Neural Computing and Applications
https://doi.org/10.1007/s00521-024-09562-9

REVIEW

Identifying the most accurate machine learning classification technique to detect network threats

Mohamed Farouk[1] · Rasha Hassan Sakr[2] · Noha Hikal[3]

Received: 30 July 2023 / Accepted: 24 January 2024

Abstract

Insider threats have recently become one of the most urgent cybersecurity challenges facing numerous businesses, such as public infrastructure companies, major federal agencies, and state and local governments. Our purpose is to find the most accurate machine learning (ML) model to detect insider attacks. In the realm of machine learning, the most convenient classifier is usually selected after further evaluation trials of candidate models which can cause unseen data (test data set) to leak into models and create bias. Accordingly, overfitting occurs because of frequent training of models and tuning hyperparameters; the models perform well on the training set while failing to generalize effectively to unseen data. The validation data set and hyperparameter tuning are utilized in this study to prevent the issues mentioned above and to choose the best model from our candidate models. Furthermore, our approach guarantees that the selected model does not memorize data of the threats occurring in the local area network (LAN) through the usage of the NSL-KDD data set. The following results are gathered and analyzed: support vector machine (SVM), decision tree (DT), logistic regression (LR), adaptive boost (AdaBoost), gradient boosting (GB), random forests (RFs), and extremely randomized trees (ERTs). After analyzing the findings, we conclude that the AdaBoost model is the most accurate, with a DoS of 99%, a probe of 99%, access of 96%, and privilege of 97%, as well as an AUC of 0.992 for DoS, 0.986 for probe, 0.952 for access, and 0.954 for privilege.

Keywords Machine learning · Insider threats · Insider attacks · NSL-KDD data set · Cybersecurity

Abbreviations

NSL	Network Security Laboratory
KDD	Knowledge Discovery in Databases
ML	Machine learning
DoS	Denial-of-service
LAN	Local area network
SVM	Support vector machine
DT	Decision tree
LR	Logistic regression
AdaBoost	Adaptive boost
GB	Gradient boosting
RFs	Random forests
ERTs	Extremely randomized trees
CV	Cross-validation
CIA	Confidentiality, integrity, availability
HTTP	Hypertext Transfer Protocol
MIT	Massachusetts Institute of Technology
US	United States
DFD	Data flow diagram
TCP	Transmission control protocol
UDP	User datagram protocol
ICMP	Internet control message protocol
SS	Standard scaler
RSA	Random search algorithm
SAG	Stochastic average gradient
CD	Coordinate descent

✉ Mohamed Farouk
mhmd_farouk_50@std.mans.edu.eg

Rasha Hassan Sakr
rashah@mans.edu.eg

Noha Hikal
Dr_nahikal@mans.edu.eg

1 Department of Information Security, Faculty of Computers and Information Sciences, Mansoura University, Mansoura 35516, Egypt

2 Department of Computer Science, Faculty of Computers and Information Sciences, Mansoura University, Mansoura 35516, Egypt

3 Department of Information Technology, Faculty of Computers and Information Sciences, Mansoura University, Mansoura 35516, Egypt

MSE	Mean squared error
SAMME	Stagewise additive modeling with a multi-class exponential
SAMMER	Stagewise additive modeling with a multi-class exponential real
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
UFS	Univariate feature selection
ANOVA	Analysis of variance
RFE	Recursive feature elimination
TP	True positive
TN	True negative
FP	False positive
FN	False negative
TPR	True-positive rate
FPR	False-positive rate
Acc	Accuracy
Rec	Recall
CM	Confusion matrix
AUC	Area under the curve
ROC	Receiver operator characteristic

1 Introduction

Insiders, such as employees, have legal access to an enterprise's resources in order to perform their job duties; as a result, detecting insider threats is one of the most difficult challenges facing security administrators and makes it difficult to identify these internal threats [1, 2]. That is why this study employed a variety of supervised machine learning classifiers with specific criteria to find the most accurate classifier to predict these insider threats, mainly LAN attacks from the NSL-KDD data set [3–5].

According to [6], 94% of firms have had insider data breaches in the last 12 months, and 84% have encountered security difficulties caused by nontechnical errors (Insider Data Breach Survey, 2021). Humans are the leading cause of disastrous insider data breaches. Therefore, malicious insiders are the major concern of department heads, with 28% agreeing to the previous statement.

[7] published a report stating that insider threat occurrences increased by 44% over the last two years, with a cost climb of more than a third to USD 15 million (Cost of Insider Threats Global Report, 2022). In addition, the cost of corporate credential theft rose by 65% since 2020, from USD 2 million to USD 4 million today. Furthermore, the time to contain an insider threat incident increased from 77 to 85 days, implying that organizations spent more on containment operations. When issues take more than 90 days to settle, communities incur an average yearly cost of USD 17 million [5].

The data breach attacks are classified into different categories [8]: passive attacks, active attacks, close-in attacks, insider attacks, and distribution attacks. Insider attacks are among the most significant threats to information systems because of their impact on confidentiality, integrity, and availability (CIA), especially if they occur on a LAN. These attacks can impact businesses, reputations, and finances [9].

The purpose of the study is to find the most accurate classifier for identifying insider attacks that occur on LANs. Additionally, the significance of the study lies in locating irregular 'attacked' LAN traffic by developing a Python code that uses scikit-learn for backend machine learning, then by plotting the charts with the Plotly open-source, Seaborn, and Matplotlib frameworks. To eliminate bias, a random search algorithm (RSA) is used to tune the hyperparameter using K-fold and stratified cross-validation methods to avoid overfitting.

This study is divided into four sections. Section one and two summarizes related articles and previous studies. Section 3 discusses the proposed framework. Finally, Sect. 4 analyzes the study's findings.

1.1 Tuning hyperparameters and risk minimization

Hyperparameters, which are also known as nuisance parameters, are values that must be specified outside of the training procedure. A regularization hyperparameter is a process to determine the optimal hyperparameter to send as input to the estimator, such as the decision tree classifier's criteria and maximum depth values. It also indexes the method in many learning issues. Because the optimum hyperparameter for one data set is not always the best for other data sets, the settings must be adjusted for each task. Before evaluating potential estimators, the hyperparameter must be modified to decrease the expected risk [10–12].

1.2 Avoid overfitting and model selection

Using a test data set in the model selection procedure can introduce an overfitting problem due to unseen data leaking into the model, which depends on the model selection procedure on the best evaluation metrics. In addition, utilizing the training data set in the model test performance will also cause an overfitting problem. The overfitting model produces inaccurate predictions and cannot handle and generalize all forms of new input (unseen data). As a result, the model may become useless [13–16].

As a solution, a technique called cross-validation (CV) is employed to mitigate overfitting. It is a powerful tool for developing and selecting ML models. Not only does it ensure that the data is suitable for the dependability of used classifiers, but it also prevents the need to split the data set. So, we are not having the underfitting issue caused by data division, a lack of samples, or insufficient learning of the model [13, 14].

Cross-validation randomly separates the training set into two logical parts: a training set and a validation set. When the existing test set, as in our NSL-KDD data set, is added, we will have three sets in total. Each set serves a different purpose: the training set is used to teach the model, the validation set is used to solve the above issues such as choosing the best model and generalizing to new data, and the test set is used to evaluate the model's performance [13, 14].

1.3 Background of the study

ML has proven to be the ideal solution for situations like anomaly detection and network intrusion detection [17, 18]. Therefore, supervised ML algorithms are used to solve the problem of the study due to their speed of response in detecting threats. Supervised ML algorithms are divided into two types [19]: classification algorithms and regression algorithms. First, classification algorithms address this issue since they can distinguish between two or more classes (normal, attack), as in the framework, the outcomes predicted are discrete class labels [20–22]. The following are the supervised ML classification algorithms: linear support vector machines (SVMs), decision trees (DTs), and logistic regression (LR).

In addition to the ensemble algorithms, they aid in solving both classification and regression problems. The goal of ensemble techniques is to combine many prediction models and enhance the outcomes. The following are the supervised ML classification ensemble algorithms: adaptive boosting (AdaBoost), gradient boost (GB), extremely randomized trees (ERTs), and random forests (RFs) [22].

Insiders exhaust an organization's resources significantly, resulting in huge financial and human losses. Because of this, insider activities on a LAN must be recognized and their impact on security politics (CIA) must be identified. Due to the various motivations of the insider which can be personal, political, or economic [17, 23, 24], a plan must be prepared to detect all possible disasters and security concerns. As a result, the study aims to immediately characterize these risks on a LAN and instantly support the security administrators by identifying the most accurate ML classifier.

There is a need to comprehend the link between insiders and their threats. Insiders can gain access rights to networks, either legally or illegally. Legally, various departments can get access to each other because of variances across departments, joint ventures, outsourcing, and the potential of recruiting temporary employees such as consultants. Thus, there are certainly different levels of authorization granted to these insiders [5]. Their threats involve the misuse of legitimate access rights. However, there are several types of insiders. Each type has its own procedures, risks, and data sets. The NSL-KDD data set targets anyone connected either internally or remotely to a LAN [4, 17].

2 Related articles

This section discusses the previous studies and articles related to the study at hand. In [25], the theoretical obstacles to detecting insider threats are addressed, which helped define the research topic. The study also lists the existing insider threat data set types that include emails, authentication, login, HTTP, and files but exclude insider attacks. Consequently, the importance of our research contributions is highlighted. [26] proves that a one-hot-encoding approach is capable of converting categorical features into new individual binary features to train the classification models on them. [3] offers a review of existing insider threat approaches that use NSL-KDD to detect DOS attacks. [27] provides context-specific definitions of ML model hyperparameters as well as their impact on tuning model hyperparameters for decision-making performance and various approaches to obtain optimal values. In [28], the sensitivity of hyperparameter adjustment to eliminate bias in performance prediction is explored. The researchers carry out a detailed investigation, but the results are unsatisfactory, and the classifier cannot be generalized to another test data set. As a result, there is a need to conduct additional research to locate the best classifier during the creation of a machine learning system The problems of both overfitting and underfitting are presented in [14] along with their impact on the performance model for decision-making, as well as cross-validation methods as a solution. [29] extended and simplified significant CV approaches in developing a final model based on ML. The researcher stresses the importance of generalizing to unseen data to maximize the potential of predictive models and avoid overfitting. Consequently, the researcher concludes that generalizing to unseen data cannot be overlooked and should not become only limited to training and testing to build models and extract results. [23] explored an in-depth examination of the NSL-KDD data set. The study also analyzes the issues found in the kdd99 data set in addition to evaluation metrics. [17] presents a comprehensive review and in-depth understanding of insider threats based on previously published articles and statistical data on both insiders and methodologies employed to detect them. However, the reported results of supervised machine learning are disappointing. Furthermore, most articles in the review focus on outside attacks from emails, HTTP, illegal file access, and devices while ignoring dangers from within the network. In contrast, this study is concerned about LAN insider attacks since they are more common, easily motivated, and cause maximum damage. [30] details the NSL-KDD data set features as well as both the concerns observed in kdd99 and the attack type classifications. The researchers in [27] conduct a survey assessment of insider threat concerns. They state that the extent of the insider threat is a complicated challenge since it is usually difficult to distinguish between insiders and outsiders of a community while operating within a LAN. Furthermore, some insiders can initiate attacks from the outside, for example, an employee who left the organization. The article discusses the challenge of identifying internal attacks that took place on the internal network. Therefore, the main motive behind the current study is to find the most accurate classifier that identifies these threats correctly. [19] demonstrated supervised machine learning methods and the significance of classification models in classifying anomalous behavior from ideal traffic. [31] highlights the insider threat aspects and the methods to confront these threats using either machine learning or non-machine learning techniques. In [32], it is shown that the restoration of missing values in data sets uses zero as a solution.

After surveying the above-listed studies, we conclude that ML techniques are the best solution for insider threat identification. Consequently, one of the reasons behind conducting our research is to locate the best ML technique to address the challenge of identifying insider threats.

3 Case study

In this section, the study focuses on clarifying the methodology utilized in this research paper. Figure 1 depicts the process framework which consists of seven stages: (1) collect data set, (2) preprocessing, (3) tuning models, (4) feature selection, (5) avoid overfitting, (6) training models, and (7) final evaluation. In the following subsections, each stage is explained in detail.

3.1 Data set description

MIT Lincoln Labs developed and managed the 1998 DARPA intrusion detection evaluation program, and built a LAN that simulated a US Air Force LAN, conducted several attacks, and gathered raw TCP dump data. Data flowed from the source IP address to the destination IP address depending on a specific protocol to distinguish between normal and malicious connections [13, 17]. A connection was defined as a sequence of TCP packets transmitted at certain times. Afterward, MIT Lincoln Labs extracted the features from raw DARPA and packaged them into the first ready-to-use version, known as KDD99. However, more issues were discovered in the KDD99 data set [30, 33, 34]. A lot of these issues were solved in the updated NSL-KDD data set such as removing redundant records which lead to reducing the size of training and test sets and doing experiments easier and faster [21, 30, 35].

3.2 Data set analysis

The NSL-KDD data set contains 41 features and is divided into two files: the training data set file and the test data set file. On one hand, the test data set file has 125,973 entries, and on the other hand, the test data set file has 22,544 records [30].

As shown in Fig. 2, the Python code retrieved by Pandas, a data analysis tool, classifies the feature data types into an object (nominal), int64, and float64 [13, 19, 23]. Figure 2 also displays the counting and variation in the unique values among the nominal features in the training and test data sets. Finally, Fig. 2 illustrates the service feature in the training data set equals 70 unique values, whereas the service feature in the test data set equals 64 unique values. In the preprocessing phase, the researcher tries to tackle this issue.

The class label contains five main categories of classifications [21, 30, 33]:

i. Normal: normal connections.
ii. DoS: denial-of-service, for example, Smurf.
iii. Probing: surveillance, such as port sweep.
iv. Access: unauthorized remote machine access, e.g., spying.
v. Privilege: unauthorized access to local superuser (root) privileges, e.g., Rootkit.

The probability distribution of the training data set is different from the test data set. The test data set should contain a lot more attacks than the training set just until the estimators can predict new offensives and the system can simulate reality [21, 30, 33]. Figure 3 shows the class label sizes in the NSL-KDD training data set, whereas Fig. 4 illustrates the class label sizes in the NSL-KDD test data set.

3.3 Data preprocessing

This stage is one of the most critical phases in the machine learning approach. Figure 5 exhibits the data flow diagram (DFD) for the data preprocessing procedure. As shown in Fig. 5, we first separate the numerical and categorical features. Then, we repair the missing in-service feature between the training and test sets. Afterward, we apply

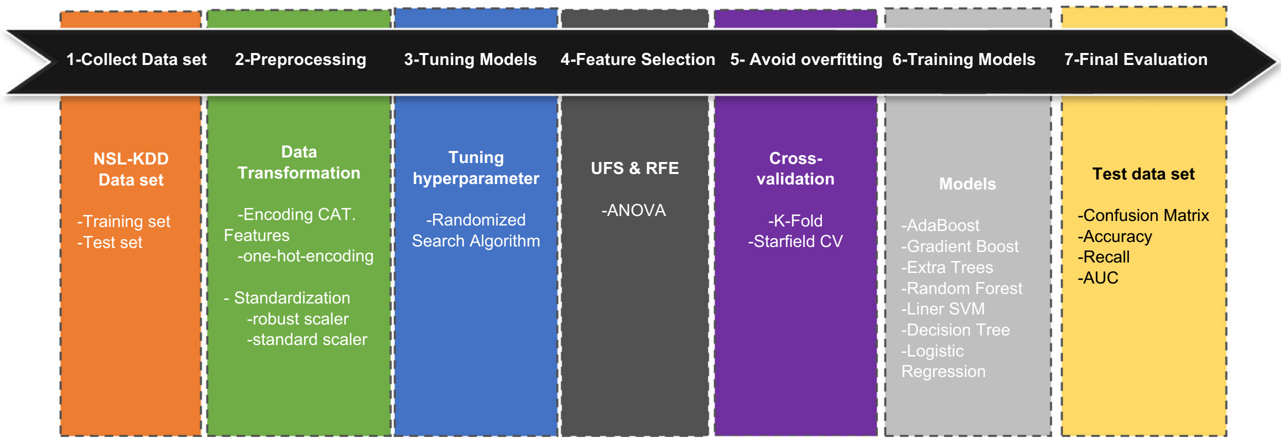


Fig. 1 The process framework

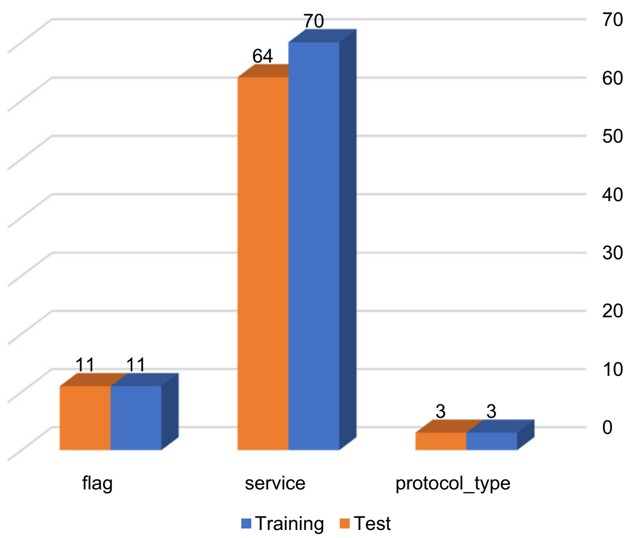


Fig. 2 Unique values of the nominal features

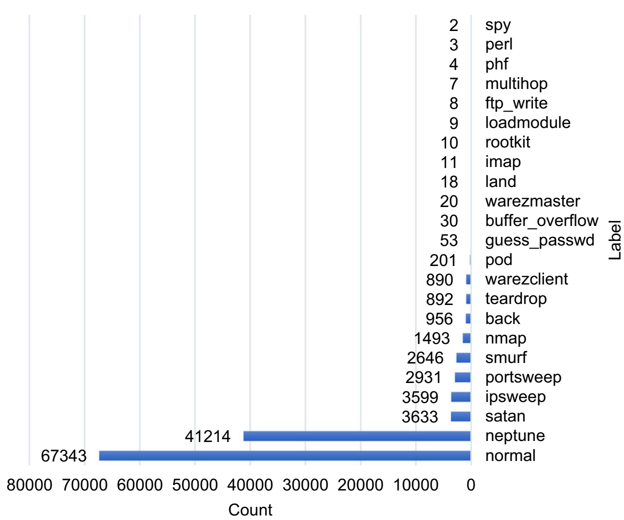


Fig. 3 The count of each class label in the training set

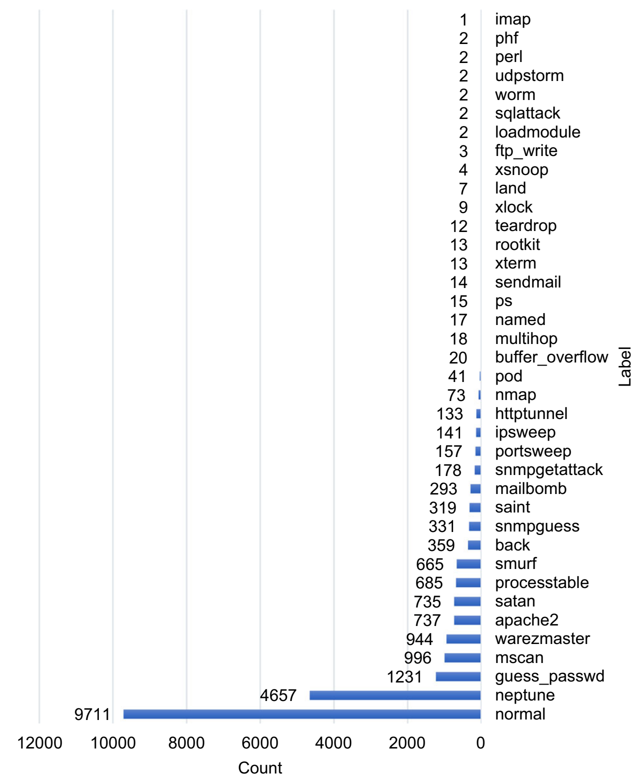
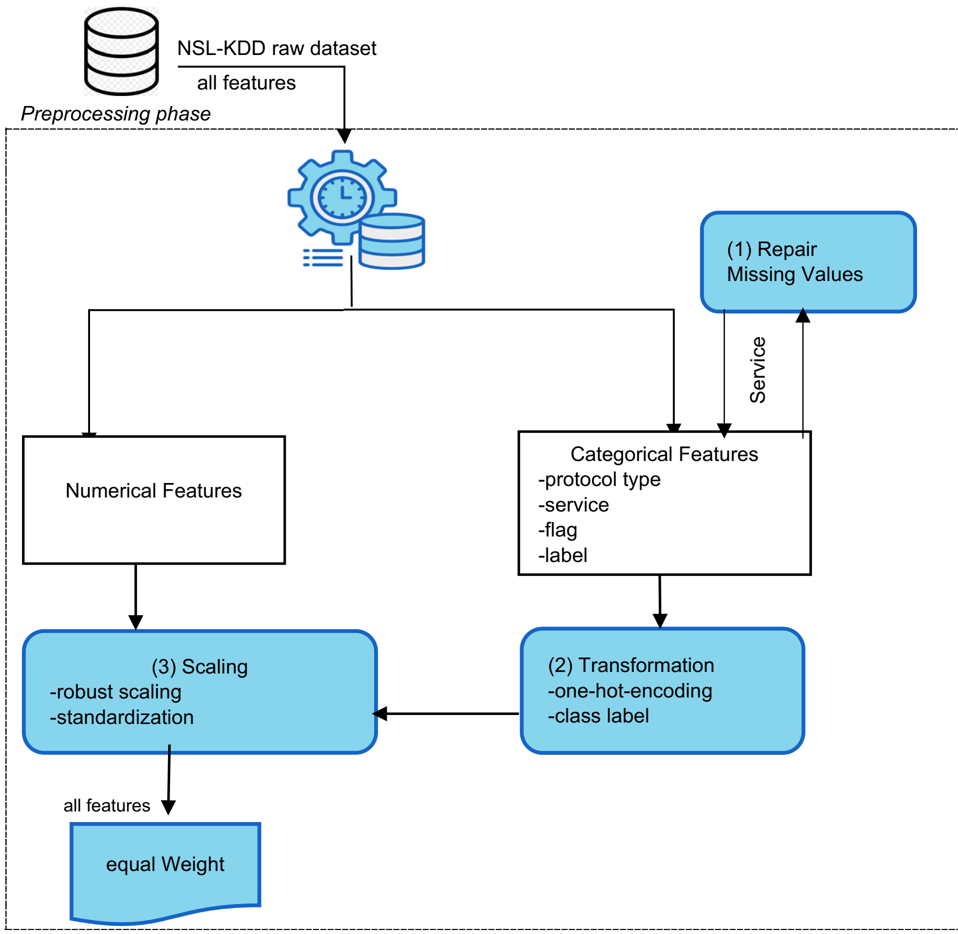


Fig. 4 The size of each class label in the test data set

transformational techniques to the categorical features. Finally, we perform scaling methods on all features before recombining them. The preprocessing principle attempts to transform the raw data set into a beneficial format while also ensuring that the data set is clean and noise-free, as a result, the estimator's decision is not affected [32]. The following section describes the preprocessing methods applied to the training and test data sets:

Fig. 5 DFD for the data preprocessing stage



3.3.1 Data transformation

After checking the purity of the data, transformation techniques are applied because most machine learning models do not accept categorical features. First, categorical features are converted into numbers, a process known as 'encoding' the category features [32]. There are four categorical features in the NSL-KDD data set, namely protocol type, service, flag, and class label. After that, all the features are standardized by assigning them the same weight until the classifiers are not able to choose the values based on the greater weight. The data transformation methods used are the following: one-hot-encoding and class label.

One-hot-encoding is also known as dummy encoding. This process converts categorical features into binary. This process is carried out in two stages [26]. First, the unique values of the categorical features are transformed into new binary features. Then, the feature's unique value that the connection detected is assigned the value of 1, and the remainder is assigned the value of 0. This methodology was only used for categorical features [36, 37], namely protocol type, service, and flag. The class label was handled otherwise.

The protocol type, service, and flag features, which are displayed in Fig. 2, denote that there is a striking difference between the service feature offered by the training data set on one hand and the testing data set on the other. To equate the testing data set with the training data set, a zero value is used to compensate for the missing values [32, 38]. Table 1 exhibits samples of the protocol-type feature after applying dummy encoding.

In addition to the 38 original features [39], the data set has been increased to include a total of 122. The added features are three protocol-type features, 70 service features, and 11 flag features. Table 2 presents the complete number of features after encoding.

The class label contains sub-attacks that fall within the scope of five main categories, namely DoS, probe, access, privilege, and a normal connection. Each of these attacks is

Table 1 An extract of the protocol-type feature

Original feature			
Protocol type	Protocol type ICMP	Protocol type TCP	Protocol type UDP
TCP	0	1	0
ICMP	1	0	0
TCP	0	1	0
TCP	0	1	0
TCP	0	1	0
UDP	0	0	1
UDP	0	0	1

converted into a unique integer in the same class label column, as shown in Table 3. After converting all categorical attributes to integers, each group is handled individually. As for normal connections, traffic is introduced to each of them, allowing the models to differentiate between regular and irregular attacks or connections. Figure 6 depicts the magnitude of each attack type as well as normal traffic in both the training and test data sets.

3.3.2 Scaling

The feature scale aims to place all features on the same scale, indicating that all features are equally important [13]. Figure 7 depicts the data set before any scaling is applied. The approaches listed below are used. Scaling uses the following: robust scaler and standardization.

First, robust scaler removes outliers by eliminating the median and scaling the data based on the quantile range [13, 40]. Figure 8 depicts the data set after applying a robust scaler. The following formula (1) is used:

$$X_{\text{new}} = \frac{Xi - X_{\text{median}}}{IQR(Q3(x) - Q1(x))} \tag{1}$$

where: X_{new}*:* Standardized value*,* Xi*:* Original values, X_{median}*:* sample median*,* $Q1$*:* 1st quartile, and $Q3$*:* 3rd quartile.

Second, standardization in Python is called a standard scaler (SS). It specifies that the standard deviation is equal to 1, and the mean of the values changes to 0 [13, 41]. Figure 9 displays data after applying SS. The (Z score) Eq. (2) determines this SS:

$$Z = \frac{x - \mu}{\sigma} \tag{2}$$

where: X: values, μ*:* mean, and σ*:* standard deviation.

3.4 Tuning model

The tuning process is a matter of trial and error. The statistical ML model experiments repeatedly with different hyperparameter values [13, 14]. After that, its efficiency is compared to the validation set to determine which set of hyperparameter results in the most accurate model [6]. The main technique used in tuning the model is known as RSA.

RSA defines for each hyperparameter a statistical distribution from which values are randomly picked and utilized to train the model. This step increases the likelihood of quickly determining practical prime values for each hyperparameter [6, 12]. The following Table 4 depicts the results of the RSA for the optimal hyperparameter of each model, the best hyperparameter affecting the decision model process, the hyperparameter datatype, the hyperparameter default values at which each model was operating, the start–end random values, and finally the chosen optimal values for each hyperparameter.

In the following paragraphs, we describe the mathematical functions of hyperparameters in our models. First, the linear SVM model employs two equations to determine the loss hyperparameter. A hinge is a form of the cost function in which a margin or distance from the classification border is defined according to the following Eq. (3) and the squared hinge by Eq. (4) [42, 43], where t is the actual result, either 1 or 0.

$$\text{hinge}(y) = \max(0, 1 - t.y) \tag{3}$$

$$\text{squared hinge}(y) = \max(0, 1 - t.y)^2 \tag{4}$$

Second, the criterion hyperparameter contains three arguments in the DT model: Gini, entropy, and log loss. The equations are as follows [14, 32, 43, 44]:

The Gini determines the splitting for each feature and quantifies the impurity of (D). The following formula (5) determines Gini:

Table 2 All the features after encoding

Original features	Protocol type	Service	Service cont	Flag
F1: Duration	F39: Protocol_type_Icmp	F42: Service_IRC	F80: Service_netbios_Ns	F112: Flag_OTH
F2: Src_bytes	F40: Protocol_type_Tcp	F43: Service_X11	F81: Service_Netbios_Ssn	F113: Flag_REJ
F3: Dst_bytes	F41: Protocol_type_Udp	F44: Service_Z39_50	F82: Service_netstat	F114: Flag_RSTO
F4: Land		F45: Service_Aol	F83: Service_Nnsp	F115: Flag_RSTOS0
F5: Wrong_fragment		F46: Service_Auth	F84: Service_Nntp	F116: Flag_RSTR
F6: Urgent		F47: Service_Bgp	F85: Service_Ntp_U	F117: Flag_S0
F7: Hot		F48: Service_Courier	F86: Service_Other	F118: Flag_S1
F8: Num_failed_logins		F49: Service_Csnet_Ns	F87: Service_Pm_Dump	F119: Flag_S2
F9: Logged_in		F50: Service_Ctf	F88: Service_Pop_2	F120: Flag_S3
F10: Num_compromised		F51: Service_daytime	F89: Service_Pop_3	F121: Flag_SF
F11: Root_shell		F52: Service_discard	F90: Service_printer	F122: Flag_SH
F12: Su_attempted		F53: Service_domain	F91: Service_private	
F13: Num_root		F54: Service_domain_U	F92: Service_red_I	
F14: Num_file_creations		F55: Service_echo	F93: Service_remote_Job	
F15: Num_shells		F56: Service_Eco_I	F94: Service_Rje	
F16: Num_access_files		F57: Service_Ecr_I	F95: Service_Shell	
F17: Num_outbound_cmds		F58: Service_Efs	F96: Service_Smtp	
F18: Is_Host_login		F59: Service_Exec	F97: Service_Sql_Net	
F19: Is_Guest_login		F60: Service_Finger	F98: Service_Ssh	
F20: Count		F61: Service_Ftp	F99: Service_sunrpc	
F21: Srv_count		F62: Service_Ftp_Data	F100: Service_supdup	
F22: Serror_rate		F63: Service_Gopher	F101: Service_systat	
F23: Srv_serror_rate		F64: Service_Harvest	F102: Service_telnet	
F24: Rerror_rate		F65: Service_Hostnames	F103: Service_Tftp_U	
F25: Srv_rerror_rate		F66: Service_Http	F104: Service_tim_I	
F26: Same_Srv_rate		F67: Service_Http_2784	F105: Service_time	
F27: Diff_Srv_rate		F68: Service_Http_443	F106: Service_Urh_I	
F28: Srv_diff_host_rate		F69: Service_Http_8001	F107: Service_Urp_I	
F29: Dst_host_count		F70: Service_Imap4	F108: Service_Uucp	
F30: Dst_host_Srv_count		F71: Service_Iso_Tsap	F109: Service_Uucp_Path	
F31: Dst_Host_Same_Srv_Rate		F72: Service_Klogin	F110: Service_Vmnet	
F32: Dst_host_diff_Srv_rate		F73: Service_Kshell	F111: Service_whois	
F33: Dst_host_same_Src_port_rate		F74: Service_Ldap		
F34: Dst_host_srv_diff_host_Rate		F75: Service_link		
F35: Dst_host_serror_rate		F76: Service_login		
F36: Dst_Host_Srv_Serror_Rate		F77: Service_Mtp		
F37: Dst_host_rerror_rate		F78: service_Name		
F38: Dst_host_srv_rerror_rate		F79: Service_Netbios_Dgm		

*F: Feature

Table 3 The class labels categories to unique integers

Category	Class label
Normal connection	0
DoS	1
Probe	2
Access	3
Privilege	4

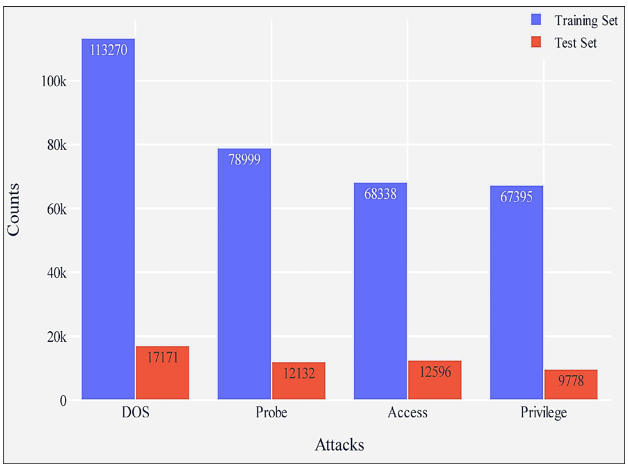


Fig. 6 The size of each group of attacks in the NSL-KDD data set

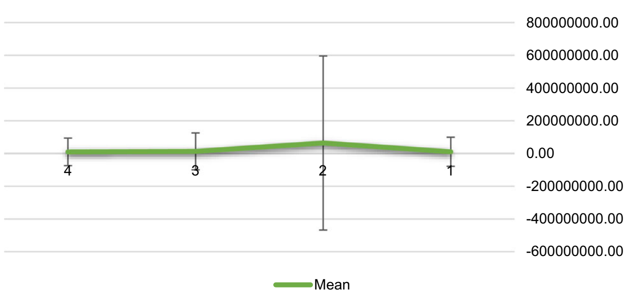


Fig. 7 Unscaled data

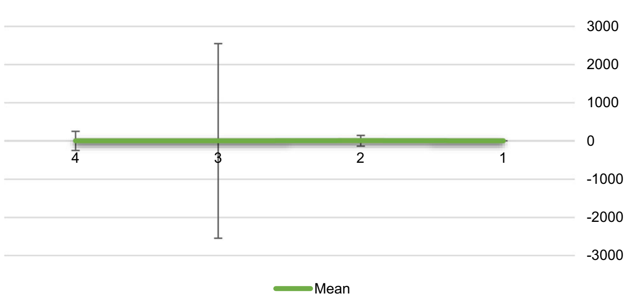


Fig. 8 Data after applying robust scaling

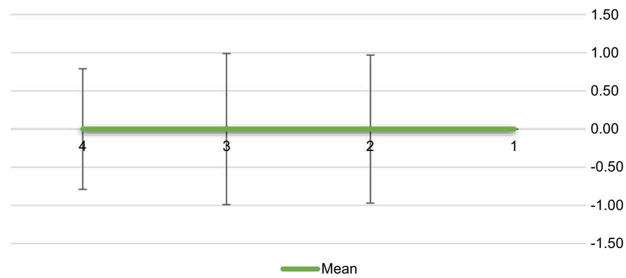


Fig. 9 The data set's deviation shape after applying the standard scaling

$$\mathrm{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{5}$$

where: p_i is the probability that a tuple in $D \in C_i$ and is estimated by $|C_{i,D}|/|D|$. The total is calculated across m classes.

Entropy is a metric of information that is used to evaluate the impurity or uncertainty in a set of data. It controls how a decision tree splits data. p_x indicates the probability of the x the class in the data set D, where $x = 1, 2, \ldots, n$. The following formula (6) is used to compute entropy:

$$\text{Entropy} : H(D) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{6}$$

Log loss is employed when predicting whether a Boolean (true or false) is something with a likelihood range from certainly true (1) to obviously false (0). The log loss formula (7) is defined as:

$$\mathrm{logloss} = -1/N \sum_{i=1}^{N} (\log(Pi)) \tag{7}$$

where: N is the number of instances, and Pi is the model likelihood.

Third, the solver hyperparameter in the LR model has five approaches [45]. To begin with, Newton's approach employs a quadratic function around (xn) to approximate $f(x)$ in each iteration [46, 47]. Then, limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS): limited memory refers to keeping only a few vectors and uses an inverse hessian matrix that is estimated using gradient evaluation-specified updates [46]. In addition, the library for large linear classification (Lib-linear) employs a coordinate descent (CD) approach to solve optimization issues by executing sequential approximation reduction along coordinate directions [48]. Furthermore, stochastic average gradient descent (SAG) is an iterative approach for optimization by gradient descent and incremental aggregated gradient technique modification that uses a random sample of prior gradient values and is suitable for large data sets since it can be handled quickly [49, 50]. Finally,

Table 4 Optimum values of hyperparameters for the models

Model	Hyperparameter	Type	Default	Strat	End	RSA
Linear SVM	C,	Float	1.0	0.5	1.6	1.5
	Intercept_scaling,	Float	1.0	1.0	6.0	3.0
	Loss,	Nominal	squared_hine	(3) (4)	–	hinge
	Max_iter,	Int	100	@100	130	100
	Verbose	Int	0	0	5	3
DT	Class_weight	Nominal	None	–	–	balanced
	Criterion,	Nominal	gini	(5) (6) (7)	–	entropy
	Max_depth,	Int	none	10	30	20
	Max_leaf_nodes,	Int/float	none	10	20	10
	Min_samples_leaf	Int/float	2	1	4	2
LR	C,	Float	1.0	0.5	1.6	1.55
	Intercept_scaling,	Float	1.0	1.0	6.0	4.0
	Max_iter,	Int	100	100	130	110
	Solver,	Nominal	Lbfgs	–	–	liblinear
	Verbose	Int	0	0	5	5
GB	Criterion,	Nominal	friedman_mse	(8) (9)	–	squared_error
	Learning_rate,	Float	0.1	0.0	0.3	0.2
	Max_depth,	Int	3	1	5	1
	Max_features,	Nominal	None	(10) (11)	–	Log2
	Min_samples_leaf,	Int/float	1	1	3	2
	Min_samples_split,	Int/float	2	2	4	3
	Min_weight_fraction_leaf,	Float	0.0	0.0	0.4	0.1
	Verbose,	Int	0	0	4	3
	Warm_start	Bool	False	–	–	True
AdaBoost	Algorithm,	Nominal	SAMME.R	(12) (13)	–	SAMME
	Learning_rate,	Float	1.0	0.0	2.0	0.1
	N_estimators	Int	50	30	80	70
RFs	Max_depth,	Int	None	10	30	20
	Min_samples_leaf,	Int/Float	1	1	4	3
	Min_samples_split,	Int/Float	2	2	6	4
	Min_weight_fraction_leaf,	Float	0.0	0.0	0.3	0.1
	Warm_start	Bool	False	–	–	True
ERTs	Criterion,	Nominal	Gini	(6)	–	Entropy
	Max_depth,	Int	None	10	30	20
	Max_leaf_nodes,	Int/Float	None	10	25	20
	Min_samples_leaf,	Int/Float	1	1	5	4
	Min_samples_split,	Int/Float	2	2	6	5
	Warm_start	Bool	False	–	–	True

SAGA is an extension of SAG that considers the improved version to have a quicker convergence than SAG [49, 50].

Fourth, the purpose of the criterion hyperparameter in the GB model is to evaluate the quality of a data split. The criterion hyperparameter includes 'friedman_mse' for mean squared error (MSE) with Friedman improvement score and 'squared_error' for mean squared error. The 'friedman_mse' Eq. (8) and MSE Eq. (9) are defined as [51, 52]:

$$\text{Friedman\_mse} : i^2(R_l, R_r) = \frac{w_l w_r}{w_l + w_r}(\bar{y}_l - \overline{y_r})^2 \tag{8}$$

where: W_l is the sum of weight for the left part, W_r is the sum of weight for the left part, and $\bar{y}_l$ and $\overline{y_r}$ are the mean left and right.

$$\mathrm{MSE} = \frac{\sum (y_i - p_i)^2}{n} \tag{9}$$

where: y_i is the i^{th} observed value, p_i is the corresponding predicted value, and n is the number of observed values.

Fifth, the 'max_features' hyperparameter utilized in the GB model is the maximum number of features that are permitted on each individual tree. In the first selection, Sqrt will take the square root of the overall number of features. The sqrt Eq. (10) realized as follows [51]:

$$\text{max\_features} = \text{sqrt}(n\text{\_features}) \tag{10}$$

Another option is $\log_2$, which will take $\log_2$ of the number of features. The $\log_2$ Eq. (11) realized as follows:

$$\text{max \_features} = \log_2(n\text{\_features}) \tag{11}$$

Sixth, the algorithm hyperparameter for AdaBoost Classifier offers two options: 'SAMME.R' and 'SAMME.' SAMME is an acronym for stagewise additive modeling with a multi-class exponential loss function and R is an acronym for real. For each weak learner, SAMME employs a separate set of 'decision influence' weights (alphas). SAMME.R, on the other hand, allocates an equal weight to each weak learner and evaluates the class likelihood, which usually converges faster than SAMME [53–56]. The SAMME and SAMME.R EQs (12) and (13) are defined as:

$$\text{SAMME}: H(x) = \arg\max_k \sum_{t=1}^{T} \alpha_t I(h_t x = k) \tag{12}$$

$$\text{SAMME.R}: H(x) = \arg\max_k \sum_{t=1}^{T} s_k^t(x) \tag{13}$$

where: $H(x)$ classification predictions, T weak learners, α_t weight for weak learner t, $h_t x$ the prediction of weak learner t, and s_k^t as a multiplier.

3.5 Features selection

This section lists the features that are used throughout the training models. The following approaches are employed:

3.5.1 Univariate feature selection (UFS)

It is a statistical method that exploits the discrepancy between the qualities until a threshold value is obtained from them. This threshold value is used to determine the real features through the recursive feature elimination method utilized to train the models [19, 22, 57].

The 'f_classif' function, named in the scikit-learn ML framework, finds variance using univariate statistical tests that rely on the analysis of variance (ANOVA) F value [19, 22, 58]. It computes the overall comparison error and finds a greater F value when the variance between groups is less than within the groups, indicating a higher likelihood that the observed difference is actual rather than random. Consequently, it excludes features that differ in variance and selects features with the same variance. This technique has picked 13 features for each attack category, and then the recursive feature elimination approach has used number 13 as the threshold. The F-statistic EQs (14) and (15) in one-way ANOVA are represented as follows [19, 22, 58]:

$$F = \frac{\text{between - groups variance}}{\text{within - group variance}} \tag{14}$$

$$F = \frac{\text{MSGroups}}{\text{MSError}} = \frac{\frac{\text{SSGroups}}{I-1}}{\frac{\text{SSError}}{nT-I}} \tag{15}$$

where: MS: mean square, SS: sum of squares, I: number of groups, and nT: sample size.

3.5.2 Recursive Feature Elimination (RFE)

It is a type of wrapper used for feature selection algorithms. The RFE seeks to identify acceptable feature subsets. It operates immediately following the UFS technique. First, each model is implemented individually using tuned hyperparameters. Then, all features are passed to establish their relevance to one another. Ultimately, the least important features are pruned. The RFE recursively continues this technique on the reduced set until it obtains the requisite feature count defined as the threshold by the UFS method [19, 22, 57]. Table 5 reports the select features of each model using the RFE method for each attack category based on UFS's threshold.

3.6 Cross-validation

As previously indicated, cross-validation [13, 14, 22] is an efficient instrument for designing and choosing ML models. It is employed in the study to avoid overfitting. The following methods, which are part of cross-validation, are used in the study.

3.6.1 K-Fold CV

The original training data set is divided into equal-sized folds (K subsamples) with random sampling. The model is trained using the fold by (K-1) as training data and then verified using the remaining folds. It entails repeating and recording the arithmetic mean and standard deviation of the k-folds produced from the evaluation measures on the various partitions [6, 22]. The following Table 6, 7, 8, 9, 10, 11, and 12 show the outcomes (accuracy, recall, and area under the curve) of the K-fold CV mean and ± standard deviation between folds for each model, with the better results highlighted in bold in Table 10.

Table 5 Select features of each model for each attack

Method	Model	Attacks	Selected Features	No. of Feature
UFS&RFE	Linear SVM	DoS	F1, F5, F7, F10, F14, F19, F21, F35, F37, F40, F41, F42, F92	13
		Probe	F9, F13, F20, F24, F26, F29, F31, F33, F37, F57, F67, F87, F92	
		Access	F9, F13, F20, F29, F30, F31, F33, F38, F41, F42, F43, F97, F114	
		Privilege	F1, F2, F7, F12, F14, F19, F21, F28, F29, F30, F31, F35, F119	
	DT	DoS	F2, F6, F7, F8, F9, F20, F26, F31, F58, F67, F68, F69, F70	13
		Probe	F2, F3, F7, F8, F9, F32, F33, F37, F63, F67, F92, F121, F122	
		Access	F9, F11, F20, F30, F32, F41, F67, F72, F73, F74, F80, F96, F97	
		Privilege	F3, F7, F8, F9, F10, F13, F26, F29, F30, F31, F32, F67, F68	
	LR	DoS	F1, F5, F7, F14, F19, F20, F22, F23, F40, F41, F42, F92, F118	13
		Probe	F9, F20, F21, F24, F28, F29, F31, F33, F37, F55, F57, F67, F92	
		Access	F9, F13, F20, F26, F27, F29, F30, F31, F41, F42, F67, F97, F114	
		Privilege	F1, F2, F7, F19, F20, F21, F24, F28, F31, F35, F63, F87, F103	
	GB	DoS	F2, F3, F5, F10, F20, F23, F26, F31, F34, F35, F58, F67, F122	13
		Probe	F2, F3, F30, F31, F32, F33, F34, F37, F41, F57, F61, F67, F92	
		Access	F1, F2, F3, F7, F8, F19, F30, F32, F33, F34, F35, F63, F71	
		Privilege	F2, F3, F6, F7, F10, F11, F12, F14, F15, F29, F30, F33, F63	
	AdaBoost	DoS	F2, F3, F7, F20, F21, F22, F30, F36, F37, F40, F58, F92, F118	13
		Probe	F1, F2, F21, F26, F30, F31, F32, F33, F34, F37, F41, F67, F92	
		Access	F1, F2, F3, F7, F20, F29, F31, F32, F33, F34, F35, F67, F71	
		Privilege	F1, F2, F3, F11, F12, F14, F29, F30, F35, F38, F63, F87, F103	
	RFs	DoS	F2, F3, F7, F8, F24, F33, F37, F61, F63, F67, F92, F97, F122	13
		Probe	F2, F3, F7, F8, F24, F33, F37, F61, F63, F67, F92, F97, F122	
		Access	F1, F2, F3, F7, F8, F13, F30, F31, F32, F33, F34, F38, F63	
		Privilege	F1, F2, F3, F7, F11, F14, F15, F29, F30, F31, F33, F37, F63	
	ETs	DoS	F2, F5, F9, F20, F26, F31, F35, F36, F58, F67, F117, F118, F122	13
		Probe	F2, F24, F26, F29, F30, F31, F32, F33, F37, F57, F92, F117, F122	
		Access	F1, F2, F3, F7, F19, F20, F21, F29, F30, F31, F33, F34, F63	
		Privilege	F2, F3, F7, F11, F12, F14, F15, F20, F29, F30, F31, F33, F63	

Table 6 Results of K-Fold CV for linear SVM model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
Linear SVM	DoS	97 (± 0.00296)	97 (± 0.00257)	0.998 (± 0.00106)
	Probe	98 (± 0.00320)	98 (± 0.00314)	0.996 (± 0.00095)
	Access	95 (± 0.01605)	95 (± 0.01660)	0.982 (± 0.00485)
	Privilege	99 (± 0.00072)	76 (± 0.21124)	0.983 (± 0.05011)

Table 7 Results of K-Fold CV for DT model

Classifier	Attack	Measure		
		Acc. (%)	Rec. (%)	AUC
DT	DoS	99 (± 0.00092)	99 (± 0.00105)	0.999 (± 0.00022)
	Probe	99 (± 0.00115)	99 (± 0.00292)	0.995 (± 0.00142)
	Access	91 (± 0.00914)	95(± 0.00548)	0.980 (± 0.00666)
	Privilege	99(± 0.00030)	80(± 0.11204)	0.984 (± 0.03003)

Table 8 Results of K-Fold CV for LR model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
LR	DoS	99 (± 0.00114)	99 (± 0.00117)	0.999 (± 0.00026)
	Probe	98 (± 0.00276)	98 (± 0.00262)	0.997 (± 0.00081)
	Access	94 (± 0.00520)	96 (± 0.00875)	0.990 (± 0.00299)
	Privilege	97 (± 0.00806)	94 (± 0.11419)	0.977 (± 0.06994)

Table 9 Results of K-Fold CV for GB model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
GB	DoS	99 (± 0.00047)	99 (± 0.00047)	0.999 (± 0.00005)
	Probe	99 (± 0.00054)	99 (± 0.00166)	0.999 (± 0.00032)
	Access	99 (± 0.00107)	95.5(± 0.03091)	0.998 (± 0.00312)
	Privilege	99 (± 0.00042)	78(± 0.19788)	0.993 (± 0.01895)

Table 10 Results of K-Fold CV for AdaBoost model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
AdaBoost	DoS	99 (± 0.00072)	99 (± 0.00074)	0.999 (± 0.00004)
	Probe	99 (± 0.00136)	99 (± 0.00301)	0.999 (± 0.00035)
	Access	99 (± 0.00093)	94 (± 0.03454)	0.998 (± 0.00354)
	Privilege	99 (± 0.00824)	96 (± 0.12431)	0.969 (± 0.12478)

Table 11 Results of K-Fold CV for RFs Model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
RFs	DoS	99 (± 0.00085)	99 (± 0.00091)	0.999 (± 0.00102)
	Probe	99 (± 0.00100)	98 (± 0.00320)	0.996 (± 0.00134)
	Access	99 (± 0.00155)	92 (± 0.04108)	0.943 (± 0.03075)
	Privilege	99 (± 0.00047)	71(± 0.26499)	0.990 (± 0.01711)

Table 12 Results of K-Fold CV for ERTs model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
ERTs	DoS	99 (± 0.00042)	99 (± 0.00040)	0.999 (± 0.00017)
	Probe	99 (± 0.00111)	98 (± 0.00398)	0.999 (± 0.00052)
	Access	99 (± 0.00128)	92 (± 0.03482)	0.992 (± 0.00763)
	Privilege	99 (± 0.00039)	74 (± 0.21001)	0.995 (± 0.01095)

Table 13 Results of stratified K-Fold CV for AdaBoost model

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
AdaBoost	DoS	99 (± 0.00095)	99 (± 0.00100)	0.999 (± 0.00008)
	Probe	99 (± 0.00260)	99 (± 0.00701)	0.999 (± 0.00091)
	Access	99 (± 0.00227)	95 (± 0.06456)	0.999 (± 0.00633)
	Privilege	99 (± 0.00881)	96 (± 0.23824)	0.967 (± 0.23886)

3.6.2 Stratified K-fold CV

It is the same as a K-fold CV but uses stratified sampling to avoid two issues: random sampling in the K-fold CV method and the imbalance in the sample size in the data set. The strata have nearly the same rate of samples as in the original data set, and each fold has the same size as normal and attack samples. Consequently, whichever criteria are used to evaluate them, the findings will be consistent across all folds [13, 22, 59]. Table 13 illustrates the results of the stratified K-fold CV applied to the model which achieved better results in K-fold CV. Furthermore, the stratified K-fold CV approach delivers good results for the AdaBoost model, guaranteeing that the above-mentioned concerns are addressed.

3.7 Training models

This stage covers how to train machine learning algorithms on the training data set. Algorithm (1) demonstrates the implementation phase of the framework. This framework is written in Python and uses the scikit-learn framework as a backend ML tool to analyze the predicted data and find the best model to assess the probable normal and abnormal behavior on a LAN.

Algorithm 1 Supervised Learning Algorithm

Input: D: Training Data set = (x, y)
Output: Classify Normal or Malicious behavior in traffic
Step 1: Data Pre-Processing
Step 2: Tuning Classifier Hyperparameter (RSA)
Step 3: Feature Selection (UFS and RFE)
Step 4: Avoid Overfitting (K-Fold and Stratified K-Fold)
Step 5: Learn Classifier on the Data set.
Return Classifier

3.8 Final evaluation

The final evaluation performance is implemented by using the test set. The primary principles of testing the models are their capacity to appropriately adjust to new, previously unobserved data and the model's quality, which is determined via some evaluation measures. Performance estimators are derived from the confusion matrix (CM), which visualizes the prediction results [22, 23, 33, 39], represented by four rates, as indicated in Table 14 The number of predicted values is represented in each column of the CM, while the number of actual values is represented in each row.

Table 14 Confusion matrix

Predicted values		
Actual values	Normal	Attack
Normal	TP	FN
Attack	FP	TN

TP **=** True Positive (Normal Traffic Predicted as Normal).

TN **=** True Negative (Malicious Traffic Predicted as Malicious).

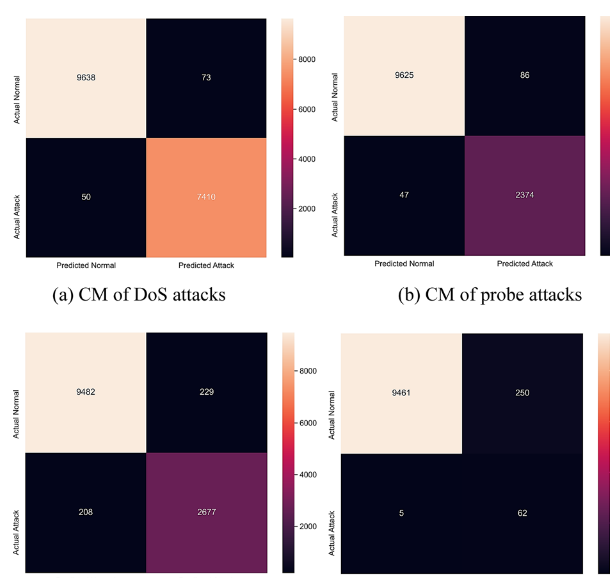
FP **=** False Positive (Malicious Traffic Predicted as Normal).

FN **=** False Negative (Normal Traffic Predicted as Malicious).

Our research findings reveal that the AdaBoost model gets a higher accuracy, as exhibited in Fig. 10 which shows the CMs of the experiment results for predicted values and actual values based on the above-mentioned rates by the AdaBoost model.

Accuracy (Acc), recall (Rec), or true-positive rate [14, 23, 33], and area under the receiver operating characteristic curve (AUC-ROC) are three essential assessment metrics generated from the rates listed above. The accuracy score denotes the proportion of true-positive and true-negative predictions generated by a model as a percentage of the total number of predictions made by Eq. (16).

$$\mathrm{Acc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$



(a) CM of DoS attacks (b) CM of probe attacks

(c) CM of access attacks (d) CM of privilege attacks

Fig. 10 CMs for the AdaBoost model

Table 15 The final evaluation of the AdaBoost model of predicted attacks

Classifier	Attack	Measure		
		Acc (%)	Rec (%)	AUC
AdaBoost	DOS	99.2	99.3	0.992
	Probe	99	98	0.986
	Access	96.5	92.7	0.952
	Privilege	97.3	92.5	0.954

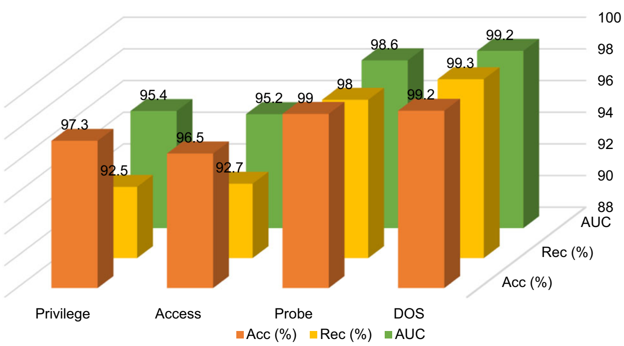
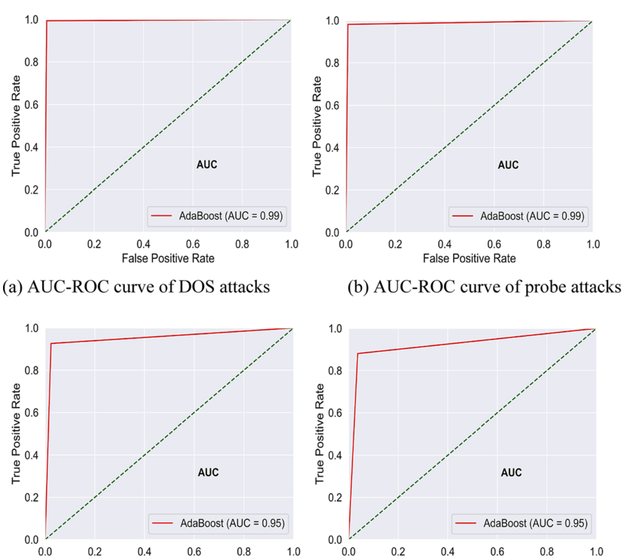


Fig. 11 Representing AdaBoost model results



(a) AUC-ROC curve of DOS attacks (b) AUC-ROC curve of probe attacks

(c) AUC-ROC curve of access attacks (d) AUC-ROC curve of privilege attacks

Fig. 12 AUC-ROC curves of AdaBoost model * MERGEFORMAT * MERGEFORMAT

The recall [14, 23, 33, 39] of the ML model indicates its ability to define the proportion of true positives that are correctly classified by Eq. (17), while the AUC-ROC indicates positive predictions that are classified higher than negative predictions. The ROC-AUC curve is represented as a plot of the false-positive rate (FPR) as the x-axis versus the TPR as the y-axis. (17) and (18) EQs are used to calculate AUC-ROC [13, 14, 22, 39]. Table 15 and Fig. 11 show the findings for the most accurate model (AdaBoost).

$$\text{Rec} = \frac{TP}{TP + FN} \tag{17}$$

$$FPR = \frac{FP}{TN + FP} \tag{18}$$

Figure 12 describes the AUC-ROC evaluation for detecting attacks by the AdaBoost model. It accurately identifies attack samples with an AUC of DoS attaining 0.992 (7410 out of 7460 samples) of Probe reaching 0.986 (2374 out of 2421 samples), of Access totaling 0.952 (2677 out of 2885 samples), and of privilege achieving 0.954 (62 out of 67 samples).

3.8.1 Comparison with related works

We compared our proposed method with the existing related works that used the NSL-KDD data set in the field. Three metrics were used to compare the performance of our method with others: recall, AUC (TPR vs. FPR), and accuracy. Our current study produces superior results through employing the AdaBoost model in all attack branches. For example, our DoS attacks have a recall of 99.3% and an AUC of 0.992, in addition to overall accuracy for all attack categories reaching 98.5%. In contrast, the highest results of [60] were a recall of 96.5%, an AUC of 0.980, and an overall accuracy of 94%. Table 16 displays all the results.

4 Conclusion and future work

This study aims to determine the most accurate ML classifier for detecting LAN attacks. The research findings demonstrate that the AdaBoost model has the highest classification accuracy for both insider attacks and normal traffic behavior, with 99% DoS, 98% probe, 96% access, and 97% privilege. It also has an AUC of 0.992 DoS, 0.986 probe, 0.952 access, and 0.954 privilege. The study is carried out using the publicly accessible NSL-KDD data set, with an AUC rate measure overriding previous approaches in this data set due to the strategies used to remove noise from the data set, the choice of relevant features, the tuning of hyperparameters, and the minimization of bias. As a future recommendation, the techniques used in the study might be integrated into firewall

Table 16 Performance comparison with existing related works

Study	Model	No. of feature	DoS		Probe		Access		Privilege		Overall Acc (%)
			Rec (%)	AUC	Rec (%)	AUC	Rec (%)	AUC	Rec (%)	AUC	
Our approach	AdaBoost	13	99.3	0.992	98	0.986	92.7	0.952	92.5	0.954	98.5
[60]	BP	–	80.2	0.863	96.6	0.921	31.2	0.649	87.2	0.781	73.1
	RBF	–	88.7	0.933	60.3	0.786	55.5	0.766	84.6	0.722	77.1
	SVM	–	86.9	0.837	87.2	0.866	0	0	90.3	0.917	80.2
	LIBSVM	–	88.7	0.897	90.8	0.876	55.2	0.765	88.1	0.853	81.6
	KELM	–	56.6	0.751	88.2	0.896	28.3	0.64	95.6	0.827	76.8
	CNN	–	91.5	0.941	89.4	0.939	63.4	0.803	95.7	0.907	88.4
	DBN	–	90.2	0.917	89.9	0.933	49.3	0.74	94.2	0.903	87.1
	DBN-KELM	–	85.6	0.908	83.6	0.947	81.5	0.903	95.6	0.941	89.2
	DBN-EGWO-KELM	–	96.5	0.980	98.1	0.984	98.1	0.950	87.5	0.931	94.0

configurations to identify insider threats and assist cybersecurity specialists in making the work environment secure and minimizing risks.

Author contributions MF designed the study, performed the computations, interpreted data, and wrote the manuscript. RHS and NH encouraged Mohamed Farouk[1] to investigate the problem of the study, supervised the study, and contributed to the final manuscript.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Data availability The data set supporting the conclusions of this article is available in the University of New Brunswick repository. Here is the hyperlink to a data set: http://205.174.165.80/CICDataset/NSL-KDD/Dataset/

Declarations

Conflict of interest Not applicable.

Ethical approval There is not any ethical conflict.

References

1. Cybersecurity and infrastructure security agency (2022) Insider threat mitigation. CISA. https://www.cisa.gov/insider-threat-mitigation Accessed 20 Aug. 2022
2. Yuan S, Wu X (2021) Deep learning for insider threat detection: review, challenges, and opportunities. Comput Secur. https://doi.org/10.1016/j.cose.2021.102221
3. Kim A, Oh J, Ryu J, Lee K (2020) A review of insider threat detection approaches with IoT perspective. Special section on secure communication for the next generation 5g and IOT networks. https://doi.org/10.1109/ACCESS.2020.2990195
4. Pallabi Parveen JE (2011) Insider threat detection using stream mining and graph mining. IEEE third international conference on privacy, security, risk and trust and 2011 IEEE third international conference on social computing. https://doi.org/10.1109/PASSAT/SocialCom.2011.211
5. Nebrase Elmrabit SHY (2020) Insider threat risk prediction based on bayesian network. Comput Secur. https://doi.org/10.1016/j.cose.2020.101908
6. Egress (2021) 94 % of organizations suffer data breaches. Egress. https://www.egress.com/newsroom/94-percent-of-organisations-have-suffered-insider-data-breaches. Accessed 9 April 2022
7. Proofpoint (2022) 2022 Ponemon cost of insider threats global report. Proofpoint. https://protectera.com.au/wp-content/uploads/2022/03/The-Cost-of-Insider-Threats-2022-Global-Report.pdf. Accessed 30 April 2022
8. Dastres R, Soori M (2021) A review in recent development of network threats and security measures. Int J Inf Sci Comput Eng 15(1). https://hal.science/hal-03128076
9. Korotka MS, Yin LR, Basu SC (2014) Information assurance technical framework: an end user perspective. J Inf Priv Secur. https://doi.org/10.1080/15536548.2005.10855759
10. Lei J (2019) Cross-validation with confidence. J Am Stat Assoc. https://doi.org/10.1080/01621459.2019.1672556
11. Probst P, Boulesteix AL, Bischl B (2019) Tunability: importance of hyperparameters of machine learning algorithms. J Mach Learn Res 20(1):1934–1965

12. Ahmad Esmaeili ZG (2023) Agent-based collaborative random search for hyperparameter tuning and global function optimization. Systems. https://doi.org/10.3390/systems11050228
13. Montesinos López OA, Montesinos López A, Crossa J (2022) General elements of genomic selection and statistical learning, preprocessing tools for data preparation, & overfitting, model tuning, and evaluation of prediction performance. In: multivariate statistical machine learning methods for genomic prediction. Springer, Cham, pp 25–139. https://doi.org/10.1007/978-3-030-89010-0
14. Zhou ZH (2021) Model selection and evaluation. In: machine learning, 1st edn. Springer, Singapore, pp 25–55. https://doi.org/10.1007/978-981-15-1967-3
15. Yates LA (2021) Parsimonious model selection using information theory: a modified selection rule. Ecol Soc Am. https://doi.org/10.1002/ecy.3475
16. Yates LA (2022) Cross validation for model selection: a review with examples from ecology. Ecol Monogr. https://doi.org/10.1002/ecm.1557
17. Al-Mhiqani MN, Ahmad R, Zainal Abidin Z, Yassin W, Hassan A, Abdulkareem KH, Ali NS, Yunos Z (2020) A review of insider threat detection: classification, machine learning techniques, datasets, open challenges, and recommendations. Appl Sci. https://doi.org/10.3390/app10155208
18. Aram Kim JO (2019) SoK: a systematic review of insider threat detection. J Wirel Mob Netw Ubiquitous Comput Dependable Appl. https://doi.org/10.22667/JOWUA.2019.12.31.046
19. Sarker IH (2021) Machine learning: algorithms, real world applications and research directions. SN Comput Sci 2(3):160. https://doi.org/10.1007/s42979-021-00592-x
20. Altwaijry BB (2023) Insider threat detection using machine learning approach. Appl Sci. https://doi.org/10.3390/app13010259
21. Abualkibash M (2019) Intrusion detection system classification using different machine learning algorithms on kdd-99 and nsl-kdd datasets-a review paper. Int J Comput Sci Inf Technol. https://doi.org/10.5121/ijcsit.2019.11306
22. Müller Andreas C, Guido S (2017) Introduction to machine learning with python: a guide for data scientists. O'Reilly Media, Sebastopol, CA
23. Xu W, Jang-Jaccard J, Singh A, Wei Y, Sabrina F (2021) Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset. IEEE Access. https://doi.org/10.1109/ACCESS.2021.3116612
24. Alsowail RA, Al-Shehari T (2022) Techniques and countermeasures for preventing insider threats. Peer J Comput Sci. https://doi.org/10.7717/peerj-cs.938
25. Yuan S, Wu X (2021) Deep learning for insider threat detection: review challenges and opportunities. Comput Secur 104:102221. https://doi.org/10.1016/j.cose.2021.102221
26. Scikit-Learn (2019) sklearn preprocessing OneHotEncoder. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html. Accessed 5 May 2022
27. Homoliak I, Toffalini F, Guarnizo J, Elovici Y, Ochoa M (2019) Insight into insiders and IT: a survey of insider threat taxonomies, analysis, modeling, and countermeasures. ACM Comput Surv. https://doi.org/10.1145/3303771
28. Schratz P, Muenchow J, Iturritxa E, Richter J, Brenning A (2019) Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data. Ecol Model. https://doi.org/10.1016/j.ecolmodel.2019.06.002
29. Berrar D (2019) Cross-validation. Encycl Bioinform Comput Biol. https://doi.org/10.1016/b978-0-12-809633-8.20349-x
30. Ngueajio MK, Washington G, Rawat DB, Ngueabou Y (2023) Intrusion detection systems using support vector machines on the KDDCUP'99 and NSL-KDD datasets: a comprehensive survey. Intell Syst Appl. https://doi.org/10.1007/978-3-031-16078-3_42
31. Oladimeji TO, Ayo CK, Adewumi SE (2019) Review on insider threat detection techniques. J Phys Conf Ser. https://doi.org/10.1088/1742-6596/1299/1/012046
32. Han J, Kamber M, Pei J (2011) Getting to know your data and data preprocessing. In: data mining: concepts and techniques, 3rd edn. San Francisco, pp 39–124. https://doi.org/10.1016/C2009-0-61819-5
33. Yin C, Zhu Y, Fei J, He X (2017) A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access. https://doi.org/10.1109/ACCESS.2017.2762418
34. Özgür A, Erdem H (2016) A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. Peer J Preprints. https://doi.org/10.7287/peerj.preprints.1954v1
35. Liu L, Chen C, Zhang J, De Vel O, Xiang Y (2019) Insider threat identification using the simultaneous neural learning of multi-source logs. IEEE Access. https://doi.org/10.1109/access.2019.2957055
36. Zeng C, Lu H, Chen K, Wang R, Tao J (2023) Synthetic minority with cutmix for imbalanced image classification. Intell Syst Appl. https://doi.org/10.1007/978-3-031-16078-3_37
37. Wang Q, Yang G, Wang L, Fu J, Liu X (2023) SR-IDS: a Novel network intrusion detection system based on self-taught learning and representation learning. Artificial neural networks and machine learning–ICANN 2023. https://doi.org/10.1007/978-3-031-44213-1_46
38. Zhang A, Lipton ZC, Li M, Smola AJ (2022) Linear neural networks. In: dive into deep learning, 1st edn. pp 87–128
39. Moon SA (2020) Feature selection methods simultaneously improve the detection accuracy and model building time of machine learning classifiers. Symmetry. https://doi.org/10.3390/sym12091424
40. Scikit-Learn (2023) Sklearn preprocessing robustscaler. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html?highlight=robust#sklearn.preprocessing.RobustScaler.fit. Accessed 15 May 2022
41. Scikit-Learn (2022) Preprocessing data. Scikit-Learn. https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing. Accessed 17 May 2022
42. Luo J, Qiao H, Zhang B (2021) Learning with smooth Hinge losses. Neurocomputing. https://doi.org/10.1016/j.neucom.2021.08.060
43. Géron Aurélien (2017) Support vector machines. In: hands-on machine learning with scikit-learn and tensorflow: concepts, tools, and techniques to build intelligent systems. 1st edn. O'Reilly Media, Sebastopol, CA, pp 145–166.
44. Manzali Y, Chahhou M, El Mohajir M (2017) Impure decision trees for auc and log loss optimization. IEEE Xplore. https://doi.org/10.1109/WITS.2017.7934675
45. Scikit-Learn (2014) model logistic regression. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed 25 October 2023
46. Wicht D, Schneider M, Böhlke T (2019) On quasi-newton methods in fast fourier transform-based micromechanics. Int J Numer Methods Eng. https://doi.org/10.1002/nme.6283
47. Wang C, Sun D, Toh KC (2010) Solving log-determinant optimization problems by a newton-cg primal proximal point algorithm. SIAM J Optim. https://doi.org/10.1137/090772514
48. Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: a library for large linear classification. J Mach Learn Res 9:1871–2174
49. Defazio A, Bach F, Lacoste-Julien S (2014) SAGA: a fast-incremental gradient method with support for non-strongly convex composite objectives. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.1407.0202

50. Chen A, Chen B, Chai X, Rui B, Li H (2017) A novel stochastic stratified average gradient method: convergence rate and its complexity. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.1710.07783
51. scikit-learn (2009) Gradient boosting classifier. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html. Accessed 10 October 2023
52. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. The Ann Stat. https://doi.org/10.1214/aos/1013203451
53. Scikit-learn (2023) ensemble AdaBoost Classifier. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html. Accessed 12 October 2023
54. Hastie T, Rosset S, Zhu J, Zou H (2009) Multi-class AdaBoost. Stat Its Interface. https://doi.org/10.4310/sii.2009.v2.n3.a8
55. Ferrario A, Hämmerli R (2019) On boosting: theory and applications. Soc Sci Res Netw. https://doi.org/10.3929/ethz-b-000383242
56. oneDAL (2023) AdaBoost multiclass classifier. OneDAL. https://oneapi-src.github.io/oneDAL/daal/algorithms/boosting/adaboost-multiclass.html. Accessed 20 October 2023
57. Scikit-Learn (2019) Feature selection. Scikit-Learn. https://scikit-learn.org/stable/modules/feature_selection.html. Accessed 18 May 2022
58. Chen T, Xu M, Tu J, Wang H, Niu X (2018) Relationship between omnibus and post-hoc tests: an investigation of performance of the F test in ANOVA. Shanghai archives of psychiatry. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5925602/
59. SciKit-Learn (2009) Cross-validation: evaluating estimator performance. Scikit-Learn. https://scikit-learn.org/stable/modules/cross_validation.html. Accessed 22 May 2022
60. Wang Z, Zeng Y, Liu Y, Li D (2021) Deep belief network integrating improved kernel-based extreme learning machine for network intrusion detection. IEEE Access. https://doi.org/10.1109/ACCESS.2021.3051074