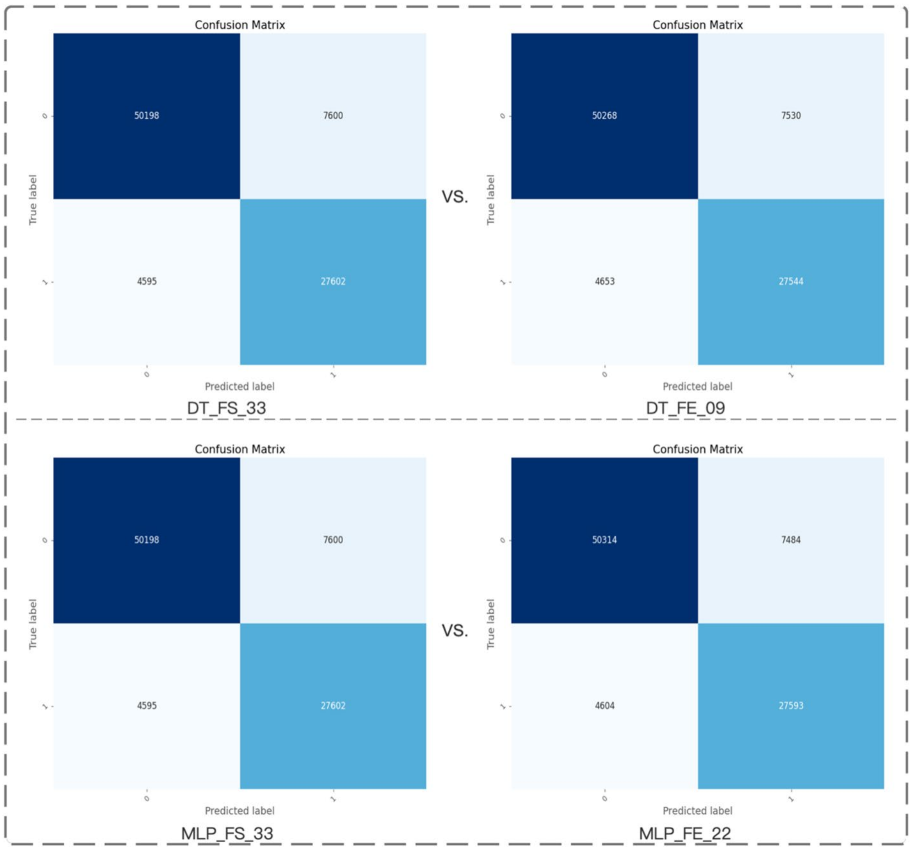


Fig. 13 The confusion matrix of the outstanding models between FS and FE in binary classification

to different models' differing feature counts, such as DT and MLP, while that of FS, varies significantly compared with that of FE, according to the Table 11. However, F1-score of FS for both DT and MLP improves significantly when the number of features increase from 9 till 33, particularly for the attack class, compared with that of FE, which proved that the performance can be improved when more informative features are added. Moreover, based on the outstanding models highlighted in the table, we further present

Table 12 FS vs. FE for multiclass classification with 9 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	**72.65**	**40.81**	**33.33**	**29.39**	**0.4553**	**8.38**	***0.75***	***11.00***
RF	71.42	33.75	27.65	24.06	0.3606		10.00	826.79
kNN	70.11	33.06	26.29	22.29	0.5271		4.26	807,363.15
NB	19.22	24.14	34.37	19.2	0.2498		0.88	50.40
MLP	72.65	40.81	33.33	29.39	0.4553		72.16	230.48
Feature extraction								
DT	***77.04***	***57.75***	***45.29***	***40.66***	***0.5768***	***5.27***	**1.40**	**12.13**
RF	76.25	42.42	43.33	36.83	0.5563		20.99	1317.2
kNN	41.74	43.36	21.24	15.26	0.1835		1.53	10,193.23
NB	52.43	30.69	50.32	32.64	0.4104		0.81	72.35
MLP	76.90	53.66	45.11	39.77	0.5692		129.93	239.49

Table 13 FS vs. FE for multiclass classification with 22 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	**73.52**	**59.02**	**35.47**	**32.53**	**0.4781**	**7.82**	***1.60***	**19.29**
RF	69.24	28.63	21.32	16.82	0.3446		10.19	790.10
kNN	72.55	60.57	30.39	28.82	0.4451		0.64	192,447.54
NB	19.36	34.90	41.03	17.70	0.2758		1.58	99.98
MLP	73.52	59.02	35.47	32.53	0.4781		107.55	179.01
Feature extraction								
DT	***77.25***	***72.18***	***48.18***	***45.00***	**0.5840**	***4.92***	**3.47**	***12.93***
RF	77.14	61.21	45.24	40.45	0.5716		25.02	922.89
kNN	51.86	41.04	31.09	29.50	0.5809		1.23	200,249.55
NB	55.84	37.60	**63.32**	41.57	0.4618		1.92	169.60
MLP	**77.43**	66.47	46.00	41.50	0.5793		180.86	135.30

the confusion matrix of the outstanding models in Fig. 13. We can find out that FS outperforms FE slightly on normal traffic classification, while provides less capability to recognize the attack traffic.

Multiclass classification

Next, we investigate the performance and computational time analysis of both FS and FE methods for multi-class classification, using Tables 12, 13, 14, 15, and 16. The same as the logic of the binary classification, five iterations are performed for each feature number scheme in order to obtain an affirmative result. The results of each cycle are then used to calculate the average result. The tables highlight the best values in bold and underscore the superior results for both feature selection and extraction in 'bold and italics', following the same criteria applied in the binary classification outlined in "Binary classification" section.

The performance of multi-class classification performance is significantly lower than that of binary classification, such as the accuracy and f1-score of the best trained model

Table 14 FS vs. FE for multiclass classification with 33 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	**77.23**	***77.23***	**45.69**	**41.00**	**0.5750**	**8.28**	***1.20***	***15.34***
RF	70.21	37.68	23.57	20.42	0.3751		26.77	1558.13
kNN	76.23	65.60	40.58	37.63	0.5440		**0.69**	225,211.54
NB	32.64	29.45	45.45	23.91	0.2758		0.96	157.66
MLP	77.28	66.66	45.83	**41.26**	0.5762		158.29	198.80
Feature extraction								
DT	***77.62***	**67.34**	***48.67***	***45.53***	***0.5831***	***6.13***	**5.18**	**21.81**
RF	77.33	61.54	45.64	41.19	0.5753		30.58	901.71
kNN	66.34	46.62	33.70	30.94	0.4532		0.66	227,584.21
NB	44.71	42.86	57.43	39.01	0.3895		1.67	233.32
MLP	77.45	68.01	46.00	41.50	0.5792		213.14	194.79

Table 15 FS vs. FE for multiclass classification with 47 features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	**77.25**	***72.18***	***48.18***	***45.00***	**0.5524**	**5.47**	***3.47***	***12.93***
RF	77.14	61.21	45.24	40.45	0.3606		25.02	922.89
kNN	51.86	41.04	31.09	29.50	0.5271		1.23	200,249.55
NB	55.84	37.60	63.32	41.57	0.2498		1.92	169.6
MLP	77.43	66.47	46.00	41.50	0.5638		180.86	135.3
Feature extraction								
DT	65.35	50.12	37.58	32.36	0.4156	***5.01***	**7.85**	**12.35**
RF	76.65	52.36	44.17	39.43	0.5626		35.90	719.71
kNN	67.51	55.02	39.23	33.82	0.4694		0.44	143,359
NB	32.40	36.20	56.10	31.93	0.3050		0.64	210.78
MLP	***77.46***	**68.08**	**46.02**	**41.51**	***0.5795***		90.35	103.62

Table 16 FS vs. FE for multiclass classification with 77 (full) features

Models	Accuracy (%)	Precision (%)	Re-call (%)	F1-score (%)	MCC	FS (s)	Training (s)	Inference (ms)
Feature selection								
DT	51.29	33.16	25.02	16.00	0.2450	***0***	1.94	16.79
RF	69.38	29.90	21.68	19.06	0.3507		13.46	758.22
kNN	57.24	48.56	34.46	25.69	0.3720		0.48	187,612.41
NB	20.14	34.66	41.28	19.76	0.1973		0.79	257.37
MLP	**77.47**	**67.96**	**46.03**	**41.56**	**0.5796**		**151.02**	***117.19***
Feature extraction								
DT	54.98	38.23	34.68	22.60	0.2982	**3.98**	10.01	16.80
RF	76.84	62.80	44.55	39.72	0.5661		48.97	801.26
kNN	57.25	45.09	34.42	25.47	0.3718		0.49	188,883.26
NB	22.50	34.92	42.80	22.52	0.2204		0.76	322.33
MLP	***77.46***	***68.01***	***46.05***	***41.53***	**0.5796**		***116.71***	**118.47**

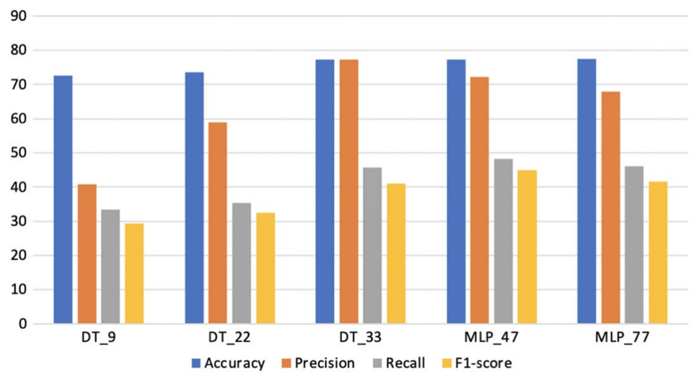


Fig. 14 The best performance of FS models for multi-class classification

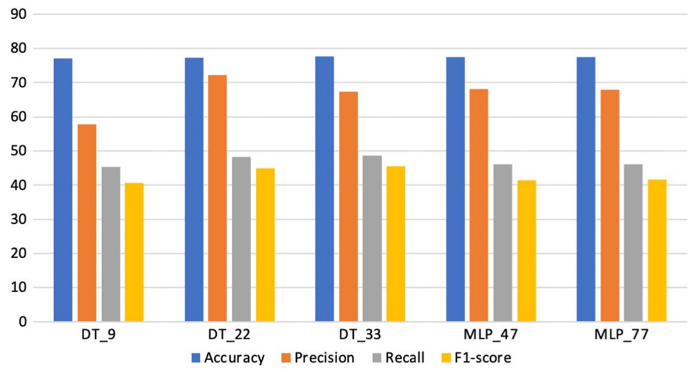


Fig. 15 The best performance of FE models for multi-class classification

decision tree in multi-classification are 72.65% and 29.39%, while those of are 80.73% and 78.44% in binary classification, when using 9 selected features. Because more complex sub-class distribution of the data and less training data for each class, especially for the rare attack instances such as MITM attack, which cause low detection rate compared with normal/attack binary models. However, we concentrate on the performance comparison of two feature reduction methods under multi-class classifiers.

We firstly investigate how performance changes with increasing number of features for both feature selection and feature reduction method. As the same outcome of binary classification, the increasing number of reduced features can improve the performance of feature selection model, while there is no obvious effect for that of feature extraction model. It is shown from Figs. 14 and 15, when the number of reduced features increases, the classification performance of feature selection generally improves, particularly from that of 9 features to 47 selected features, while that of feature extraction has no significant improvement, particularly for the accuracy. In addition, more features can also cause performance degrade for both feature selection and feature extraction. For example, the f1-score of FS model decrease when the number of features increased from 47 to 77 (full) features, while that of FE model decrease starting from 33 features, since more noisy or irrelevant features are expected to worsen the detection performance.

Moreover, the classification performance of FE is much better than that of FS especially for small number of features, which is the same as that of binary classification. As

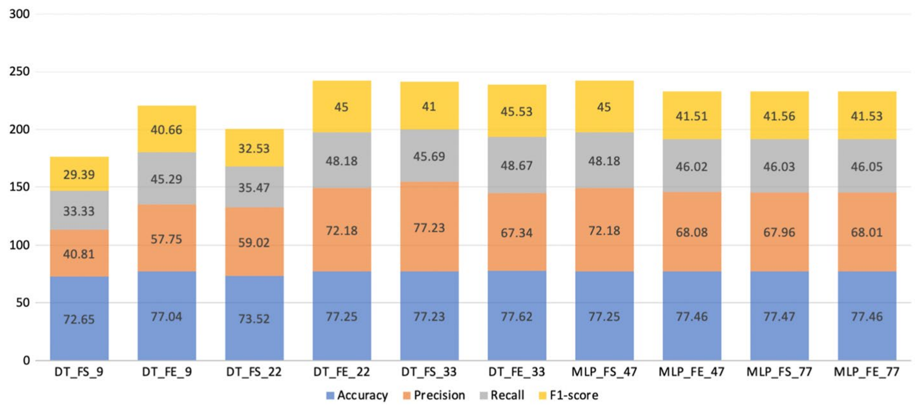


Fig. 16 The performance comparison of FS and FE models for multi-class classification

it is shown from Fig. 16, comparing the two feature reduction methods, we find that the classification performance of feature extraction is much better than that of feature selection, when the number of features is relatively small, such as 9 and 22. For instance, the highest accuracy and F1-score of DT model with feature extraction is 77.04% and 40.66% respectively, while those of feature selection is lower with 72.65% and 29.39%. However, when there is increasingly larger number of features added, starting from 33, 47 till 77 full features, the performance gap between FS and FE is not significant, and only the precision of FS is higher than that of FE, when the number of features is 33, 47 and 77 full features.

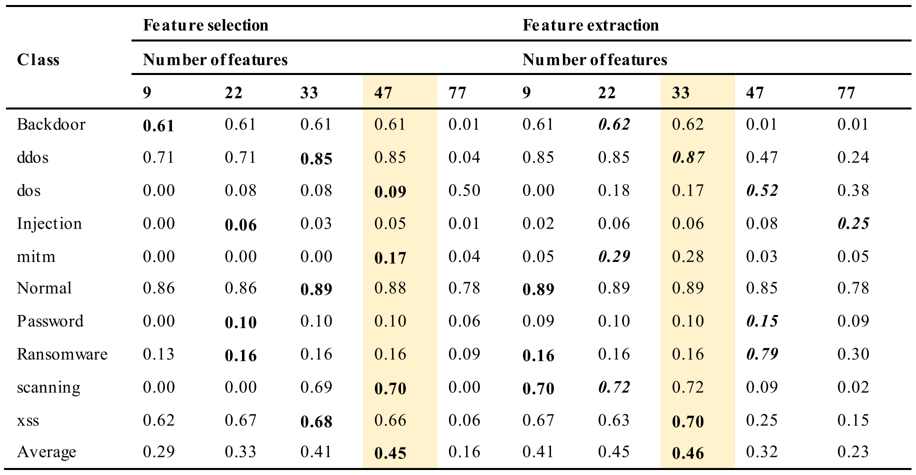
In addition, different from the binary scenario, where MLP is the best classifier for feature selection, and KNN is the best classifier for feature extraction, with 33 features, while the favorite models of multi-class classification using FS and FE are different. It is showed from the Figs. 14 and 15, DT outperforms other classifiers in both FS and FE, when the number of features is relatively small, such as 9, 22, 33 features, while MLP achieve higher performance than other models, when that of features increase to 47 till 77 full features, that is because DT as less complex tree-based model can handle the data with limited features, however, MLP as more complex neural networks can handle the data with more features.

As for run time performance, we firstly investigate the run time of the two feature reduction methods. Since there is no change for the two feature reduction algorithms in binary classification and multiple classification, thus the run time for multi-class classification is the same as that of binary classification, which is explained in "Binary classification" section. Moreover, as for the model training time, the same as that of binary classification, FS takes less time than FE when the number of features is relatively small, such as 9, 22, and 33 for all the DT models according to the Tables 12, 13 and 14, However, when MLP as the best performance model in the number of features 47 and 77 full features, the model training time of FE is significant lower than that of FS, according to the Tables 15 and 16. As for the inference time of the best performance models, DT is obviously more efficient than that of MLP for both feature selection and feature extraction for all feature settings. Inference time of FS is lower than that of FE when limited number of features are used, such as 9, however, in contrast, more time is used than for FS than FE when the number of features increased.

Table 17 Class-wise F1-score comparison between FS and FE in multiclass classification

Class	**Feature selection**					**Feature extraction**				
	Number of features					**Number of features**				
	9	**22**	**33**	**47**	**77**	**9**	**22**	**33**	**47**	**77**
Backdoor	**0.61**	0.61	0.61	0.61	0.61	0.61	***0.62***	0.62	0.61	0.61
ddos	0.71	0.71	0.85	**0.86**	0.86	0.85	0.85	***0.87***	0.86	0.86
dos	0.00	0.08	0.08	**0.09**	0.09	0.00	***0.18***	0.17	0.09	0.09
Injection	0.00	0.06	0.03	***0.08***	0.06	0.02	0.06	0.06	0.06	0.06
mitm	0.00	0.00	0.00	0.00	0.00	0.05	***0.29***	0.28	0.00	0.00
Normal	0.86	0.86	**0.89**	0.89	0.89	**0.89**	0.89	0.89	0.89	0.89
Password	0.00	**0.10**	0.10	0.10	0.10	0.09	**0.10**	0.10	0.10	0.10
Ransomware	0.13	**0.16**	0.16	0.16	0.16	**0.16**	0.16	0.16	0.16	0.16
Scanning	0.00	0.00	0.69	**0.70**	0.70	**0.70**	***0.72***	0.72	0.70	0.70
xss	0.62	0.67	**0.68**	0.68	0.68	0.67	0.63	***0.70***	0.68	0.68
Average	0.29	0.33	0.41	**0.45**	0.42	0.41	0.45	***0.46***	0.42	0.42

Table 18 Class-wise F1-score comparison between FS and FE in multiclass classification of the same DT



Class	**Feature selection**					**Feature extraction**				
	Number of features					**Number of features**				
	9	**22**	**33**	**47**	**77**	**9**	**22**	**33**	**47**	**77**
Backdoor	**0.61**	0.61	0.61	0.61	0.01	0.61	***0.62***	0.62	0.01	0.01
ddos	0.71	0.71	**0.85**	0.85	0.04	0.85	0.85	***0.87***	0.47	0.24
dos	0.00	0.08	0.08	**0.09**	0.50	0.00	0.18	0.17	***0.52***	0.38
Injection	0.00	**0.06**	0.03	0.05	0.01	0.02	0.06	0.06	0.08	***0.25***
mitm	0.00	0.00	0.00	**0.17**	0.04	0.05	***0.29***	0.28	0.03	0.05
Normal	0.86	0.86	**0.89**	0.88	0.78	**0.89**	0.89	0.89	0.85	0.78
Password	0.00	**0.10**	0.10	0.10	0.06	0.09	0.10	0.10	***0.15***	0.09
Ransomware	0.13	**0.16**	0.16	0.16	0.09	**0.16**	0.16	0.16	***0.79***	0.30
scanning	0.00	0.00	0.69	**0.70**	0.00	**0.70**	***0.72***	0.72	0.09	0.02
xss	0.62	0.67	**0.68**	0.66	0.06	0.67	0.63	**0.70**	0.25	0.15
Average	0.29	0.33	0.41	**0.45**	0.16	0.41	0.45	**0.46**	0.32	0.23

Finally, our comparison focuses on the F1-scores for detecting individual attack types, involving 10 attack classes and 1 normal class (as outlined in "Methodology" section) across Tables 17, 18, and 19. These tables outline the performance of FS and FE using 9, 22, 33, 47 features, and 77 full features. Within these tables, our emphasis remains on the DT and MLP classifiers for FE and FS, respectively, to achieve optimal detection performance, as previously discussed. Observations from Tables 17 and 18 indicate that FE generally outperforms FS across most classes, except for the injection attack in Table 19. Notably, both methods achieve higher F1-scores for specific classes such as DDoS, Normal, Scanning, and XSS in contrast to other classes.

Remarkably, the multiclass classification accuracy of FE proves less affected by the number of reduced features compared to FS. A significant finding emerges from FS's inability to accurately detect any MITM samples, even with the best models, across all numbers of features. In contrast, FE using the best classifiers can successfully detect

Table 19 Class-wise F1-score comparison between FS and FE in multiclass classification of the same MLP

Class	Feature selection					Feature extraction				
	Number of features					Number of features				
	9	22	33	47	77	9	22	33	47	77
Backdoor	**0.61**	0.61	0.61	0.61	0.61	0.61	**0.61**	0.61	0.61	0.61
ddos	0.71	0.71	0.85	**0.86**	0.86	0.85	**0.86**	0.86	0.86	0.86
dos	0.00	0.08	0.08	**0.09**	0.09	0.00	**0.09**	0.09	0.09	0.09
Injection	0.00	0.06	0.03	***0.08***	0.06	0.02	0.06	0.06	0.06	0.06
mitm	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Normal	0.86	0.86	**0.89**	0.89	0.89	0.89	**0.89**	0.89	0.89	0.89
Password	0.00	**0.10**	0.10	0.10	0.10	0.09	**0.10**	0.10	0.10	0.10
Ransomware	0.13	0.16	**0.16**	0.16	0.16	0.16	**0.16**	0.16	0.16	0.16
Scanning	0.00	0.00	0.69	**0.70**	0.70	0.70	**0.70**	0.70	0.70	0.70
xss	0.62	0.67	**0.68**	0.68	0.68	0.67	**0.68**	0.68	0.68	0.68
Average	0.29	0.33	0.41	***0.45***	0.42	0.40	**0.42**	0.42	0.42	0.42

MITM samples with 9, 22, and 33 features. This observation is primarily attributed to the machine learning classifier rather than the chosen feature reduction method.

Further exploration into the matter includes a comparison of the F1-scores for each class between the two feature reduction methods, using the same Decision Tree and MLP classifiers in Tables 18 and 19, respectively. Table 19 shows that, similar to FS, FE with the same MLP classifier is unable to detect any MITM attack samples accurately. These tables demonstrate that FE, employing the same classifier, tends to identify a broader range of attack types compared to FS. For example, FS with DT classifiers fails to detect Injection, Password, and Scanning attacks under 9 features, while its counterpart successfully identifies these attacks across all five numbers of features. The disparity arises from FE's ability to extract crucial information from all available features, enabling detection of a wider range of attack types, unlike the FS approach that predominantly relies on a subset of selected features highly correlated to specific attack types.

Additionally, we look into the confusion matrix of the excellent models in Fig. 17 based on the models that stand out in the table. In contrast to the binary classification result, we can observe that FE marginally beats FS in the usual traffic identification task. In particular, DT_FE_33 and MLP_FE_22 are both capable of identifying normal traffic more accurately than DT_FS_47 and MLP_FS_47, respectively. In terms of attack categorization, DT_FE_33 is more capable than MLP_FE_22; on the other hand, MLP_FS_22 is less capable than MLP_FE_22 of classifying backdoor assaults, while MLP_FE_22 is less capable than MLP_FS_47 of recognizing DoS attacks. No model hyperparameter adjustment or class imbalance optimization may result in a loss of the ability to detect every attack. However, since the goal of this study is to distinguish between feature extraction and feature selection, we can do optimization to improve classification performance even more in the future.

Result verification statistically

In order to come up with the affirmative conclusion, statistical verification is implemented using T-test in this section. Two metrics are used for verification including

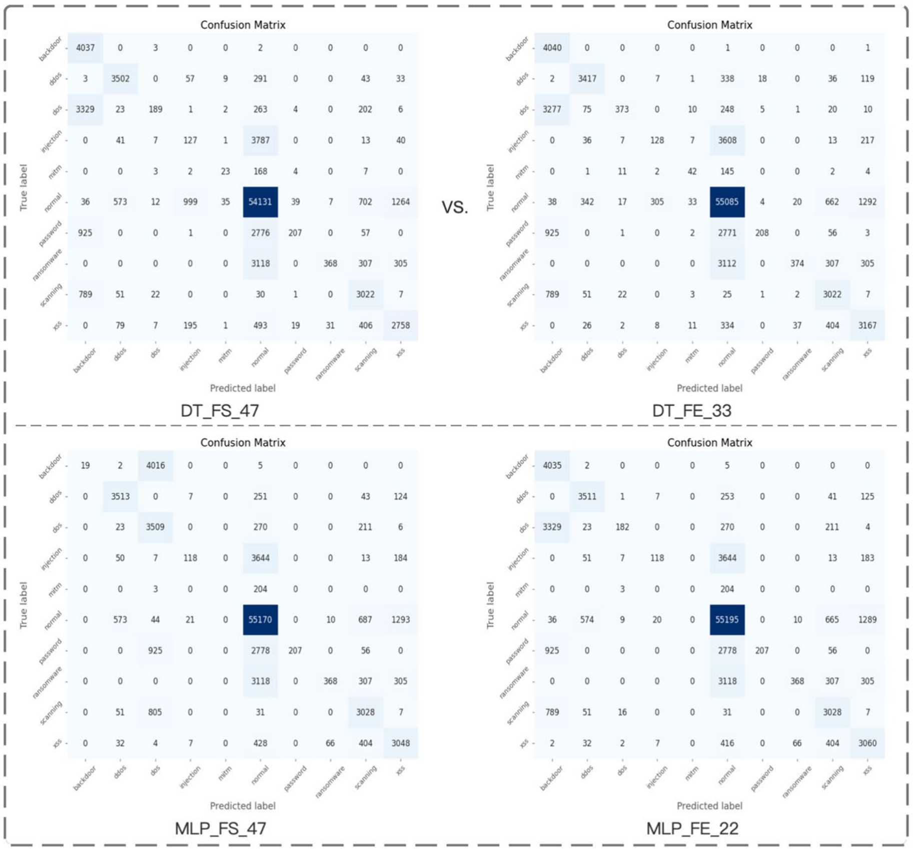


Fig. 17 The confusion matrix of the outstanding models between FS and FE in multi-classification

Table 20 Summary of t-test verification test

No.	Data	T-statistic	P-values	Significant
1	Accuracy of binary and multiclassification using FE between 9, 22 and 33, 47 features respectively	− 9.3139	0.0026	*
2	Accuracy of binary and multiclassification using FS between 33, 47 and 9, 22 features respectively	− 7.0103	0.0006	*
3	Feature reduction time between FS and FE for each feature scheme	3.5833	0.0372	*
4	Model training time between FS and FE for each feature scheme	− 2.2707	0.0324	*
5	Model inference time between FS and FE for each feature scheme (except for kNN model)	− 3.4921	0.0251	*
6	DT runtime (including the run time of both model building and inference) compared to other models for both FS and FE	− 3.6216	0.0152	*

t-statistic and p-value, for example, the negative sign indicates that the average of one group is significantly below the average of the other group. The t-statistic is a measure of how many standard deviations a data point is from the mean of the distribution. The p-value is a measure of the evidence against a null hypothesis. In the context of a t-test, it represents the probability of obtaining the observed results (or more extreme) if the null hypothesis is true. A small p-value (typically less than the significance level, e.g., 0.05) suggests that we can reject the null hypothesis. The statistical test summary, Table 20, is based on the original data in binary and multi-classification results from Tables 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16. The data in comparison Table 21 that have been validated using the t-test method are listed here; for information on the other conclusion features, see the tables and figures in "Binary classification" and "Multiclass classification" sections.

In summary, when evaluating binary and multiclass classification within the NIDS utilizing the Network TON-IoT dataset, FE emerges as not only offering superior classification performance but also reduced feature reduction time compared to its FS counterpart, particularly as the number of reduced features increases. The advantage of FE is notably higher and more consistent, especially with smaller quantities of features, such as 9 and 22. However, FS generally demonstrates shorter model training and inference times than feature extraction, which is significant to lightweight model design of NIDS in IoT network.

Among the five classifiers, the DT proves to be the optimal choice for enhancing the performance of both feature reduction methods, particularly when the number of features remains small, such as 9 and 22. Conversely, a neural network-based MLP exhibits superior performance for both reduction methods as the number of features increases, reaching values of 33, 47, and 77 full features (refer to the Figs. 14, 15). It's important to highlight that FE shows less sensitivity to changes in the number of reduced features compared to FS, a trend that remains consistent across both binary and multiclass classifications. A detailed, comprehensive comparison between two feature reduction methods in NIDS using contemporary IoT dataset is provided in Table 21 for further insights.

Table 21 Comparison between FE and FS in various scenarios

No.	Content	FS	FE
1	Higher accuracy when no. of features is small, such as 9 and 22		✓
2	Higher accuracy when no. of features gets large, such as 33 and 47	✓	
3	Lower feature reduction time		✓
4	Lower model training time	✓	
5	Lower inference time	✓	
6	DT is the most favorite classifier considering runtime	✓	✓
7	DT is the most favorite classifier considering performance for multi-classification with small and moderate number of features, such as 9, 22, and 33	✓	✓
8	MLP is the most favorite classifier considering performance for multi-classification with more features, such as 47 and 77	✓	✓
9	Less sensitive to the number of selected/extracted features		✓
10	Less sensitive to various machine learning models		✓
11	Detection performance degrades when number of features is too large		✓
12	Detection performance increase when more informative features added	✓	
13	Detect more diverse attack types when using the same classifier		✓
14	F1-score of attack class is much lower than that of normal class (binary)	✓	✓
15	F1-score of attack class degrades when number of features increases (binary)	✓	
16	Higher F1-score in detecting DDoS, normal, scanning and XSS classes		✓
17	Higher F1-score in detecting injection classes	✓	
18	More potential to improve performance when the number of features is small	✓	
19	More potential to improve performance when the number of features is large		✓

Conclusions

IoT systems and networks always suffer from computational resource constraints, which impact the applicability of attack classification model training, validation, and deployment for cyber security in real IoT scenarios. Feature reduction is pivotal for constructing a cost-effective and lightweight model capable of classifying attacks in IoT scenarios. Specifically, the objective is to mitigate the challenges associated with resource constraints in IoT devices by reducing the number of features through a thorough evaluation of feature selection and feature extraction methods. In this study, we conducted a thorough comparison between two dimensionality reduction methods, FS and FE, using the contemporary and heterogenous Network TON-IoT dataset for classification in NIDS. Our extensive analysis revealed that, when reducing a significant number of features (e.g., 9 or 22), FE not only achieved higher accuracy in attack detection but also required less time for dimensionality reduction. However, as the number of features increased (e.g., 33 or more), FS outperformed feature extraction. Therefore, FS demonstrated more potential with fewer features, whereas FE showed room for improvement with a larger number of features. Additionally, we observed that the effectiveness of FS declined significantly with an increased number of selected features, while FE consistently improved. Our study identified the MLP as the optimal classifier for FE, while the DT was the top performer in FS providing the highest accuracy in attack detection. Both two reduction methods favor the DT for lightweight classification models. Moreover, we found that FE is less sensitive to changes in the number of features and can detect a broader range of attack types compared to FS. Both methods exhibit a tendency to detect more attacks, especially abnormal classes, when a larger number of features are selected or extracted. These insights offer valuable guidance for choosing the most suitable intrusion detection method in specific IoT scenarios. It's important to note that our assessment concentrated on two specific feature reduction techniques using classic machine learning algorithms on the TON-IoT dataset. Future research aims to explore the applicability of these findings across a variety of IoT datasets with different applications, such as IoTNIDS, BoT-IoT, MQTT-IoT-IDS, and Edge-IIoTSet. Moreover, our future plans involve conducting an extensive evaluation of additional feature reduction methods within authentic IoT environments, aiming to narrow the gap between academic offline analysis and real-time analysis in practical IoT scenarios.

Acknowledgements
I am very grateful to Dr. Shahizan for his invaluable knowledge on this project. This project was completed with the help of his review and guidance suggestions. I express my gratitude to Ms. Hewan Chen and Dr. Lizawati for their excellent teamwork in working on this paper together.

Author contributions
JL: conceptualization, methodology, investigation, writing—original draft preparation. MSO: review and editing, supervision. HC: methodology, software, validation, formal analysis, visualization. LMY: resources, data curation, project administration. All authors have read and agreed to the published version of the manuscript.

Funding
This work was conducted without specific external funding. The authors independently conducted and completed the research with no financial support from any funding agency or grant.

Availability of data and materials
Data will be made available on request.

Declarations

Ethics approval and consent to participate
This research project does not involve the use of human or animal subjects. As such, formal ethical approval was not required for the completion of this study. Informed consent was obtained from all individual participants included in the study.

Consent for publication
All authors have consented to the submission and publication of this manuscript. Informed consent for the use of any images, patient details, or other potentially identifiable information has been obtained.

Competing interests
The authors declare that they have no competing interests relevant to the publication of this manuscript.

Received: 14 November 2023 Accepted: 11 February 2024
Published online: 24 February 2024

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. 2015;17(4):2347–76. https://doi.org/10.1109/COMST.2015.2444095.
2. Zhou W, Jia Y, Peng A, Zhang Y, Liu P. The effect of IoT new features on security and privacy: new threats, existing solutions, and challenges yet to be solved. IEEE Internet Things J. 2019;6(2):1606–16. https://doi.org/10.1109/JIOT.2018.2847733.
3. Chaabouni N, Mosbah M, Zemmari A, Sauvignac C, Faruki P. Network intrusion detection for IoT security based on learning techniques. IEEE Commun Surv Tutor. 2019;21(3):2671–701. https://doi.org/10.1109/COMST.2019.2896380.
4. Mishra P, Varadharajan V, Tupakula U, Pilli ES. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Commun Surv Tutor. 2019;21(1):686–728. https://doi.org/10.1109/COMST.2018.2847722.
5. Tama BA, Comuzzi M, Rhee K-H. TSE-IDS: a two-stage classifier ensemble for intelligent anomaly-based intrusion detection system. IEEE Access. 2019;7:94497–507. https://doi.org/10.1109/ACCESS.2019.2928048.
6. Hall MA. Correlation-based feature selection for machine learning. Doctoral dissertation, The University of Waikato; 1999. p. 198.
7. Yan B, Han G. Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system. IEEE Access. 2018;6:41238–48. https://doi.org/10.1109/ACCESS.2018.2858277.
8. Amiri F, Rezaei Yousefi M, Lucas C, Shakery A, Yazdani N. Mutual information-based feature selection for intrusion detection systems. J Netw Comput Appl. 2011;34(4):1184–99. https://doi.org/10.1016/j.jnca.2011.01.002.
9. Aminanto ME, Choi R, Tanuwidjaja HC, Yoo PD, Kim K. Deep abstraction and weighted feature selection for Wi-Fi impersonation detection. IEEE Trans Inform Forensic Secur. 2018;13(3):621–36. https://doi.org/10.1109/TIFS.2017.2762828.
10. Zachos G, Essop I, Mantas G, Porfyrakis K, Ribeiro JC, Rodriguez J. Generating IoT edge network datasets based on the TON_IoT telemetry dataset. In: 2021 IEEE 26th international workshop on computer aided modeling and design of communication links and networks (CAMAD), Porto, Portugal. IEEE; 2021. p. 1–6. https://doi.org/10.1109/CAMAD52502.2021.9617799.
11. Saied M, Guirguis S, Madbouly M. Review of artificial intelligence for enhancing intrusion detection in the internet of things. Eng Appl Artif Intell. 2024;127: 107231. https://doi.org/10.1016/j.engappai.2023.107231.
12. Ambusaidi MA, He X, Nanda P, Tan Z. Building an intrusion detection system using a filter-based feature selection algorithm. IEEE Trans Comput. 2016;65(10):2986–98. https://doi.org/10.1109/TC.2016.2519914.
13. 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. In: Proceedings of the first workshop on building analysis datasets and gathering experience returns for security, Salzburg Austria. ACM; 2011. p. 29–36. https://doi.org/10.1145/1978672.1978676.
14. Kasongo SM, Sun Y. Performance analysis of intrusion detection systems using a feature selection method on the UNSW-NB15 dataset. J Big Data. 2020;7(1):105. https://doi.org/10.1186/s40537-020-00379-6.
15. Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 military communications and information systems conference (MilCIS), Canberra, Australia. IEEE; 2015. p. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942.
16. Disha RA, Waheed S. Performance analysis of machine learning models for intrusion detection system using Gini impurity-based weighted random forest (GIWRF) feature selection technique. Cybersecurity. 2022;5(1):1. https://doi.org/10.1186/s42400-021-00103-8.
17. Shafiq M, Tian Z, Bashir AK, Du X, Guizani M. CorrAUC: a malicious Bot-IoT traffic detection method in iot network using machine-learning techniques. IEEE Internet Things J. 2021;8(5):3242–54. https://doi.org/10.1109/JIOT.2020.3002255.
18. Koroniotis N, Moustafa N, Sitnikova E, Turnbull B. Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. Futur Gener Comput Syst. 2019;100:779–96. https://doi.org/10.1016/j.future.2019.05.041.
19. Khammassi C, Krichen S. A GA-LR wrapper approach for feature selection in network intrusion detection. Comput Secur. 2017;70:255–77. https://doi.org/10.1016/j.cose.2017.06.005.
20. Aslahi-Shahri BM, et al. A hybrid method consisting of GA and SVM for intrusion detection system. Neural Comput Appl. 2016;27(6):1669–76. https://doi.org/10.1007/s00521-015-1964-2.

21. Halim Z, et al. An effective genetic algorithm-based feature selection method for intrusion detection systems. Comput Secur. 2021;110: 102448. https://doi.org/10.1016/j.cose.2021.102448.
22. Alazzam H, Sharieh A, Sabri KE. A feature selection algorithm for intrusion detection system based on Pigeon inspired optimizer. Expert Syst Appl. 2020;148: 113249. https://doi.org/10.1016/j.eswa.2020.113249.
23. Liu J, Yang D, Lian M, Li M. Research on intrusion detection based on particle swarm optimization in IoT. IEEE Access. 2021;9:38254–68. https://doi.org/10.1109/ACCESS.2021.3063671.
24. Chohra A, Shirani P, Karbab EB, Debbabi M. Chameleon: optimized feature selection using particle swarm optimization and ensemble methods for network anomaly detection. Comput Secur. 2022;117: 102684. https://doi.org/10.1016/j.cose.2022.102684.
25. Moustafa N, Slay J. A hybrid feature selection for network intrusion detection systems: central points. In: Proceedings of the 16th Australian information warfare conference, held on the 30 November–2 December, 2015. p. 5–13. https://doi.org/10.4225/75/57A84D4FBEFBB.
26. Moustafa N, Turnbull B, Choo K-KR. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. IEEE Internet Things J. 2019;6(3):4815–30. https://doi.org/10.1109/JIOT.2018.2871719.
27. Leevy JL, Hancock J, Khoshgoftaar TM, Peterson JM. IoT information theft prediction using ensemble feature selection. J Big Data. 2022;9(1):6. https://doi.org/10.1186/s40537-021-00558-z.
28. 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 Secur J Glob Perspect. 2016;25(1–3):18–31. https://doi.org/10.1080/19393555.2015.1125974.
29. Gavel S, Raghuvanshi AS, Tiwari S. An optimized maximum correlation based feature reduction scheme for intrusion detection in data networks. Wirel Netw. 2022;28(6):2609–24. https://doi.org/10.1007/s11276-022-02988-w.
30. Zhou L, Zhu Y, Zong T, Xiang Y. A feature selection-based method for DDoS attack flow classification. Futur Gener Comput Syst. 2022;132:67–79. https://doi.org/10.1016/j.future.2022.02.006.
31. Arora K, Aggarwal AK. Approaches for image database retrieval based on color, texture, and shape features. In: Handbook of research on advanced concepts in real-time image and video processing. Hershey: IGI Global; 2018. p. 28–50.
32. Miseikis J, Brijacak I, Yahyanejad S, Glette K, Elle OJ, Torresen J. Multi-objective convolutional neural networks for robot localisation and 3D position estimation in 2D camera images. In: 2018 15th international conference on ubiquitous robots (UR). IEEE; 2018. p. 597–603.
33. Aggarwal AK. Learning texture features from glcm for classification of brain tumor MRI images using random forest classifier. Trans Signal Process. 2022;18:60–3.
34. Xu X, Wang X. An adaptive network intrusion detection method based on PCA and support vector machines. In: Li X, Wang S, Dong ZY, editors. Advanced data mining and applications. Berlin: Springer; 2005. p. 696–703.
35. Liu G, Yi Z, Yang S. A hierarchical intrusion detection model based on the PCA neural networks. Neurocomputing. 2007;70(7–9):1561–8. https://doi.org/10.1016/j.neucom.2006.10.146.
36. Kuang F, Xu W, Zhang S. A novel hybrid KPCA and SVM with GA model for intrusion detection. Appl Soft Comput. 2014;18:178–84. https://doi.org/10.1016/j.asoc.2014.01.028.
37. Abdulhammed R, Faezipour M, Musafer H, Abuzneid A. Efficient network intrusion detection using PCA-based dimensionality reduction of features. In: 2019 international symposium on networks, computers and communications (ISNCC), Istanbul, Turkey. IEEE; 2019. p. 1–6. https://doi.org/10.1109/ISNCC.2019.8909140.
38. 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 Inf. 2022;18(9):6503–11. https://doi.org/10.1109/TII.2021.3139363.
39. Tan Z, Jamdagni A, He X, Nanda P. Network intrusion detection based on LDA for payload feature selection. IEEE Globecom Workshops. 2010;2010:1545–9. https://doi.org/10.1109/GLOCOMW.2010.5700198.
40. Pajouh HH, Dastghaibyfard G, Hashemi S. Two-tier network anomaly detection model: a machine learning approach. J Intell Inf Syst. 2017;48(1):61–74. https://doi.org/10.1007/s10844-015-0388-x.
41. Pajouh HH, Javidan R, Khayami R, Dehghantanha A, Choo K-KR. A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in IoT backbone networks. IEEE Trans Emerg Topics Comput. 2019;7(2):314–23. https://doi.org/10.1109/TETC.2016.2633228.
42. Khan FA, Gumaei A, Derhab A, Hussain A. A novel two-stage deep learning model for efficient network intrusion detection. IEEE Access. 2019;7:30373–85. https://doi.org/10.1109/ACCESS.2019.2899721.
43. Zhou X, Hu Y, Liang W, Ma J, Jin Q. Variational LSTM enhanced anomaly detection for industrial big data. IEEE Trans Ind Inf. 2021;17(5):3469–77. https://doi.org/10.1109/TII.2020.3022432.
44. Popoola SI, 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. 2021;8(6):4944–56. https://doi.org/10.1109/JIOT.2020.3034156.
45. Dao T-N, Lee H. Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection. IEEE Internet Things J. 2022;9(16):14438–51. https://doi.org/10.1109/JIOT.2021.3078292.
46. D'Angelo G, Palmieri F. Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. J Netw Comput Appl. 2021;173: 102890. https://doi.org/10.1016/j.jnca.2020.102890.
47. Ngo V-D, Vuong T-C, Van Luong T, Tran H. Machine learning-based intrusion detection: feature selection versus feature extraction. arXiv; 2023. http://arxiv.org/abs/2307.01570. Accessed 10 July 2023.
48. Moustafa N. A new distributed architecture for evaluating AI-based security systems at the edge: network TON_IoT datasets. Sustain Cities Soc. 2021;72: 102994. https://doi.org/10.1016/j.scs.2021.102994.
49. Kotsiantis SB, Kanellopoulos D, Pintelas PE. Data preprocessing for supervised leaning. Int J Comput Sci. 2007;1(12):6.
50. Guo G. An intrusion detection system for the internet of things using machine learning models. In: 2022 3rd international conference on big data, artificial intelligence and internet of things engineering (ICBAIE), Xi'an, China. IEEE; 2022. p. 332–5. https://doi.org/10.1109/ICBAIE56435.2022.9985800.

51. Gad AR, Nashat AA, Barkat TM. Intrusion detection system using machine learning for vehicular ad hoc networks based on ToN-IoT dataset. IEEE Access. 2021;9:142206–17. https://doi.org/10.1109/ACCESS.2021.3120626.
52. Tan Z, Jamdagni A, He X, Nanda P. Network intrusion detection based on LDA for payload feature selection. In: 2010 IEEE Globecom workshops, Miami, FL, USA. IEEE; 2010. p. 1545–9. https://doi.org/10.1109/GLOCOMW.2010.5700198.
53. Fatani A, Dahou A, Al-Qaness MAA, Lu S, Abd Elaziz MA. Advanced feature extraction and selection approach using deep learning and Aquila optimizer for IoT intrusion detection system. Sensors. 2021;22(1):140. https://doi.org/10.3390/s22010140.
54. Ingre B, Yadav A, Soni AK. Decision tree based intrusion detection system for NSL-KDD Dataset. In: Satapathy SC, Joshi A, editors. Information and communication technology for intelligent systems (ICTIS 2017), vol. 2. Cham: Springer International Publishing; 2018. p. 207–18.
55. Negandhi P, Trivedi Y, Mangrulkar R. Intrusion detection system using random forest on the NSL-KDD dataset. In: Shetty NR, Patnaik LM, Nagaraj HC, Hamsavath PN, Nalini N, editors. Emerging research in computing, information, communication and applications. Singapore: Springer Singapore; 2019. p. 519–31.
56. Almseidin M, Alzubi M, Kovacs S, Alkasassbeh M. Evaluation of machine learning algorithms for intrusion detection system. In: 2017 IEEE 15th international symposium on intelligent systems and informatics (SISY); 2017. p. 000277–82. https://doi.org/10.1109/SISY.2017.8080566.
57. Mukherjee S, Sharma N. Intrusion detection using Naive Bayes classifier with feature reduction. Procedia Technol. 2012;4:119–28. https://doi.org/10.1016/j.protcy.2012.05.017.
58. Amato F, Mazzocca N, Moscato F, Vivenzio E. Multilayer perceptron: an intelligent model for classification and intrusion detection. In: 2017 31st international conference on advanced information networking and applications workshops (WAINA); 2017. p. 686–91. https://doi.org/10.1109/WAINA.2017.134.

Publisher's Note