Hindawi
Security and Communication Networks
Volume 2021, Article ID 8836057, 28 pages
https://doi.org/10.1155/2021/8836057

Research Article

Empirical Evaluation of Noise Influence on Supervised Machine Learning Algorithms Using Intrusion Detection Datasets

Khalid M. Al-Gethami,[1] **Mousa T. Al-Akhras**,[1,2] **and Mohammed Alawairdhi**[1]

[1]*Computer Science Department, College of Computing and Informatics, Saudi Electronic University, Riyadh, Saudi Arabia*
[2]*Computer Information Systems Department, King Abdullah II School for Information Technology, The University of Jordan, Amman, Jordan*

Correspondence should be addressed to Khalid M. Al-Gethami; khalid.m.algethami@gmail.com

Received 22 April 2020; Revised 4 December 2020; Accepted 30 December 2020; Published 15 January 2021

Academic Editor: Tom Chen

Optimizing the detection of intrusions is becoming more crucial due to the continuously rising rates and ferocity of cyber threats and attacks. One of the popular methods to optimize the accuracy of intrusion detection systems (IDSs) is by employing machine learning (ML) techniques. However, there are many factors that affect the accuracy of the ML-based IDSs. One of these factors is noise, which can be in the form of mislabelled instances, outliers, or extreme values. Determining the extent effect of noise helps to design and build more robust ML-based IDSs. This paper empirically examines the extent effect of noise on the accuracy of the ML-based IDSs by conducting a wide set of different experiments. The used ML algorithms are decision tree (DT), random forest (RF), support vector machine (SVM), artificial neural networks (ANNs), and Naïve Bayes (NB). In addition, the experiments are conducted on two widely used intrusion datasets, which are NSL-KDD and UNSW-NB15. Moreover, the paper also investigates the use of these ML algorithms as base classifiers with two ensembles of classifiers learning methods, which are bagging and boosting. The detailed results and findings are illustrated and discussed in this paper.

1. Introduction

The numbers of cyber threats facing individuals, organizations, and government agencies are increasing rapidly. In addition, the ferocity of such attacks has also increased to more destructive levels. High-level intruders such as the advanced persistent threat (APT) are utilizing stealthier methods to penetrate protected systems and networks. This emphasizes the urgent need to optimize intrusion detection systems (IDSs).

IDSs are a form of technical security controls that can be used to detect different forms of intrusions, malicious patterns, probing attempts, and unauthorized activities.

The accuracy of IDSs can be optimized using machine learning (ML) techniques which were used successfully in many fields such as image identification and pattern recognition. In addition, ML-based techniques are also widely deployed in the intrusion detection field. One of the biggest advantages of ML-based IDSs is that they can be utilized to detect both misuse and anomaly attacks. However, ML-based techniques are more suitable to detect anomalies which are extremely important to confront zero-day attacks.

New intrusion detection techniques, whether they are based on ML techniques or not, are usually tested on intrusion-related datasets that contain normal traffic and activities injected with different forms of attacks. These datasets are widely used to evaluate the effectiveness of newly proposed IDSs. However, the presence of noise might influence the effectiveness and impacts the accuracy of ML-based IDSs, especially when implementing such systems in the real world, where noisy data are more likely to occur.

In this paper, we will empirically investigate the effect of mislabelled instances that can influence the effectiveness of ML-based IDSs. Different levels of noise are injected, and the classification accuracy of the ML-based IDSs is studied. Furthermore, the effect of using noise filtering on the accuracy of ML-based IDSs is also analyzed. Noise will be filtered by excluding outliers and extreme value instances.

The aim of analyzing noise with ML algorithm is to test the robustness of different ML algorithms in noisy situations. In other words, it will help to determine the most noise-resilient ML algorithm. In addition, the paper also investigates the employment of ensemble learning techniques with the IDSs.

The rest of this paper is organized as follows: Section 2 reviews IDS, relevant ML algorithms, ensemble of classifiers, noise filtering, and several works from the literature related to the use of ML in IDS. Section 3 presents the proposed methodology. Several datasets related to intrusion detection are reviewed. Also, the conducted experiments are described including noise injection, noise filtering, and the metrics used to compare the performance of different ML models. Section 4 discusses the results and analyses the outcomes of the conducted experiments. Finally, Section 5 concludes the paper and presents avenues for future work.

2. Literature Review

This section reviews intrusion detection, ML, and noise. It starts by demonstrating the types, components, and deployment of IDSs. ML categories, ML algorithms that are used in this paper, and the ensemble learning methods are also discussed. It also discusses noise filtering by removing the outliers and the extreme values. Finally, this section demonstrates previous research efforts that focus on ML-based IDSs.

2.1. Intrusion Detection System. IDS helps to detect many forms of attacks and sends alarms to the system or the security administrators. It is critical to develop an IDS that achieves high detection rates with no or minimum false alarms. IDS can be broadly categorized into misuse and anomaly detection.

Elsayed et al. [1] gave credit to Anderson for the introduction of the concept of intrusion detection in his 1980 paper. That paper discussed the feasibility of detecting misuse by investigating the audit trails. Elsayed et al. [1] pointed out that the original IDS was based on expert systems.

According to Rathore et al. [2], the concept of the IDS was initially introduced in 1986 [3] and 1987 [4] by Denning based on expert systems. The system was referred to as the intrusion detection expert system (IDES). However, Kumar and Venugopalan [5] gave a balanced credit; according to them, Anderson introduced the concept of the IDS while Denning formalized IDS by introducing a rule-based expert system that can detect malicious users' activities in real time. An overview of the historical IDS milestones can be found in [5].

2.1.1. Types of IDSs. The way the collected data are inspected defines how efficiently an IDS can detect different categories of probes and intrusions. Generally, the methods that an IDS analysis engine uses to inspect the collected data can be broadly categorized into either misuse detection or anomaly detection.

Misuse detection approaches operate by matching the current behavior with known and previously defined attack patterns, signatures, or rules [6]. Therefore, it is critical to maintain an IDS up to date and inclusive of different known attack patterns, signatures, and rules to ensure the effectiveness of the misuse detection IDS. However, it is not feasible to use the misuse-based methods to detect zero-day attacks because no patterns, signatures, or rules are defined for zero-day attacks.

On the other hand, anomaly detection techniques operate based on the ability to detect deviations from normal behavior or pattern [7]. It operates based on the principle that malicious behavior differs from normal behavior, which makes it feasible to detect it [6]. Defining normal behavior requires a period of time; consequently, anomaly-based IDS needs sometime after the installation to become active. Anomaly-based IDSs are more prone to errors producing high rates of false alarms.

ML-based IDS can be used to detect both misuse and anomaly [5, 6]. In particular, the supervised ML can be used with misuse detection, whereas unsupervised ML-based techniques can be used to conduct anomaly detection. Table 1 highlights the main differences between the misuse and the anomaly intrusion detection methods [6].

2.1.2. Components of IDSs. A typical network-based IDS consists of four components, which are a decoder, a preprocessor, a decision engine sensor, and a defense response. The decoder uses data collection tools to gather specific network traffic in raw format and passes these data to the second component. The preprocessor examines the set of protocols that are used in the transmission of the data and extracts certain features. After that, the decision engine sensor utilizes these features to differentiate normal traffic from malicious traffic. If malicious traffic were detected, it sends a signal to the defense response, which, in turn, sends an alert to the security administrators and logs the event in the database [8]. Figure 1 illustrates the typical components of an IDS.

2.1.3. Deployment of IDSs. IDSs can be deployed either in the network or within the host. An IDS that is deployed across the network is typically called network-based IDS (NIDS) and an IDS that runs at a certain host is called host-based IDS (HIDS) [9]. NIDS can be used to monitor the traffic within the organization's network, typically installed at the network choke points. On the other hand, HIDS can be installed on critical servers to detect potential intrusions or misuses. It can also be installed on the decoy systems (Honey pots) to detect potential intrusions.

2.2. Machine Learning. Machine learning (ML) is a branch of artificial intelligence (AI). The term "machine learning" was coined by Arthur Samuel back in 1959 when he was working at IBM [5]. Samuel defined ML as "the field of study that gives computers the ability to learn without being explicitly programmed" [5]. It is closely related to

Table 1: The main differences between misuse and anomaly intrusion detection methods.

Criteria	Misuse detection	Anomaly detection
Basic principle	Using previously defined patterns/signatures/ rules to detect intrusion	Establishes normal behavior profile and detects intrusion based on deviations from that profile
Detecting unknown attacks	No	Yes
Detecting known attacks	Very good	Good
Rates of false positive	Very low	Moderate to high (depends on many factors)
Constant update of the database	Yes	No
Suitable ML approach	Supervised ML	Unsupervised ML
Biggest challenge	Maintaining an up-to-date database of all known attack signatures	Differentiating normal from malicious behavior

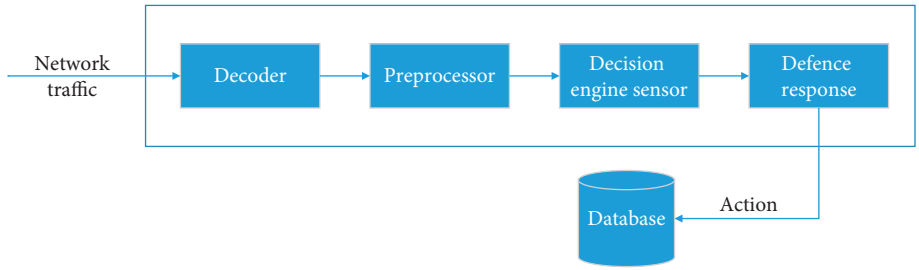

Figure 1: The typical components of an IDS.

computational statistics [10]. The first ML algorithms emerged in the 1970s [11]; nowadays, ML is becoming a popular research area [12].

ML algorithms can be classified into four distinct categories, supervised learning, semisupervised learning, unsupervised learning, and reinforcement learning [13]. Both supervised and unsupervised learning can be divided further into two different subcategories. The general classification of ML is presented in Figure 2.

2.2.1. Supervised Learning. In supervised learning, an algorithm is presented with a set of data that contains the input data and the corresponding output. The algorithm attempts to discover the underlying relation between the input and the output [11]. In supervised learning, data are labelled. Supervised learning can be typically used in two categories of problems [14]:

(i) Classification problems that aim to predict a discrete number of values

(ii) Regression problems that aim to predict a continuous-valued output

This paper focuses on supervised learning. More specifically, to study the effect of noise on the classification models, each ML model will be trained on labelled instances of two different IDS datasets (described later). In the first dataset, each instance is labelled as either normal or anomaly. In the second dataset, instances are labelled as either 0 (normal) or 1 (malicious).

2.2.2. Semisupervised Learning. Semisupervised learning algorithm is a combination of supervised and unsupervised algorithms that leverages a small percentage of labelled instances to generalize on a large percentage of unlabelled instances. This is very useful in certain cases such as searching for a particular person in an extremely large set of images. For instance, a certain person could be manually labelled in few images and the model will start mining for that person in the remaining large part of the images [13].

2.2.3. Unsupervised Learning. Unsupervised learning attempts to deduce the hidden structure of the datasets. Typically, all data in unsupervised learning algorithms are unlabelled. Unsupervised learning problems can be categorized as follows [11]:

(i) Clustering problems that attempt to divide the data into clusters that satisfy certain criteria

(ii) Dimensionality reduction problems that attempt to reduce the dimensionality of the data while maintaining the fundamental aspects of the data, i.e., highest variability

2.2.4. Reinforcement Learning. This form of learning is based on trial-and-error approach, in which a learning system gathers certain data and takes an action; if the action yielded a positive result, then a reward will be recorded. If that action resulted in an unwanted result, then the system will learn that this action in that context will not likely work well in future events. This form of learning is suitable for certain fields such as robotics. It can be utilized to learn robotics how to move, carry an object, or avoid physical obstacles [13].

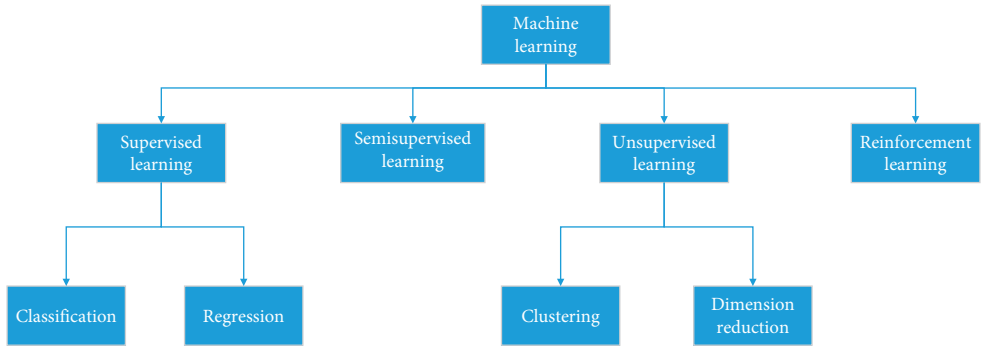

Figure 2: A general ML classification.

2.3. ML Algorithms Used in This Paper. There are a large number of ML algorithms. The ML algorithms that are used in this paper are as follows.

2.3.1. Decision Trees (DTs). DT uses a set of rules to classify data based on the attributes' values [10]. Classification is represented in a tree-structure format where branches represent the selection of the value of input features that lead to those classifications and leaves represent class labels. DTs are characterized by the high accuracy of the classification and the simplicity of implementation. However, decision trees are biased with multilevel features [15].

In this paper, J48 flavor of decision trees is used. During the tree construction, the training dataset is recursively divided into several subsets. The best split is often based on children impurity such as the entropy which is defined as follows [16, 17]:

$$\text{entropy} = -\sum_{0}^{c-1} p(i|t)\log_2 p(i|t), \tag{1}$$

where c is the number of classes and $0\log_2 0 = 0$ in entropy calculations.

2.3.2. Random Forest (RF). RF combines multiple DTs where each DT is constructed based on the values of independent random vectors. The results of the RF can be controlled by the majority or weighted voting [15]. RF can be considered as one of the ensemble learning methods [2] because RF tends to combine the outcomes of several DTs [6]. When the number of trees is sufficiently large, the upper bound of the generalization error converges according to the following formula [16]:

$$\text{generalization error} \le \frac{\overline{\rho}\left(1 - s^2\right)}{s^2}, \tag{2}$$

where $\overline{\rho}$ is the average correlation among the trees and s is a measure of strength of the tree classifiers. The strength refers to the average performance of the classifiers measured probabilistically as

$$\text{margin}, M(X, Y) = P\left(\widehat{Y}_\theta = Y\right) - \max_{Z \neq Y} P\left(\widehat{Y}_\theta = Z\right), \tag{3}$$

where $\widehat{Y}_\theta$ is the predicted class of X according to a classifier built from some random vector θ. The higher the margin, the more likely the classifier correctly predicts an example X.

2.3.3. Support Vector Machine (SVM). SVM is a supervised learning algorithm that uses a hyperplane to classify the data. The hyperplane is used to separate the data into different classes based on the features' space. SVM algorithm is among the most robust and accurate ML algorithms [10]. A linear SVM classifier differentiates between instances from two classes according to the following equation [13]:

$$\widehat{y} = \begin{cases} 0, & \text{if } w^T x + b < 0, \\ 1, & \text{if } w^T x + b \ge 0, \end{cases} \tag{4}$$

where x represents an instance to be classified, w represents the weight vector, and b is the bias value.

2.3.4. Artificial Neural Networks (ANNs). ANN was inspired by the human brain. The implementation of the ANN in the computer realm consists of a number of artificial neurons that are distributed across different layers. An ANN typically consists of three layers [6, 18, 19], which are the input layer, the hidden layer, and the output layer. The dataset is passed to the input nodes and the output nodes are used to present the classification results. This paper uses multilayer perceptron (MLP) to implement ANN within the experiments. MLP is an ANN, usually trained using backpropagation algorithm [17, 20].

The output of the hidden neuron i can be calculated as follows:

$$h_i = f^1\left(b_i + \sum_{j=1}^{n} W_{ij} X_i\right), \tag{5}$$

where x_i is the input data, x_i, $i = 1, \ldots, n$; W_{ij} denotes the weight connecting input neuron j to hidden neuron i, b_i is the bias for neuron i, and f^1 is the used transfer function.

The output of the output neuron i can be calculated as follows [17]:

$$O_i = f^2\left(b_i + \sum_{j=1}^{n_k} V_{ij} h_i\right), \tag{6}$$

where n_k is the number of hidden neurons and V_{ij} denotes the weight connecting hidden neuron i to the output neuron j.

The goal of ANN is to minimize the total sum of squared error between the predicted output and the true output (ground truth) [16]:

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(yi - \hat{y}_i)^2. \quad (7)$$

The weights of the network can be updated according to the gradient descent method as follows [16]:

$$w_j \leftarrow w_j - \lambda \frac{\partial E(w)}{\partial w_j}. \quad (8)$$

2.3.5. Naïve Bayes (NB). NB is called naïve because it relies on a simplifying assumption, which assumes that the attribute values are conditionally independent of each other. One of the main advantages of Naïve Bayes is the speed of training and testing. However, the attribute values of some intrusion detection datasets such as the KDD CUP 99 and the NSL-KDD are highly dependent on each other [7]. This means that the accuracy of NB is highly influenced by the degree of dependencies between the attribute values of the datasets [15]. When building an IDS classifier based on the NB algorithm, the classification is performed according to the following equation:

$$y(f_1, f_2, \ldots, f_m) = \underset{k\in\{1,\ldots,K\}}{\operatorname{argmax}}\, p(C_k)\prod_{i=1}^{m} p(f_i|C_k), \quad (9)$$

where m represents the number of features, k represents the number of classes, f_i stands for the i-th feature, C_k stands for the k-th class, $p(C_k)$ is the prior probability of C_k, and $p(f_i|C_k)$ represents the conditional probability.

2.3.6. Comparison of the ML Algorithms. A comparison between the above ML algorithms is illustrated in Table 2 [15, 21]. More details about the differences between common ML algorithms can be found in [21].

2.4. Ensemble of Classifiers. Ensemble learning consolidates several techniques to obtain better overall accuracy, which outperforms the accuracy of each single technique [8]. One of the main advantages of using ensemble learning is that it might solve the overfitting issues and improves the classification accuracy. On the other hand, the disadvantage of ensemble learning includes the increase in the required time and memory to build the model when compared to individual classifiers.

Additionally, learning concepts might become more difficult to understand, i.e., less plausible. Popular methods for conducting ensemble learning are bagging, boosting, voting, and stacking [6]. This paper will utilize both bagging and boosting.

Bagging, also called bootstrap aggregation, is an ensemble method that is used to improve accuracy and reduce overfitting. This is achieved by deploying a model-averaging technique. On the other hand, boosting operates by training multiple weak learners and aggregating the weighted results [15]. In this paper, we have used the AdaBoost method to implement boosting on the intrusion datasets.

2.5. Noise Filtering. Noisy data may affect the accuracy of different ML algorithms. This paper adopts an empirical approach to study the potential effect of noise. It discusses the removal of outliers and extreme values to filter noisy data. There is a wide set of different algorithms that can be used to conduct noise filtering. This paper uses interquartile range (IQR) filter for noise filtering. This filter detects outliers and extreme values from a given dataset. According to this filter, the outliers are defined according to the following equations [22]:

$$\mathrm{Q3} + \mathrm{OF} * \mathrm{IQR} < x \leq \mathrm{Q3} + \mathrm{EVF} * \mathrm{IQR}, \quad (10)$$

$$\mathrm{Q1} - \mathrm{EVF} * \mathrm{IQR} \leq x < \mathrm{Q1} - \mathrm{OF} * \mathrm{IQR}. \quad (11)$$

On the other hand, the extreme values are located according to the following equations:

$$x > \mathrm{Q3} + \mathrm{EVF} * \mathrm{IQR}, \quad (12)$$

$$x < \mathrm{Q1} - \mathrm{EVF} * \mathrm{IQR}, \quad (13)$$

where Q1 = 25% quartile, Q3 = 75% quartile, IQR is the interquartile range, which represents the difference between Q1 and Q3, OF is the outlier factor, and EVF is the extreme value factor.

After applying this filter, outliers and extreme values will be labelled into two additional attributes. After that, all instances within these two attributes can be removed from the dataset. This means that outlier and extreme values will be excluded from the dataset, resulting in an instant reduction. Table 3 illustrates the original number of instances and the new number of instances in the used datasets before and after applying noise filtering for both datasets.

After verifying that the removal of the noisy instances did not eliminate all the attack instances, it is also critical to ensure that benign and malicious traffic were not entirely eliminated. To accomplish that, the distribution of the different classes of network traffic before and after noise filtering for both the training and the testing portions of the UNSW-NB15 dataset was calculated and the results are illustrated in Table 4, which illustrate that none of the network traffic classes were entirely eliminated after the noise filtering process. This was feasible to calculate because one of the UNSW-NB15 attributes, called the *attack_cat* attribute, was originally dedicated to illustrate the number of instances within each category. Unfortunately, it was unfeasible to calculate the distribution of the different categories of the network traffic for the NSL-KDD dataset because there was no attribute that distinguishes these categories from each other.

Table 2: A comprehensive comparison between different ML classification algorithms [15, 21].

ML algorithm	Advantages	Disadvantages
DT	(i) Very simple and fast (ii) Not affected by the increase of the dimensionality of the data (iii) The model is easily understood (plausible) (iv) Generates good accuracy based on the quality of the data (v) Support incremental learning	(i) Requires long time to train the model (ii) Requires larger amount of memory for analyzing large databases (iii) Not suitable for problems that require diagonal partitioning (iv) Can generate a complex representation for some concepts due to the replication (v) The order of the attributes affects the performance
RF	(i) Can resist overfitting (ii) Does not require attributes selection (iii) Model variances decrease with the increase in the number of trees (iv) Increasing the number of trees does not affect the bias of the model	(i) Difficult to interpret the model (ii) Correlated variables cause performance degradation (iii) Reliance on the random generator of the implementation
SVM	(i) One of the most robust and accurate algorithms (ii) Requires few data for training and is not affected by the dimensionality of the data (iii) Generate the best function to conduct binary classification (iv) Less prone to overfitting compared to other algorithms	(i) Requires intensive computations to build the model (ii) Very slow in the learning phase (iii) Memory required doubles as the number of training instances increases (iv) It is difficult to interpret the results of the model
ANN	(i) Tolerable to noise and able to predict new classes (ii) Applicable even if the relation between attributes and classes is not well defined (iii) Suitable for continuous values (iv) Computations can be accelerated due to ANN parallel nature (v) Applied successfully to different real-world issues such as handwritten character recognition and laboratory medicine	(i) Requires a long time to train a model (ii) Very difficult to interpret how the model works (iii) Requires empirical adjustment of different parameters such as the number of hidden layers and the number of nodes in each layer
NB	(i) Requires short time for training and it is easy to construct the model (ii) Can be computationally optimized (iii) Can be used with large datasets (iv) Outcomes are easily interpreted (v) It operates in a well and robust manner even though it might not be the best algorithm for a certain application	(i) Theoretically, classifiers based on NB algorithms have a low error rate. However, in practice, this is not entirely true due to the assumption that different attributes are independent of each other (ii) Yields low accuracy results compared to other ML algorithms

Table 3: The total number of instances before and after excluding the outliers and the extreme values.

Dataset	Data file	Before filtering			After filtering		
		Total no. of instances	No. of benign instances	No. of attack instances	Total no. of instances	No. of benign instances	No. of attack instances
NSL-KDD	KDDTrain+	125,973	67,343	58,630	45,331	12,360	32,971
	KDDTest+	22,544	9,711	12,833	2,670	2,149	521
UNSW-NB15	UNSW-NB15 training	175,341	56,000	119,341	54,729	7,503	47,226
	UNSW-NB15 testing	82,332	37,000	45,332	31,468	10,045	21,423

2.6. Relevant Literature. Based on the authors' research in the literature, there is a great lack of papers that empirically discuss the subject of using ML techniques on IDS-related datasets in the presence of noise.

An empirical study between several classification algorithms on two intrusion-related datasets, namely, KDD CUP 99 and NSL-KDD, was conducted by Hussain and Lalmuanawma [23]. The noise was injected to a specific set of attributes by 10% and 20%. However, NSL-KDD is basically an enhanced version of the KDD CUP 99. The redundant instances that existed in the KDD CUP 99 dataset were removed in the NSL-KDD dataset. The ML algorithms, in this paper, achieved more realistic results with the NSL-KDD dataset in comparison with the KDD CUP 99. This was

Table 4: The distribution of the network traffic between different categories before and after noise filtering in both the training and the testing portion of the UNSW-NB15.

Traffic categories	The training portion				The testing portion			
	Before noise filtering		After noise filtering		Before noise filtering		After noise filtering	
	No. of instances	Distribution (%)	No. of instances	Distribution (%)	No. of instances	Distribution (%)	No. of instances	Distribution (%)
Normal	56,000	31.9377	7,503	13.7093	37,000	44.9399	10,045	31.9213
Analysis	2000	1.1406	1,275	2.3296	677	0.8222	534	1.6969
Reconnaissance	10491	5.9831	7,313	13.3622	3,496	4.2462	2,801	8.9011
Shellcode	1133	0.6461	801	1.4635	378	0.4591	292	0.9279
Fuzzers	18184	10.3706	8,253	15.0797	6,062	7.3628	3,851	12.2378
Worm	130	0.0741	10	0.0182	44	0.0534	11	0.0349
Generic	40000	22.8126	5,650	10.3235	18,871	22.9206	7,052	22.4100
DoS	12264	6.9943	8,868	16.2034	4,089	4.9664	2,442	7.7602
Exploits	33393	19.0446	13,672	24.8990	11,132	13.5208	3,981	12.6509
Backdoors	1746	0.9957	1,384	2.5288	583	0.7081	459	1.4586

an expected outcome due to the absence of redundant records in the NSL-KDD dataset. However, in the current manuscript, we will investigate a different form of noise that could take place due to mislabelled instances, which is, in turn, injected only to a single attribute. Furthermore, in this paper, we will use two independent intrusion datasets.

Other efforts such as [24, 25] investigated the effect of noise with regression tasks, which entails the attempts to predict a numerical value. However, in this paper, we are focusing on classification, which entails classifying different instances to one of two or more predefined categories.

Wang [14] used deep learning (DL) IDS with adversaries. Different attack algorithms taken from the image processing field are used in the intrusion detection field mainly with the NSL-KDD dataset. DL techniques are vulnerable to adversarial imperceptible malicious examples in the image classification domain. Adversarial examples are noisy data specially crafted to attack the DL model. The Jacobian-based saliency map attack (JSMA) and fast-gradient sign method (FGSM) were used to inject noisy data. Furthermore, the paper injected noisy data in the most salient features, which are the features that have the most influence on the results.

Table 5 summarizes several recent journal articles from different publishers that discuss the subject of using ML methods in intrusion detection.

3. Proposed Methodology

This paper empirically investigates the robustness of ML algorithms on intrusion-related datasets with and without the presence of noise. The paper also examines the effectiveness of noise filtering algorithms with different ML algorithms. This section discusses several intrusion-related datasets with more focus on the datasets that are used in this paper, an explanation of the proposed set of experiments, noise injection, noise filtering, evaluation metrics, and the tool that is used to conduct the experiments.

3.1. Datasets. Intrusion-related datasets are used to train and test the proposed IDS models. KDD CUP 99 and NSL-KDD have been extensively used in intrusion detection-related research studies [27]. Table 6 compares the major datasets related to intrusion detection [6, 10, 27, 29, 35, 36].

3.1.1. DARPA 1998. The DARPA 1998 dataset was created within the Intrusion Detection Evaluation Project. It consists of two sets of data, one for training and the other for testing. The DARPA 1998 dataset is one of the early datasets that were publicly available [37].

3.1.2. KDD CUP 99. Knowledge Discovery in Databases (KDD) CUP 99 was derived from the DARPA 1998 dataset. It was explicitly generated to develop ML, classification, and clustering algorithms with more focus on security issues [37]. KDD CUP 99 was described as a "de facto benchmark for evaluating the performance of intrusions detection algorithm" [32]. There is an issue associated with KDD CUP 99 dataset, which is redundancy.

There are 78% redundant records in the training dataset and 75% redundant records in the testing dataset [38]. Redundant records in the KDD CUP 99 dataset negatively affect the performance of the classifiers, making it biased to the more frequently repeated records.

3.1.3. NSL-KDD. The NSL-KDD was initially generated to overcome the drawbacks of the KDD CUP 99 dataset [27]. NSL-KDD was generated based on the KDD CUP 99, but NSL-KDD does not have redundant records [31]. This makes NSL-KDD more suitable to evaluate ML classifiers. NSL-KDD has the same 41 attributes from the KDD CUP 99 dataset. Table 7 lists the different data files of the NSL-KDD dataset.

It can be noticed from Table 7 that the number of attributes for both KDD and NSL-KDD is 42. In Table 6, the number of attributes is 41 and this additional attribute is the class, also called the label attribute. Details about the NSL-KDD's attributes such as the type and the description are illustrated in Table 8.

Table 5: Summary of papers that discusses the use of ML with intrusion detection.

#	Reference	Journal/publisher	Title	Description	ML algorithms	Datasets	Noise	Evaluation metrics
1	[26]	IEEE Access/ IEEE	Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection	A performance comparison between different ML algorithms	(i) SVM (linear) (ii) SVM radial basis function (RBF) (iii) Random forest (iv) Extreme learning machine (ELM)	NSL-KDD	No noise injection or filtering	(i) Accuracy (ii) Precision (iii) Recall
2	[27]	IEEE Transactions on Emerging Topics in Computational Intelligence/IEEE	A deep learning approach to network intrusion detection	Utilizing a DL approach to optimize the intrusion detection	DL	(i) KDD CUP 99 (ii) NSL-KDD	No noise injection or filtering	(i) Accuracy (ii) Precision (iii) Recall (iv) False alarm (v) *F*-score
3	[28]	IEEE Access/ IEEE	An improved intrusion detection algorithm based on GA and SVM	Using a novel intrusion detection algorithm based on GA and SVM to optimize intrusion detection accuracy and detection rate while decreasing the false positive and the training time	(i) SVM (ii) GA	KDD CUP 99	No noise injection or filtering	(i) Detection rate (DR) (ii) False-positive rate (FPR) (iii) False-negative rate (FNR)
4	[29]	IEEE Access/ IEEE	A deep learning approach for intrusion detection using recurrent neural networks	Utilizing a DL-based approach for intrusion detection using RNN and compares the results with other ML algorithms	Deep learning (DL) using recurrent neural networks (RNNs)	NSL-KDD	No noise injection or filtering	(i) Accuracy (ii) True-positive rate (TPR) (iii) False-positive rate (FPR)
5	[30]	Journal of Big Data/Springer	Intrusion detection model using machine learning algorithm on big data environment	The high dimensionality of the big data complicates the process of conducting accurate classification. The paper introduced an IDS model based on ML for big data. ChiSqSelector is used for feature selection and SVMWithSGD is used to conduct the classification.	SVM	KDD CUP 99	No noise injection or filtering	(i) Area under curve (AUROC) (ii) Area under precision-recall curve (AUPR)

Table 5: Continued.

#	Reference	Journal/publisher	Title	Description	ML algorithms	Datasets	Noise	Evaluation metrics
6	[31]	Knowledge-Based Systems/ Elsevier BV	An effective intrusion detection framework based on SVM with feature augmentation	The empirical results showed that feature augmentation helped to obtain more concise training data, which positively influenced the accuracy of the SVM algorithm	SVM	NSL-KDD	No noise injection or filtering	(i) Accuracy (ii) Detection rate (DR) (iii) False alarm rate (FAR)
7	[32]	Future Generation Computer Systems/Elsevier BV	A novel statistical technique for intrusion detection systems	A statistical IDS based on least square SVM (LS-SVM)	LS-SVM	KDD CUP 99	No noise injection or filtering	(i) Precision (ii) Recall (iii) *F*-value
8	[33]	International Journal of Network Management/ Wiley	A deep learning method to detect network intrusion through flow-based features	An IDS based on DL designed to classify network traffic into normal and abnormal using a two-dimensional feature vector	DL	(i) ISCX 2012 (ii) CICIDS 2017	No noise was injected during the experimentation	(i) Precision (ii) Recall (iii) *F*1-score (iv) False alarm rate (FAR) (v) Accuracy
9	[14]	IEEE Access/ IEEE	Deep learning-based intrusion detection with adversaries	The paper used different attack algorithms that were specifically developed to impact the classification accuracy within the image classification domain. The effectiveness of these attack algorithms tends to vary when applied on the intrusion detection dataset.	DL	NSL-KDD	The noise was injected using certain attacks such as JSMA	(i) Accuracy (ii) Precision (iii) Recall (iv) False alarm (v) *F*-score
10	[34]	Wireless Networks/ Springer US	A novel support vector machine-based intrusion detection system for mobile ad hoc networks	An IDS based on SVM that can effectively detect DoS attacks in MANETs. This is achieved by detecting malicious nodes, which highly affects the performance of MANETs.	SVM	No dataset was used. The proposed solution was tested with three routing protocols: AODV, OLSR, and DSR.	No noise injection or filtering	(i) Detection rate (DR) (ii) Mean packet delivery ratio (PDR) (iii) Average end-to-end delay (EED)

Table 6: Comparison between different datasets related to intrusion detection.

No.	Dataset	Creation date	Based on	Deployment	Attack categories	No. of attributes	No. of records
1	DARPA 1998	1998	—	NIDS	DoS Probe R2L U2R	41	Not mentioned
2	KDD CUP 99	1999	DARPA 1998	NIDS	DoS Probe R2L U2R	41	4,898,431
3	NSL-KDD	2009	KDD CUP 99	NIDS	DoS Probe R2L U2R	41	148,517
4	UNSW-NB15	2015	—	NIDS	Analysis Reconnaissance Shellcode Fuzzers Worm Generic DoS Exploits Backdoors	45	2,540,044
5	ADFA	2013	—	HIDS	Hydra-FTP Hydra-SSH Add-user Java-Meterpreter Web-shell	Not mentioned	5,773 (Windows) 5,800 (Linux)

Table 7: The data files included in the NSL-KDD dataset.

No.	NSL-KDD data files	Description	No. of attributes	No. of instances	No. of normal instances	No. of anomaly instances
1	KDDTest+	Contains the full testing set	42	22,544	9,711	12,833
2	KDDTest-21	A subset of the KDDTest+, without records of difficulty level 21 out of 21	42	11,850	2,152	9,698
3	KDDTrain+	Contains the full training set	42	125,973	67,343	58,630
4	KDDTrain+_20Percent	Represents a 20% subset of the KDDTrain+	42	25,192	13,449	11,743

Table 8: A list of the NSL-KDD attributes.

No.	Attribute	Type
1	Duration	Numeric
2	Protocol_type	Nominal
3	Service	Nominal
4	Flag	Nominal
5	Src_bytes	Numeric
6	Dst_bytes	Numeric
7	Land	Nominal
8	Wrong_fragment	Numeric
9	Urgent	Numeric
10	Hot	Numeric
11	Num_failed_logins	Numeric
12	Logged_in	Nominal
13	Num_compromised	Numeric
14	Root_shell	Numeric
15	Su_attempted	Numeric
16	Num_root	Numeric
17	Num_file_creations	Numeric
18	Num_shells	Numeric

Table 8: Continued.

No.	Attribute	Type
19	Num_access_files	Numeric
20	Num_outbound_cmds	Numeric
21	Is_host_login	Nominal
22	Is_guest_login	Nominal
23	Count	Numeric
24	Srv_count_	Numeric
25	Serror_rate	Numeric
26	Srv_serror_rate	Numeric
27	Rerror_rate	Numeric
28	Srv_rerror_rate	Numeric
29	Same_srv_rate	Numeric
30	Diff_srv_rate	Numeric
31	Srv_diff_host_rate	Numeric
32	Dst_host_count	Numeric
33	Dst_host_srv_count	Numeric
34	Dst_host_same_srv_rate	Numeric
35	Dst_host_diff_srv_rate	Numeric
36	Dst_host_same_src_port_rate	Numeric
37	Dst_host_srv_diff_host_rate	Numeric
38	Dst_host_serror_rate	Numeric
39	Dst_host_srv_serror_rate	Numeric
40	Dst_host_rerror_rate	Numeric
41	Dst_host_srv_rerror_rate	Numeric
42	Class	Nominal

3.1.4. UNSW-NB15. The University of New South Wales created the UNSW-NB15 dataset for evaluating new NIDSs. 100 GB of raw network traffic was collected to build the UNSW-NB15 dataset. The UNSW-NB15 dataset consists of ten categories of traffic; one normal and nine categories represent different forms of attacks [8]. However, ML-based techniques perform better with both KDD CUP 99 and NSL-KDD. This is due to two reasons. Firstly, many values of normal and malicious instances are almost the same in the UNSW-NB15 dataset, whereas there is a relatively reasonable difference between the normal and the malicious values in both KDD CUP 99 and NSL-KDD. Secondly, the data distribution of the UNSW-NB15 dataset is nearly the same, while it is different in the KDD CUP 99 and NSL-KDD due to the existence of new attacks in the testing set, which helps to differentiate between normal and abnormal instances when running ML algorithms [8]. Tables 9 and 10 illustrate the number of attributes, the total number of instances, the number of normal instances, and the number of malicious instances. In addition, the attributes' names and types are also illustrated.

3.1.5. ADFA. The ADFA dataset was generated to be tested for host-level intrusion detection. It was generated by the Australian Defense Academy (ADFA). The ADFA dataset encompasses two different platforms, Windows (ADFA-WD) and Linux (ADFA-LD) [10]. There are five categories of attacks within this dataset, which can be found in Table 6.

3.2. Dataset Preprocessing. This paper utilizes two intrusion detection datasets, which are the NSL-KDD dataset and the UNSW-NB15 dataset. The NSL-KDD dataset can be used directly in the experiments as the class value of the NSL-KDD is nominal, whereas the class value of the UNSW-NB15 dataset is numeric. Therefore, the class values of the UNSW-B15 dataset must be converted from numeric to nominal.

WEKA data mining tool provides a numeric to nominal filter that can be used to convert numeric values to nominal values. Typically, the class is located as the last attribute in the dataset [20]. In the NSL-KDD dataset, the last attribute, which is attribute number 42, contains two distinct nominal values, called normal and anomaly. On the other hand, the class value of the UNSW-NB15 dataset, which is attribute number 45, consists of numeric values.

3.3. Proposed Experiments. Generally, the proposed methodology consists of six distinct sets of experiments illustrated in Figure 3. Each set of experiments aims to empirically test a certain aspect of ML-based IDS performance in response to the noise.

The first set of experiments involves generating the baseline, which will be needed to investigate the potential impact of noise and the effectiveness of noise filtering techniques on ML-based IDSs. Furthermore, it will be used to measure the influence of employing ensemble learning algorithms. The results of the subsequent stages will be compared with the results of this stage.

In the second set of experiments, interquartile noise filtering algorithm will be applied to the datasets. This filter will help to identify outliers and extreme values from the datasets. The results of this phase will be compared with the baseline.

Table 9: The training and the testing datasets of the UNSW-NB15.

No.	UNSW-NB15 data files	No. of attributes	No. of instances	No. of normal instances	No. of attack instances
1	UNSW_NB15_training-set	45	82,332	37,000	45,332
2	UNSW_NB15_testing-set	45	175,341	56,000	119,341

Table 10: List of UNSW-NB15 attributes.

No.	Attribute	Type
1	Id	Numeric
2	Dur	Numeric
3	Proto	Nominal
4	Service	Nominal
5	State	Nominal
6	Spkts	Numeric
7	Dpkts	Numeric
8	Sbytes	Numeric
9	Dbytes	Numeric
10	Rate	Numeric
11	Sttl	Numeric
12	Dttl	Numeric
13	Sload	Numeric
14	Dload	Numeric
15	Sloss	Numeric
16	Dloss	Numeric
17	Sinpkt	Numeric
18	Dinpkt	Numeric
19	Sjit	Numeric
20	Djit	Numeric
21	Swin	Numeric
22	Stcpb	Numeric
23	Dtcpb	Numeric
24	Dwin	Numeric
25	Tcprtt	Numeric
26	Synack	Numeric
27	Ackdat	Numeric
28	Smean	Numeric
29	Dmean	Numeric
30	Trans_depth	Numeric
31	Response_body_len	Numeric
32	Ct_srv_src	Numeric
33	Ct_state_ttl	Numeric
34	Ct_dst_	Numeric
35	Ct_src_dport_ltm	Numeric
36	Ct_dst_sport_ltm	Numeric
37	Ct_dst_src_ltm	Numeric
38	Is_ftp_login	Numeric
39	Ct_ftp_cmd	Numeric
40	Ct_flw_http_mthd	Numeric
41	Ct_src_ltm	Numeric
42	Ct_srv_dst	Numeric
43	Is_sm_ips_ports	Numeric
44	Attack_cat	Nominal
45	Label	Numeric

In the third set of experiments, noise is injected into the intrusion datasets with the following percentages: 5%, 10%, 20%, and 30%. Then noisy data will be inserted into the ML models. The results will be compared with the baseline obtained from the first stage. This is useful to test the robustness of ML algorithms against noise.

The fourth set of experiments entails conducting noise filtering by excluding noisy instances and then injecting different levels of noise, which are 5%, 10%, 20%, and 30%. This will help to study the influence of noise on the ML algorithms that run on intrusion datasets with the absence of outlier and extreme value instances.

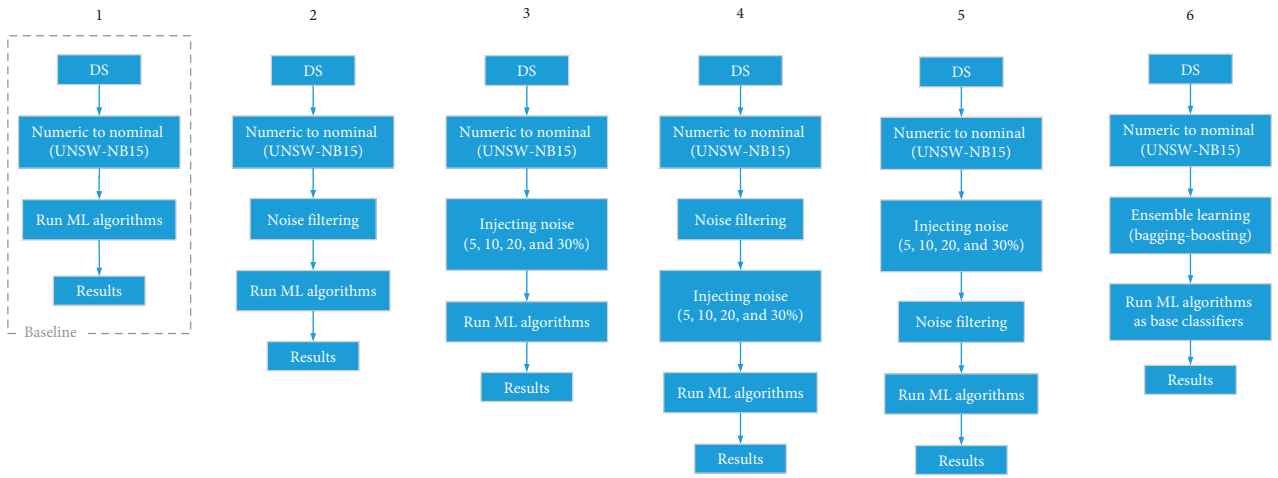

Figure 3: The overall methodology used in this paper.

(1) Load the training portion of the intrusion dataset
(2) Convert the label values from numeric to nominal (only for the UNSW-NB15)
(3) Train the ML algorithm on the training dataset
(4) Run the ML algorithm on the testing dataset
(5) Document the results
(6) Revert to step 3 using the other ML algorithms

Pseudocode 1: The first set of experiments (baseline experiment).

(1) Load the training portion of the intrusion dataset
(2) Convert the label values from numeric to nominal (only for the UNSW-NB15)
(3) Conduct noise filtering by removing the outliers and the extreme values
(4) Train the ML algorithm on the training dataset
(5) Run the ML algorithm on the testing dataset
(6) Document the results
(7) Revert to step 4 using the other ML algorithms

Pseudocode 2: The second set of experiments (noise filtering).

(1) Load the training portion of the intrusion dataset
(2) Convert the label values from numeric to nominal (only for UNSW-NB15)
(3) Conduct noise injection by manipulating the labels of the training instances (5, 10, 20, and 30%)
(4) Train the ML algorithm on the noisy training dataset
(5) Run the ML algorithm on the testing dataset
(6) Document the results
(7) Revert to step 3 until each of the ML algorithms is trained and tested with the different levels of noise

Pseudocode 3: The third set of experiments (noise injection).

(1) Load the training portion of the intrusion dataset
(2) Convert the label values from numeric to nominal (only for UNSW-NB15)
(3) Conduct noise filtering by removing the outliers and the extreme values
(4) Conduct noise injection by manipulating the labels of the training instances (5, 10, 20, and 30%)
(5) Train the ML algorithm on the noisy training dataset
(6) Run the ML algorithm on the testing dataset
(7) Document the results
(8) Revert to step 4 until each of the ML algorithms is trained and tested with the different levels of noise

Pseudocode 4: The fourth set of experiments (noise filtering then injection).

(1) Load the training portion of the intrusion dataset
(2) Convert the label values from numeric to nominal (only for UNSW-NB15)
(3) Conduct noise injection by manipulating the labels of the training instances (5, 10, 20, and 30%)
(4) Conduct noise filtering by removing the outliers and the extreme values
(5) Train the ML algorithm on the noisy training dataset
(6) Run the ML algorithm on the testing dataset
(7) Document the results
(8) Revert to step 4 until each of the ML algorithms is trained and tested with the different levels of noise

Pseudocode 5: The fifth set of experiments (noise injection then filtering).

(1) Load the training portion of the intrusion dataset
(2) Convert the label values from numeric to nominal (only for the UNSW-NB15)
(3) Initialize the ensemble learning method (bagging-boosting)
(4) Use an ML algorithm as a base classifier
(5) Train the ensemble of classifiers on the dataset
(6) Test the ensemble of classifiers on the test dataset
(7) Document the results
(8) Revert to step 3 until each ML algorithms is used with both ensemble learning methods

Pseudocode 6: The sixth set of experiments (ensemble of classifiers).

In the fifth set of experiments, noise injection is applied before noise filtering. In this experiment, noise is injected in the form of manipulating the labels of a given percentage of the training set. Noise filtering is conducted by removing outliers and extreme values.

The sixth set of experiments studies the influence of using ensemble learning techniques on the accuracy of the ML algorithms when applying it to the intrusion datasets.

A generic description of the conducted experiments is presented in the form of pseudocodes.

3.4. Noise Injection. The presence of noise might negatively affect the accuracy of ML algorithms. Noise can take different forms such as mislabelled data or incorrectly classified instances.

To obtain more accurate outcome, noisy data must be filtered first as working with noise-free data helps to avoid potential issues such as overfitting [36].

Some experiments in this paper require injecting different levels of noise into the datasets. In WEKA, noise can be injected into the datasets via the AddNoise filter. AddNoise filter "changes a percentage of a given nominal attribute's values" [20]. The default value of noise injection in the AddNoise filter is 10%. However, it can be adjusted to any other percentage.

3.5. Evaluation Metrics. It is critical to specify the adequate metrics that will be used to evaluate the ML algorithms with the intrusion detection datasets. However, the evaluation metrics should be relevant to the subject under examination. Ahmad et al. [26] used three evaluation metrics to compare the performance of SVM, RF, and EL for the IDSs. These evaluation metrics are accuracy, precision, and recall.

In this paper, the authors will use the same evaluation metrics. The classification of each testing instance falls within four cases (usually referred collectively as the confusion matrix). The four cases are as follows:

(i) True positive (TP): malicious instances that were correctly classified as malicious by the IDS
(ii) True negative (TN): normal instances that were correctly classified as normal by the IDS
(iii) False positive (FP): normal instances that were incorrectly classified by the IDS as malicious
(iv) False negative (FN): malicious instances that were incorrectly classified by the IDS as normal

The above four values are used to calculate the evaluation metrics [37]. Accuracy represents the ratio of correctly classified instances to the total number of instances in the test dataset:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \quad (14)$$

Precision represents the accuracy of the positive predictions:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \quad (15)$$

Recall is the ratio of positive instances that the classifier manages to classify correctly:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (16)$$

Accuracy is the most relevant metric to the objective of this paper as it indicates the success percentage of the ML algorithm in classifying the testing instances. Classifying the

testing instances involves determining whether it represents an intrusion or not. This will help to define the factors that can influence, either positively or negatively, the accuracy of the ML algorithms. Therefore, only the accuracy metric will be used to assess and compare the different ML algorithms from different experiments with the baseline, and the other metrics will be presented to provide a more holistic perspective on the effectiveness of the ML algorithm.

3.6. Tool Used in This Research (WEKA). This paper utilizes Waikato Environment for Knowledge Analysis (WEKA) version 3.8.3 as the main tool to conduct empirical experiments. WEKA is an open-source tool written in Java and designed to run ML algorithms. It was developed by the University of Waikato, New Zealand [39]. WEKA provides the ability to run many ML algorithms such as J48 (a DT flavor), RF, and SVM on any dataset.

3.7. The ML Algorithms' Parameters. Each ML algorithm has its own set of parameters that determines exactly how it operates. These parameters can be fine-tuned to enhance the performance of the ML-based classifiers. Table 11 lists the assigned parameters for the ML algorithms as used in the conducted empirical experiments.

3.8. Device Specifications. The device that we have used to conduct all the experiments is Dell OptiPlex 9020 with Intel i7-4770 CPU, 12 GB of random access memory (RAM), 500 GB hard disk drive (HDD), and the operating system (OS) is Microsoft Windows 10 Pro.

4. Results and Discussions

This section discusses the conducted experiments and the obtained results and analysis of the results.

4.1. Baseline Experiment. The main objective of this set of experiments is to determine the data that will be used in subsequent experiments and to establish a baseline to compare it with the results of subsequent experiments. Therefore, three baseline experiments are conducted. Each of these baselines will be constructed using the same ML algorithms. However, the only difference is in the method of using the dataset as the following:

(i) Training the model using the training dataset and testing it on a separate testing dataset.

(ii) Each dataset will be divided into three equal parts. Two-thirds will be used for training and one-third will be used for testing. This method is referred to as the percentage split. In this paper, all ML algorithms were trained on the same two-thirds of the datasets and all were tested on the same one-third that was dedicated for testing.

(iii) The entire training data of each dataset will be used for training and testing the constructed models. This will be achieved using an iterative manner called cross-validation, which entails dividing the dataset to a certain number of equal partitions (folds). All of these partitions have the same size. Then, the tests are performed by selecting one of these folds for testing and the remaining folds for training. This is achieved in an iterative manner. In this paper, we have divided the training portion to 10-folds.

The results of the first set of experiments are aggregated in Table 12 which includes details about the testing mode, the used ML methods, the names of the intrusion datasets, and the evaluation metrics. Figure 4(a) compares the results using different testing modes using the NSL-KDD dataset while Figure 4(b) compares the results using different testing modes using the UNSW-NB15 dataset.

In the supplied test set, both KDDTrain+ and KDDTest+ from the NSL-KDD were used for training and testing, respectively. On the other hand, the UNSW_NB15_training-set and the UNSW_NB15_testing-set from UNSW-NB15 were used for training and testing, respectively. In percentage split and cross-validation, the KDDTrain+ from NSL-KDD and UNSW_NB15_training-set from UNSW-NB15 were used.

With the NSL-KDD dataset, it can be noticed that the test mode highly affects the classification accuracy of the used ML algorithms. However, the observations are illustrated in the following points:

(i) The classification accuracy of the ML algorithms tends to degrade when a supplied test set is used, whereas the accuracy of the other test modes, which are percentage split and cross-validation, is high

(ii) With the supplied test set, DT (J48) achieved the highest accuracy (81.5339%), whereas RF achieved the highest accuracy with the percentage split (99.9019%) and cross-validation (99.9174%)

(iii) The NB scored the lowest accuracy in all test modes, especially with the supplied test mode (76.1178%).

The observations from the accuracy results when the UNSW-NB15 dataset is used are as follows:

(i) DT, SVM, and ANN achieved the same accuracy results across all the experiments, which is 100%.

(ii) RF algorithm achieved less accuracy results with the supplied test set (98.4903%) and it achieved (100%) classification accuracy with both percentages split and cross-validation.

(iii) NB algorithm, similarly, achieved higher accuracy results with percentage split and cross-validation, which are 92.1397% and 92.4809%, respectively, whereas it achieved the lowest classification accuracy (87.4435%) with the supplied test set.

(iv) Typically, gaining a 100% accuracy means that the ML algorithm managed to correctly classify all instances within the testing dataset.

(v) Generally, the results seem to support the claim of Moustafa et al. [8], about the inadequacy of using ML algorithms with the UNSW-NB15 dataset. The

Table 11: Used parameters values for each ML algorithm in WEKA.

ML algorithm	Parameter name	Value
DT	Batch size	100
	Binary splits	False
	Collapse tree	True
	Confidence factor	0.25
	Debug	False
	Do not check capabilities	False
	Do not make split point actual value	False
	Min num obj	2
	Num decimal places	2
	Num folds	3
	Reduced error pruning	False
	Save instance data	False
	Seeds	1
	Subtree raising	True
	Unpruned	False
	Use Laplace	False
	Use MDL correction	True
SVM	Batch size	100
	Build calibration models	False
	C	1
	Calibrator	Default
	Checks turned off	False
	Debug	False
	Do not check capabilities	False
	Epsilon	$1.0E-12$
	Filter type	Normalize training data
	Kernel	Default
	Num decimal places	2
	Num folds	−1
	Random seed	1
	Tolerance parameter	0.001
NB	Batch size	100
	Debug	False
	Display model in old format	False
	Do not check capabilities	False
	Num decimal places	2
	Use kernel estimator	False
	Use supervised discretization	False
RF	Bag size percent	100
	Batch size	100
	Break ties randomly	False
	Calc out of bag	False
	Compute attribute importance	False
	Debug	False
	Do not check capabilities	False
	Max depth	0
	Num decimal places	2
	Num execution slots	1
	Num features	0
	Num iterations	100
	Output out of bag complexity statistics	False
	Print classifiers	False
	Seed	1
	Store out of bag predictions	False

Table 11: Continued.

ML algorithm	Parameter name	Value
ANN	GUI	False
	Auto build	True
	Batch size	100
	Debug	False
	Decay	False
	Do not check capabilities	False
	Hidden layers	A
	Learning rate	0.3
	Momentum	0.2
	Nominal to binary filter	True
	Normalize attributes	True
	Normalize numeric class	True
	Num decimal places	2
	Reset	True
	Seed	0
	Training time	500
	Validation set size	0
	Validation threshold	20

Table 12: Constructing the baseline experiments with different testing modes.

Test mode	ML algorithm	NSL-KDD			UNSW-NB15		
		Accuracy	Precision	Recall	Accuracy	Precession	Recall
Supplied test set	DT (J48)	81.5339	0.858	0.815	100	1.000	1.000
	RF	80.4516	0.852	0.805	98.4903	0.985	0.985
	SVM	75.3948	0.802	0.754	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	Naïve Bayes	76.1178	0.809	0.761	87.4435	0.884	0.874
Percentage split	DT (J48)	99.7245	0.997	0.997	100	1.000	1.000
	RF	99.9019	0.999	0.999	100	1.000	1.000
	SVM	97.4574	0.975	0.975	100	1.000	1.000
	ANN	98.6085	0.986	0.986	100	1.000	1.000
	Naïve Bayes	90.5933	0.907	0.906	92.1397	0.930	0.921
Cross-validation	DT (J48)	99.7817	0.998	0.998	100	1.000	1.000
	RF	99.9174	0.999	0.999	100	1.000	1.000
	SVM	97.405	0.974	0.974	100	1.000	1.000
	ANN	98.5378	0.985	0.985	100	1.000	1.000
	Naïve Bayes	90.3829	0.905	0.904	92.4809	0.932	0.925

reasons for this inadequacy are mentioned in Section 3.1.4.

Subsequent experiments will be conducted using the supplied test set only rather than splitting a single dataset or cross-validation as this provides a better understanding of different factors that might affect the accuracy of the ML algorithms on the intrusion dataset. For the NSL-KDD dataset, KDDTrain+ and KDDTest+ will be used, to train and test the model, respectively. On the other hand, for the UNSW-NB15, UNSW_NB15_training-set and the UNSW_NB15_testing-set will also be used to train and test the model, respectively.

4.2. Noise Filtering Experiment. The second set of experiments includes using noise filtering by removing outliers and extreme values from the training and the testing datasets. The results from this set of experiments are listed in Table 13. In addition, the classification accuracy results are compared with the baseline. Figures 5(a) and 5(b) show the results of using the filtered NSL-KDD dataset and the filtered UNSW-NB15 dataset, respectively.

The effects of removing the outliers and extreme instances on the accuracy of the filtered NSL-KDD dataset are as follows:

(i) Removing the outliers and the extreme value instances slightly decreased (−0.8223%) the classification accuracy of the DT (J48) algorithm in comparison with the baseline experiment

(ii) The accuracy of the RF slightly increased (+0.2226%) after removing the outliers and the extreme instances.

(iii) For SVM, ANN, and NB, the removal of outliers and noisy instances has a noticeable positive influence. The classification accuracy of the SVM algorithm

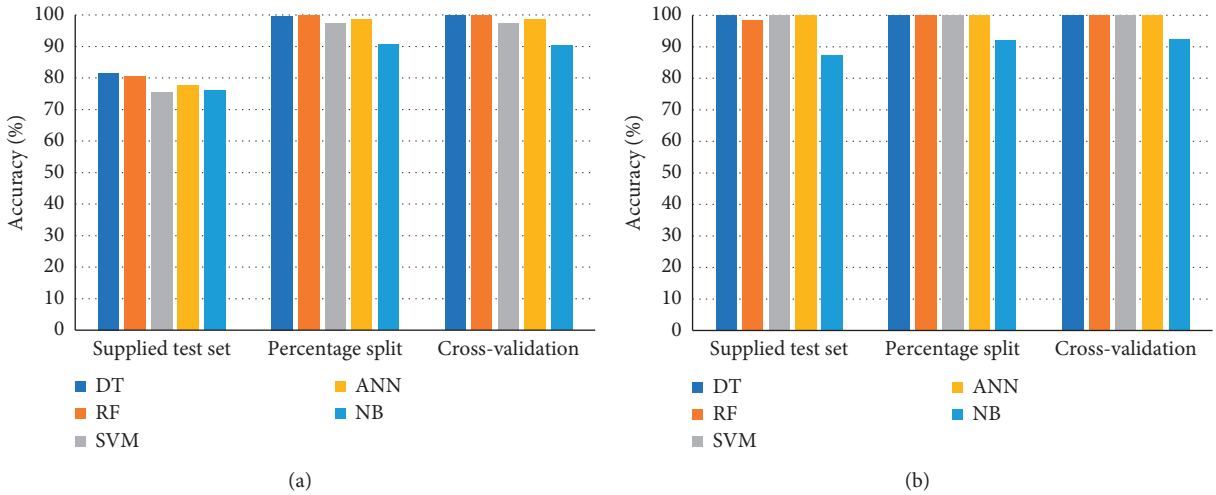

Figure 4: The baseline classification accuracy results of the intrusion datasets. (a) NSL-KDD. (b) UNSW-NB15.

increased by +5.2419%, whereas the accuracy of ANN increased by +2.922%, and for the NB, the accuracy increased by +4.3691%.

The accuracy values of the UNSW-NB15 dataset after conducting the instance reduction by removing the outliers and extreme value instances are as follows:

(i) For DT (J48), SVM, and ANN, the accuracy values of the filtered dataset remained as the baseline (100%). This means that the removal of the outliers and the extreme value instances did not affect the classification accuracy of these ML algorithms when used with the UNSW-NB15 intrusion dataset.

(ii) The accuracy of the RF algorithm was slightly improved by +1.4493% due to the removal of the outliers and the extreme value instances.

(iii) On the other hand, the accuracy value of the NB algorithm was decreased by −11.9827% as a result of noise filtering. This means the NB algorithm performs better with the presence of the outliers and extreme values in the UNSW-NB15 dataset.

4.3. Noise Injection Experiment. The third set of experiments aims to investigate the extent of the effect of noise over the five considered ML algorithms when applying them over the intrusion-related datasets. In this set of experiments, different levels of noise (5%, 10%, 20%, and 30%) are added to each training dataset, and no noise will be injected into the testing dataset. The noise will be injected using WEKA's AddNoise filter.

The noise will be injected solely into the class attribute. This will help to investigate the classification accuracy of the ML algorithms on the testing dataset, knowing that it was trained on a noisy dataset. The results after injecting the different levels of noise are illustrated in Table 14. Figures 6(a) and 6(b) show the results of using noise with the NSL-KDD dataset and the UNSW-NB15 dataset, respectively.

Noise injection affected the accuracy of the ML algorithms tested on the NSL-KDD dataset. The accuracy of some algorithms decreases with the increase of noise, and the accuracy of certain algorithm increases with the increase of noise. The observations from this set of experiments are as follows:

(i) The accuracy of the DT (J48) algorithm is slightly influenced by the different levels of noise compared to the baseline, which is 81.5339%. DT (J48) achieved better accuracy with 10% noise compared to 5% noise and it also achieved better accuracy with 30% noise in comparison with 20% noise injection.

(ii) The accuracy of the RF algorithm continuously decreased with the increase of noise (79.3204%, 77.2179%, 72.3208%, and 70.6574%), respectively, when compared with the baseline (80.4516%).

(iii) Interestingly, with the SVM algorithm, the increase of noise causes an increase in the classification accuracy when compared with the baseline (75.3948%). With the different levels of noise, the SVM algorithm scored the following accuracy results (76.2731%, 76.2376%, 78.7615%, and 78.7216%). It can be concluded that SVM managed to correctly classify more testing instances, knowing that it was originally trained on a noisy (misclassified) data.

(iv) The baseline accuracy of the ANN algorithm is 77.7147%, and with the different levels of noise, the accuracy of the ANN algorithm varies with a slight increase and decrease compared to the baseline (79.2273%, 76.4061%, 77.4308%, and 75.2928%). The accuracy of the ANN algorithm behaves similar to the DT (J48).

(v) Similar to the RF algorithm, the accuracy of the NB algorithm decreases with the increase of noise when compared with the baseline value (76.1178%),

Table 13: Results after conducting the noise filtering.

Noise filtering	ML algorithm	NSL-KDD			UNSW-NB15		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline	DT (J48)	81.5339	0.858	0.815	100	1.000	1.000
	RF	80.4516	0.852	0.805	98.4903	0.985	0.985
	SVM	75.3948	0.802	0.754	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	NB	76.1178	0.809	0.761	87.4435	0.884	0.874
Filtered	DT (J48)	80.7116	0.844	0.807	100	1.000	1.000
	RF	80.6742	0.844	0.807	99.9396	0.999	0.999
	SVM	80.6367	0.796	0.806	100	1.000	1.000
	ANN	80.6367	0.796	0.806	100	1.000	1.000
	NB	80.4869	0.746	0.805	75.4608	0.800	0.755

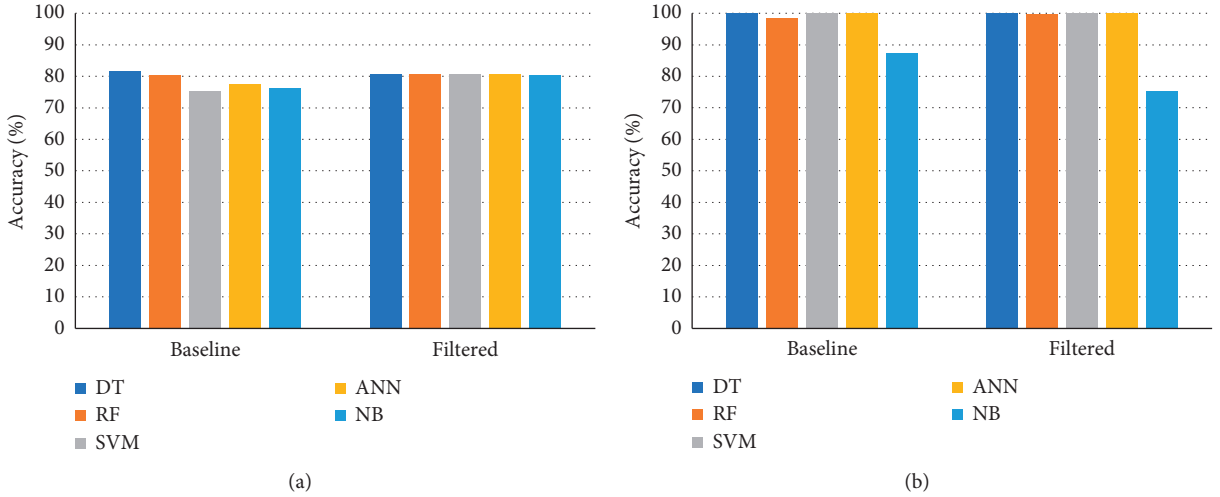

Figure 5: The classification accuracy results of the filtered datasets. (a) NSL-KDD. (b) UNSW-NB15.

except that, with the 30% noise injection, the classification accuracy increased when compared with 10% and 20% noise injection (74.9867%, 73.9354%, 71.6776%, and 74.4677%).

The effect of noise injection on the accuracy of the ML algorithms tested on the UNSW-NB15 dataset can be summarized as follows:

(i) The classification accuracy of the DT (J48) algorithm is not affected by the 5%, 10%, and 20% noise and it remained as the baseline (100%). However, the accuracy was intensely affected when a 30% noise was injected. The DT accuracy recorded 66.9412% classification accuracy. This means that the DT algorithm with the UNSW-NB15 dataset can maintain high classification accuracy until a certain limit where it starts to heavily degrade and cannot be used reliably.

(ii) With the RF algorithm, the classification accuracy continuously decreased with the increase of noise. It can be concluded that RF algorithm is sensitive to noise and it should be trained on a correctly labelled dataset in order to obtain better classification outcomes.

(iii) The classification accuracy of the SVM algorithm was not affected by any level of noise. The classification accuracy remained as the baseline (100%) even after injecting different levels of noise. This means that the SVM algorithm managed to successfully classify all testing instances even though it was trained on a mislabelled dataset. Therefore, it can be considered as the most robust algorithm in this set of experiments.

(iv) The classification accuracy of the ANN algorithm was slightly decreased with the increase in noise. The baseline accuracy of the ANN algorithm is 100%, and the accuracy results after injecting the different levels of noise are 99.9988%, 100%, 99.9988%, and 99.9818. This means that training the ANN algorithm on noisy dataset does not greatly affect the classification accuracy.

(v) The baseline value of the NB algorithm is 87.4435%, and the accuracy results after inserting the different noise levels are 77.5992%, 77.5883%, 77.2713%, and 77.7511%, respectively. This indicates that the effect of different levels of noise on the NB algorithm is almost identical.

Table 14: Injecting different levels of noise in the intrusion datasets.

Noise (%)	ML algorithm	NSL-KDD			UNSW-NB15		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline 0% noise	DT (J48)	81.5339	0.858	0.815	100	1.000	1.000
	RF	80.4516	0.852	0.805	98.4903	0.985	0.985
	SVM	75.3948	0.802	0.754	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	NB	76.1178	0.809	0.761	87.4435	0.884	0.874
5% noise injected	DT (J48)	80.7887	0.851	0.808	100	1.000	1.000
	RF	79.3204	0.844	0.793	94.8538	0.954	0.949
	SVM	76.2731	0.828	0.763	100	1.000	1.000
	ANN	79.2273	0.826	0.792	99.9988	1.000	1.000
	NB	74.9867	0.805	0.750	77.5992	0.805	0.776
10% noise injected	DT (J48)	80.7976	0.846	0.808	100	1.000	1.000
	RF	77.2179	0.814	0.772	93.9829	0.947	0.940
	SVM	76.2376	0.828	0.762	100	1.000	1.000
	ANN	76.4061	0.809	0.764	100	1.000	1.000
	NB	73.9354	0.800	0.739	77.5883	0.801	0.776
20% noise injected	DT (J48)	79.8084	0.826	0.798	100	1.000	1.000
	RF	72.3208	0.777	0.723	88.8148	0.908	0.888
	SVM	78.7615	0.848	0.788	100	1.000	1.000
	ANN	77.4308	0.813	0.774	99.9988	1.000	1.000
	NB	71.6776	0.791	0.717	77.2713	0.800	0.773
30% noise injected	DT (J48)	80.1011	0.828	0.801	66.9412	0.778	0.669
	RF	70.6574	0.750	0.707	84.4799	0.871	0.845
	SVM	78.7216	0.847	0.787	100	1.000	1.000
	ANN	75.2928	0.802	0.753	99.9818	1.000	1.000
	NB	74.4677	0.801	0.745	77.7511	0.801	0.778

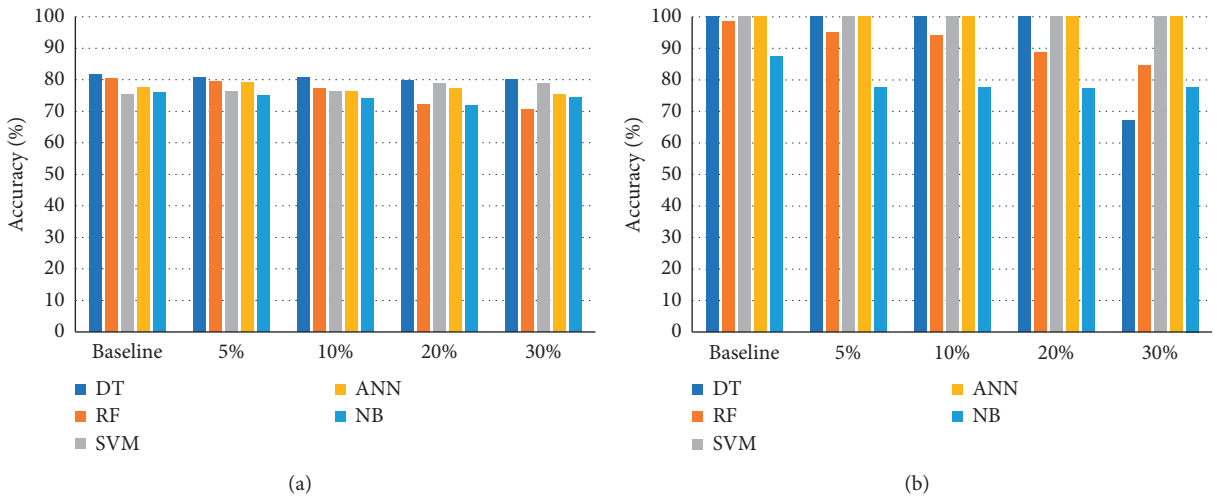

Figure 6: The classification accuracy with different levels of noise injection. (a) NSL-KDD. (b) UNSW-NB15.

Figure 7 illustrates the impact of injecting different levels of noise in the NLS-KDD dataset and the corresponding accuracy of the ML algorithms. It can be noticed that SVM algorithm is the most resilient algorithm to noise injection whereas the RF algorithm is the least resilient algorithm to noise injection.

Figure 8 illustrates the impact of injecting different levels of noise in the UNSW-NB15 dataset and the corresponding accuracy of the ML algorithms. The major degradation in the classification accuracy of the DT algorithm with the 30% noise injection is readily apparent in addition to the continuous decrease in the classification accuracy of the RF algorithm with the increase of the percentage of noise ratio.

4.4. Noise Filtering Then Injection Experiment. In the fourth set of experiments, noise is filtered by removing outliers and extreme value instances and then noise is injected with different levels (5%, 10%, 20%, and 30%) to the intrusion dataset. The third experiment investigated the effect of noise injection. The current experiment inspects the potential

impact of noise injection on filtered intrusion datasets. Table 15 shows the results obtained from this set of experiments. Figures 9(a) and 9(b) show the results of injecting noise on the filtered NSL-KDD and UNSW-NB15 datasets, respectively.

In general, removing outliers and extreme value instances appeared to have a positive influence on certain ML algorithms. Here is the analysis of the accuracy results for the NSL-KDD dataset:

(i) The DT (J48) was slightly impacted by noise when applied on a filtered dataset. The baseline accuracy was 81.5339%, and the accuracy results after injecting the different levels of noise are 80.6742%, 80.6742%, 80.5618%, and 78.5768%. The DT (J48) gave the same accuracy results with the 5% and the 10% noise injection.

(ii) The accuracy results from the third set of experiments illustrate that the accuracy of the RF algorithm has decreased with the increase of noise. In the current experiments, it seems that the RF algorithm achieved better accuracy results. The baseline accuracy of the RF algorithm is 80.4516%, and the accuracy results after filtering the dataset and injecting different levels of noise are 80.4869%, 80.2996%, 81.2734%, and 80.9738%. Therefore, it is obvious that removing outliers and extreme value instances has a positive influence on the RF algorithm when used with a noisy NSL-KDD dataset.

(iii) The accuracy of the SVM and NB algorithms with all different levels of noise injection remained exactly the same (80.4869%), whereas the accuracy of the baseline was 75.3948% and 76.1178% for SVM and NB, respectively. This means that the removal of outliers and extreme values instances eliminated the effect of noisy data.

(iv) For the ANN algorithm, the removal of outliers and extreme values made the effect of the different levels of noise notably limited. The baseline accuracy of the ANN algorithm is 76.1178%, and the accuracy after the filtering and the injecting of the different levels of noise are 80.4869%, 80.4869%, 80.4869%, and 80.4869%.

When ML algorithms were applied on the UNSW-NB15 dataset, after filtering outliers and extreme value instances and injecting noisy data, certain algorithms greatly affected such as DT (J48), RF, and NB:

(i) With DT (J48) algorithm, the accuracy with 5% and 10% noise remained as the baseline, which is 100%. A strong decrease in accuracy occurred when the noise increased to 20% and 30%. In particular, with the 20% noise injection, the classification accuracy dropped to 71.4472%, and with the 30% noise injection, the accuracy of the DT algorithm dropped again to 70.1824%.

(ii) The classification accuracy of the RF algorithm decreased with the increase of noise. The baseline accuracy is 98.4903%, and the accuracy with the different levels of noise are 96.3296%, 96.336%, 92.656%, and 87.6001%.

(iii) Similar to DT, the classification accuracy of the SVM algorithm for 5% and 10% noise levels remained the same as the baseline (100%). However, with the 20% noise injection, the classification accuracy slightly dropped to 99.9841%, and with the 30% noise injection, it dropped to 99.9174%.

(iv) The classification accuracy of the ANN algorithm was oscillated with different levels of noise. The baseline accuracy of the ANN algorithm is 100%. However, it achieved the following accuracy results with the different levels of noise (98.5001%, 99.892%, 99.9682%, and 99.9078%).

(v) With all levels of noise, NB algorithm achieved lower, but very convergent accuracy results. The baseline accuracy was 87.4435%, and the corresponding accuracy with the different levels of noise are 74.285%, 74.2055%, 74.0784%, and 74.0721%.

Figure 10 shows the impact of injecting different levels of noise to the filtered NLS-KDD dataset and the corresponding accuracy of the ML algorithms. It can be noticed that the removal of outliers and extreme value instances and then the manipulation of the instances' labels with different noise percentages have made the classification accuracy of the ML algorithms nearly similar except for the RF algorithm with the 20% noise injection and the DT algorithm with the 30% noise injection.

Figure 11 shows the impact of injecting different levels of noise to the filtered UNSW-NB15 dataset and the corresponding accuracy of the ML algorithms. The resilience of the RF algorithm has remained weak to the different levels of noise injection even after the removal of outliers and the extreme value instances.

4.5. Noise Injection Then Filtering Experiment. This set of experiments aims to investigate the influence of noise injection before filtering the dataset from outliers and extreme values. The difference between this set of experiments and the previous set of experiments is the order of dealing with noise. More specifically, the previous set of experiments conducted noise filtering then injection. However, in this set of experiments, noise injection will be conducted before filtering the noisy instances. Table 16 shows the accuracy, precision, and recall values of this experiment. Figures 12(a) and 12(b) show the results of filtering the datasets after injecting noise on both NSL-KDD and UNSW-NB15 datasets, respectively.

For the NSL-KDD dataset, SVM and the NB algorithms achieved results similar to the previous set of experiments, with all levels of noise removal and injection. The detailed analysis is as follows:

(i) The baseline accuracy of the DT algorithm was 81.5339%, whereas it achieved 77.9026%, 73.97%, 75.2434%, and 66.6292 with the different levels of noise. The DT algorithm achieved higher accuracy

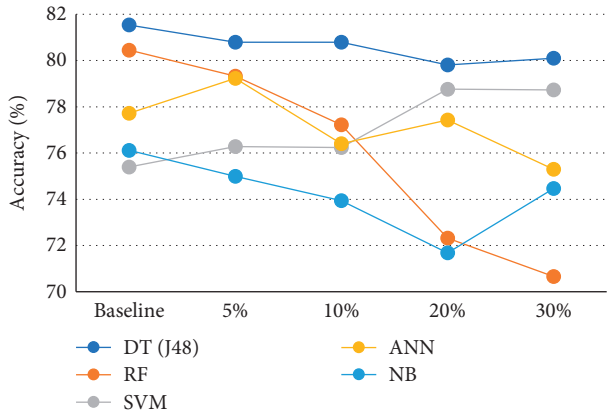

Figure 7: Noise injection in the NSL-KDD dataset and the accuracy of ML algorithms.

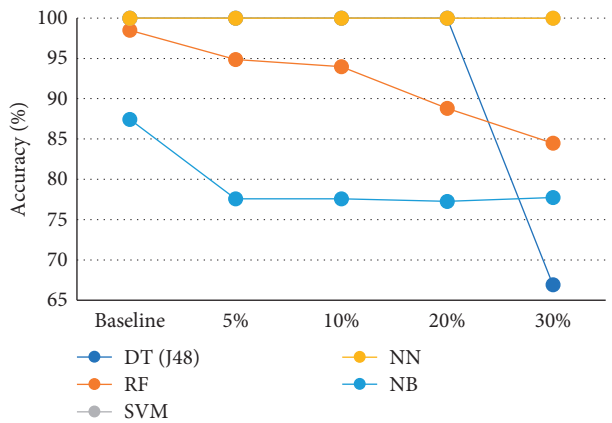

Figure 8: Noise injection in the UNSW-NB15 dataset and the accuracy of ML algorithms.

Table 15: The results of running ML algorithms on a filtered and then injected intrusion dataset.

Noise (%)	ML algorithm	NSL-KDD			UNSW-NB15		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline	DT (J48)	81.5339	0.858	0.815	100	1.000	1.000
	RF	80.4516	0.852	0.805	98.4903	0.985	0.985
	SVM	75.3948	0.802	0.754	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	NB	76.1178	0.809	0.761	87.4435	0.884	0.874
Filtered then 5% noise injected	DT (J48)	80.6742	0.844	0.807	100	1.000	1.000
	RF	80.4869	0.746	0.805	96.3296	0.967	0.963
	SVM	80.4869	0.746	0.805	100	1.000	1.000
	ANN	80.5243	0.760	0.805	98.5001	0.985	0.985
	NB	80.4869	0.746	0.805	74.285	0.795	0.743
Filtered then 10% noise injected	DT (J48)	80.6742	0.844	0.807	100	1.000	1.000
	RF	80.2996	0.721	0.803	96.336	0.967	0.963
	SVM	80.4869	0.746	0.805	100	1.000	1.000
	ANN	80.6367	0.796	0.806	99.892	0.999	0.999
	NB	80.4869	0.746	0.805	74.2055	0.795	0.742
Filtered then 20% noise injected	DT (J48)	80.5618	0.843	0.806	71.4472	0.815	0.714
	RF	81.2734	0.777	0.813	92.656	0.938	0.927
	SVM	80.4869	0.746	0.805	99.9841	1.000	1.000
	ANN	80.6367	0.796	0.806	99.9682	1.000	1.000
	NB	80.4869	0.746	0.805	74.0784	0.794	0.741
Filtered then 30% noise injected	DT (J48)	78.5768	0.662	0.786	70.1824	0.797	0.702
	RF	80.9738	0.804	0.810	87.6001	0.897	0.876
	SVM	80.4869	0.746	0.805	99.9174	0.999	0.999
	ANN	80.7116	0.766	0.807	99.9078	0.999	0.999
	NB	80.4869	0.746	0.805	74.0721	0.794	0.741

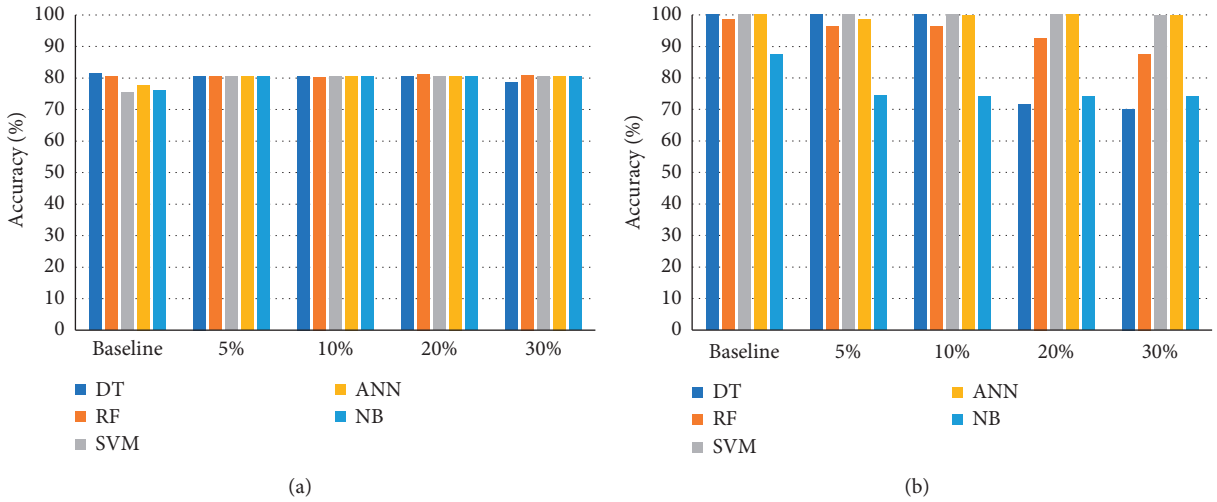

Figure 9: The ML algorithms' accuracy after noise filtering and injection. (a) NSL-KDD. (b) UNSW-NB15.

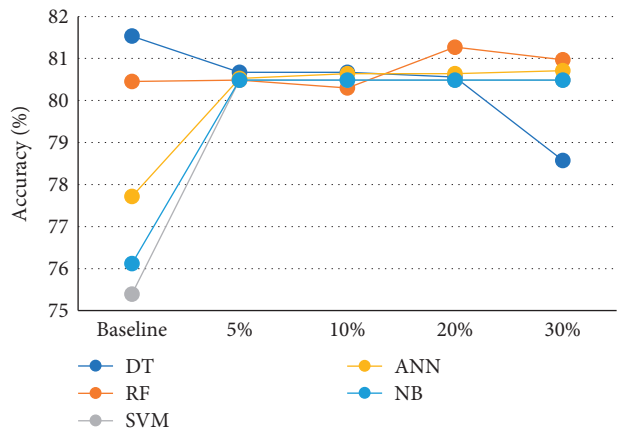

Figure 10: The accuracy of ML algorithms after filtering and noise injection (NSL-KDD).

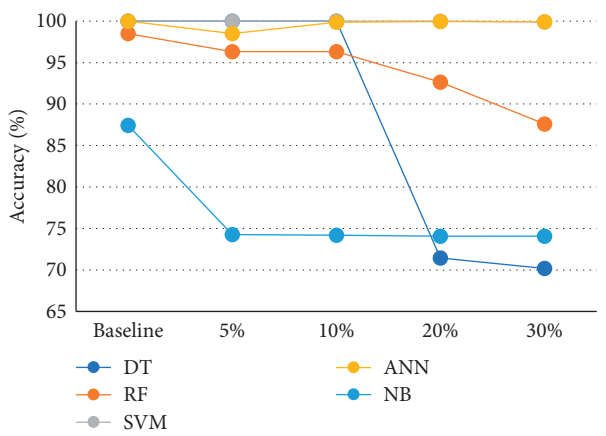

Figure 11: The accuracy of ML algorithms after filtering and noise injection (UNSW-NB15).

Table 16: The results of running ML algorithms on an injected and then filtered intrusion dataset.

Noise (%)	ML algorithm	NSL-KDD			UNSW-NB15		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline	DT (J48)	81.5339	0.858	0.815	100	1.000	1.000
	RF	80.4516	0.852	0.805	98.4903	0.985	0.985
	SVM	75.3948	0.802	0.754	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	NB	76.1178	0.809	0.761	87.4435	0.884	0.874
5% noise injected then filtering	DT (J48)	77.9026	0.828	0.779	100	1.000	1.000
	RF	77.8277	0.749	0.778	97.051	0.973	0.971
	SVM	77.6779	0.715	0.777	100	1.000	1.000
	ANN	77.8652	0.828	0.779	100	1.000	1.000
	NB	77.6779	0.715	0.777	74.2945	0.795	0.743
10% noise injected then filtering	DT (J48)	73.97	0.742	0.740	100	1.000	1.000
	RF	73.8577	0.682	0.739	95.0076	0.957	0.950
	SVM	73.8202	0.676	0.738	100	1.000	1.000
	ANN	74.0824	0.779	0.741	100	1.000	1.000
	NB	73.8202	0.676	0.738	74.2055	0.795	0.742
20% noise injected then filtering	DT (J48)	75.2434	0.762	0.752	73.392	0.849	0.734
	RF	66.7416	0.589	0.667	91.5565	0.929	0.916
	SVM	67.7154	0.620	0.677	100	1.000	1.000
	ANN	67.6404	0.539	0.676	100	1.000	1.000
	NB	67.7154	0.620	0.677	74.0753	0.794	0.741
30% noise injected then filtering	DT (J48)	66.6292	0.676	0.666	70.2269	0.788	0.702
	RF	61.985	0.595	0.620	88.6869	0.900	0.887
	SVM	61.7228	0.573	0.617	99.9873	1.000	1.000
	ANN	61.7978	0.621	0.618	99.8062	0.998	0.998
	NB	61.7228	0.573	0.617	74.088	0.794	0.741

with the 20% noise injection when compared with the 10% noise injection.

(ii) The classification accuracy of the RF algorithm continuously decreased with the increase of the noise levels.

(iii) The classification accuracies of both SVM and NB algorithms were identical in all different noise levels. This is also identical to the previous set of experiments, which means that the order of noise filtering and injection does not change the impact of the classification accuracy on these ML algorithms.

(iv) ANN algorithm achieved a slight increase in the classification accuracy with the 5% noise injection (+0.1505%) comparing to the baseline. In contrast, there was a continuous degradation in the classification accuracy with the increase in the noise level.

For the UNSW-NB15 dataset, the accuracy of ML algorithms is similar to the previous set of experiments. This means that the order of conducting noise filtering and injection does not remarkably affect the classification accuracy. In particular,

(i) DT algorithm maintained the same accuracy result with the 5% and the 10% noise levels. A large degradation in the classification accuracy has occurred with the 20% and the 30% noise levels (−26.608% and −29.7731%, respectively).

(ii) In the RF algorithm, the classification accuracy decreased with the increase in the noise levels.

(iii) Both the SVM and the ANN maintained the same accuracy results across 5%, 10%, and 20%. A small decrease in accuracy occurred with the 30% noise injection. SVM accuracy degraded −0.0127% and ANN accuracy degraded by −0.1938%.

(iv) NB algorithm achieved similar accuracy results across different levels of noise.

Figure 13 shows the impact of injecting and then filtering noise in the NLS-KDD dataset and the corresponding accuracy of the ML algorithms. The performance of all algorithms degraded with DT showing the highest accuracy with the maximum noise level. Figure 14 shows the impact of filtering and then injecting noise in the UNSW-NB15 dataset and the corresponding accuracy of the ML algorithms. DT performance shows the greatest drop among all ML algorithms.

4.6. Ensemble of Classifiers Experiment. In this experiment, two ensemble learning classifiers are used, which are bagging and boosting. In this experiment, each ML algorithm will be used as a base classifier with bagging and boosting. The detailed results of this set of experiments are illustrated in Table 17. Figures 15(a) and 15(b) show the results of ensemble of classifiers learning applied to both NSL-KDD dataset and UNSW-NB15 dataset, respectively.

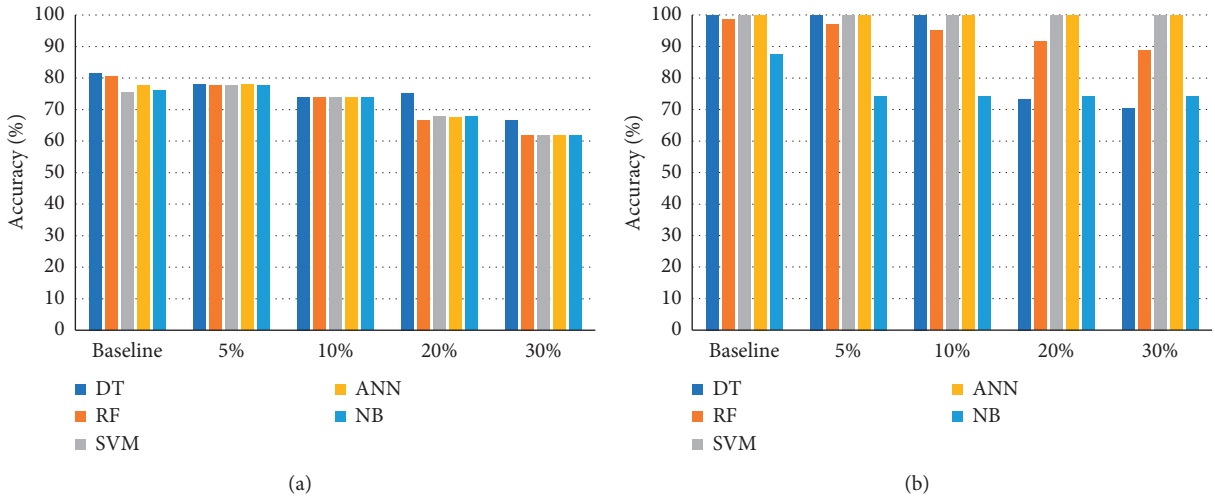

Figure 12: The state of the accuracy with noise filtering then injection. (a) NSL-KDD. (b) UNSW-NB15.

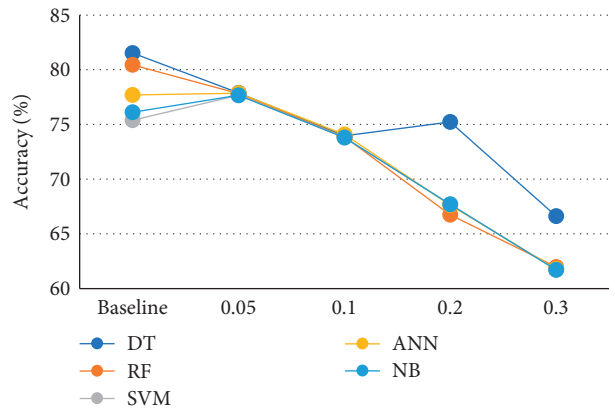

Figure 13: The accuracy of ML algorithms after injecting then filtering the noisy instances in the NSL-KDD dataset.

With the NSL-KDD dataset, it can be noticed that none of the ensemble methods caused a remarkable impact. In fact, the accuracy results indicate that the ML algorithms are slightly affected by the ensemble learning methods. In particular,

(i) The baseline accuracy of the DT algorithm was 81.5339% and it achieved a small increase in accuracy with bagging (83.7252%) which represents +2.1913% increase. With boosting, the achieved classification accuracy (77.8522%) represents a decrease by −3.6817%.

(ii) The classification accuracy of the RF algorithm was negatively affected by both ensemble learning methods. In bagging, it achieved 80.1499% classification accuracy, which is a slight decrease of −0.3017%. The classification accuracy with boosting was (79.4092%), which is also a decrease by −1.0424%.

(iii) Ensembled SVM achieved lower accuracy with bagging and higher accuracy with boosting. SVM classification accuracy with bagging is 75.0311% which represents a decrease by −0.3637%. Boosted SVM achieved an accuracy of 75.6343% which is slightly greater than the baseline by +0.2395%.

(iv) Bagged ANN algorithm achieved an accuracy of 76.0956% which is slightly lower than the baseline by −1.6836%. However, boosting yielded the same accuracy of the baseline, which is 77.7147%.

(v) Similar to the DT algorithm, the NB algorithm generated a greater accuracy value with bagging and a lower accuracy value with boosting. With bagging, it achieved 76.2952%, which is slightly greater than the baseline by +0.1774%, and with boosting, it achieved a lower classification accuracy of 73.7447%, which is less than the baseline by −2.3731%.

In the UNSW-NB15 dataset, ensemble methods did not remarkably improve the classification accuracy of the ML algorithms, except for the RF and the NB algorithms. The observations of the classification accuracy are as follows:

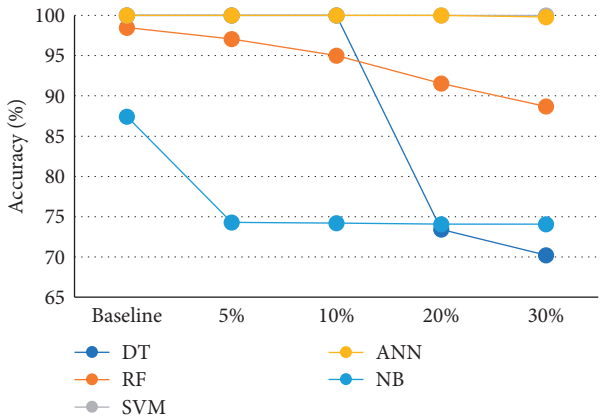

Figure 14: The accuracy of ML algorithms after noise injection then filtering on the UNSW-NB15 dataset.

Table 17: Using ensemble learning (bagging and boosting).

Ensemble learning	ML algorithm	NSL-KDD			UNSW-NB15		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline	DT (J48)	81.5339	0.858	0.815	100	1.000	1.000
	RF	80.4516	0.852	0.805	98.4903	0.985	0.985
	SVM	75.3948	0.802	0.754	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	NB	76.1178	0.809	0.761	87.4435	0.884	0.874
Bagging	DT (J48)	83.7252	0.870	0.837	100	1.000	1.000
	RF	80.1499	0.850	0.801	99.9976	1.000	1.000
	SVM	75.0311	0.800	0.750	100	1.000	1.000
	ANN	76.0956	0.808	0.761	100	1.000	1.000
	NB	76.2952	0.810	0.763	87.2929	0.882	0.873
Boosting	DT (J48)	77.8522	0.838	0.779	100	1.000	1.000
	RF	79.4092	0.846	0.794	90.5577	0.922	0.906
	SVM	75.6343	0.803	0.756	100	1.000	1.000
	ANN	77.7147	0.817	0.777	100	1.000	1.000
	NB	73.7447	0.799	0.737	99.949	0.999	0.999

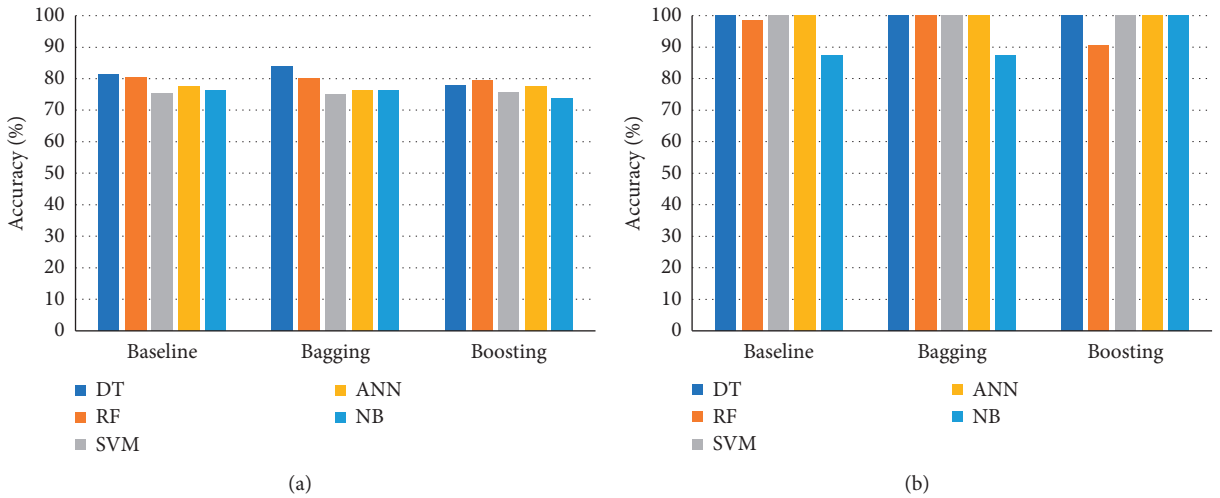

Figure 15: The accuracy results after applying the ensemble of classifiers learning method. (a) NSL-KDD. (b) UNSW-NB15.

(i) DT (J48), SVM, and ANN are not affected by bagging and boosting, and the accuracy after applying bagging and boosting remained as the baseline accuracy, which is 100%.

(ii) With the RF algorithm, there is a slight increase in accuracy with bagging (+1.5073%). However, with boosting, the classification accuracy was decreased by −7.9326%.

(iii) With the NB algorithm, there is a slight decrease in accuracy with bagging (−0.1506%). However, there is a large increase in the classification accuracy with boosting (+12.5055%). The accuracy value increased from 87.4435% to 99.949%. The factors that caused the sudden increase of the classification accuracy of the NB algorithm with boosting need to be further investigated. This might pave the way for developing NB-based IDSs for conducting near real-time detection due to the high speed of the NB algorithm [6].

5. Conclusion and Future Work

Optimizing IDS capabilities is becoming an essential requirement due to the continuous increase of cyber threats and attacks. ML-based IDSs represent one of the emerging paradigms that can be used to confront misuse and anomaly attacks. However, the classification accuracy of the ML-based IDSs is prone to several factors. Determining these factors helps to build better ML-based IDSs.

The results from the empirical experiments of this paper illustrate that the classification accuracy of the ML-based IDSs can be influenced by certain factors. These factors are the method of using the dataset for training and testing, the removal of outliers and extreme value instances, the injection of mislabelled instances, and the use of ensemble learning techniques. These factors have a diverse impact on the classification accuracy. In certain cases, these factors cause a negative impact on the classification accuracy such as the effect of noise on the accuracy of the RF algorithm. On the other hand, some of these factors caused a massive improvement of the classification accuracy such as the NB algorithm with boosting when applied on the UNSW-NB15 dataset, and in other situations, these factors did not affect the accuracy of the ML-based IDSs.

This work can be expanded in several ways. First, algorithms for noise filtering and instance reduction such as the Decremental Reduction Optimization Procedures DROP3 and DROP5 can be used, rather than using the built-in noise filtering functions of WEKA. Other datasets such as ADFA, CIC-IDS 2017, or CIC-DDoS 2019 can also be tested. The work can also be expanded by employing the fine-tuned NB algorithm [40] which improved NB performance by finding a better estimation of NB's probability terms, which makes the algorithm less stable and more suitable for building a bag of ensembles of classifiers [41]. Investigating the efficacy of these methods for the evaluation of the effect of noise using intrusion detection datasets is an interesting future research. Feature selection can also be included in future work. The impact of including or removing different features can be studied empirically, which might represent an important factor that can optimize the classification accuracy of the ML-based IDSs.

Data Availability

The two intrusion-related datasets that are used in this paper are publicly available for download. The NSL-KDD dataset can be accessed via https://www.unb.ca/cic/datasets/nsl.html and the training and the testing portions of the UNSW-NB15 dataset can be downloaded from https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The APC was funded by the Deanship of Scientific Research, Saudi Electronic University.

References

[1] S. Elsayed, R. Sarker, and J. Slay, "Evaluating the performance of a differential evolution algorithm in anomaly detection," in *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2490–2497, IEEE, Sendai, Japan, May 2015.

[2] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing*, vol. 72, no. 9, pp. 3489–3510, 2016.

[3] D. E. Denning, "An intrusion-detection model," in *Proceedings of the IEEE Computer Society Symposium on Research Security and Privacy*, pp. 118–131, Oakland, CA, USA, April 1986.

[4] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, 1987.

[5] D. A. Kumar and S. Venugopalan, "Intrusion detection systems: a review," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 8, pp. 356–370, 2017.

[6] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 686–728, 2018.

[7] J. Gupta and J. Singh, "Detecting anomaly based network intrusion using feature extraction and classification techniques," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, pp. 1453–1457, 2017.

[8] N. Moustafa, J. Hu, and J. Slay, "A holistic review of network anomaly detection systems: a comprehensive survey," *Journal of Network and Computer Applications*, vol. 128, pp. 33–55, 2019.

[9] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[10] Y. Xin, L. Kong, Z. Liu et al., "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[11] P. Louridas and C. Ebert, "Machine learning," *IEEE Software*, vol. 33, no. 5, pp. 110–115, 2016.

[12] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. M. Leung, "A survey on security threats and defensive techniques of machine learning: a data driven view," *IEEE Access*, vol. 6, pp. 12103–12117, 2018.

[13] A. Géron, *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, N. Tache, Ed., O'Reilly Media, Inc., Newton, MA, USA, 1st edition, 2017.

[14] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38367–38384, 2018.

[15] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[16] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2005.

[17] S. Smiti and M. Soui, "Bankruptcy prediction using deep learning approach based on borderline SMOTE," *Information Systems Frontiers*, vol. 22, no. 5, pp. 1067–1083, 2020.

[18] M. Al-Akhras, H. Zedan, R. John, and I. ALMomani, "Non-intrusive speech quality prediction in VoIP networks using a neural network approach," *Neurocomputing*, vol. 72, no. 10–12, pp. 2595–2608, 2009.

[19] K. El Hindi and M. Al-Akhras, "Smoothing decision boundaries to avoid overfitting in neural network training," *Neural Network World*, vol. 21, no. 4, pp. 311–326, 2011.

[20] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, Burlington, MA, USA, 3rd edition, 2011.

[21] H. Bhavsar and A. Ganatra, "A comparative study of training algorithms for supervised machine learning," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, pp. 74–81, 2012.

[22] D. Fletcher, *Class InterquartileRange*, http://weka.sourceforge.net/doc.dev/weka/filters/unsupervised/attribute/InterquartileRange.html.

[23] J. Hussain and S. Lalmuanawma, "Feature analysis, evaluation and comparisons of classification algorithms based on noisy intrusion dataset," *Procedia Computer Science*, vol. 92, pp. 188–198, 2016.

[24] A. J. Al-Mahasneh, S. G. Anavatti, and M. A. Garratt, "Evolving general regression neural networks for learning from noisy datasets," in *Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence, SSCI 2019*, pp. 1473–1478, IEEE, Canberra, Australia, December 2019.

[25] E. Leon, O. Nasraoui, and J. Gomez, "Anomaly detection based on unsupervised niche clustering with application to network intrusion detection," in *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, pp. 502–508, IEEE, Seattle, WA, USA, June 2004.

[26] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Raheem, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018.

[27] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[28] P. Tao, Z. Sun, and Z. Sun, "An improved intrusion detection algorithm based on ga and SVM," *IEEE Access*, vol. 6, pp. 13624–13631, 2018.

[29] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[30] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, "Intrusion detection model using machine learning algorithm on big data environment," *Journal of Big Data*, vol. 5, no. 1, p. 34, 2018.

[31] H. Wang, J. Gu, and S. Wang, "An effective intrusion detection framework based on SVM with feature augmentation," *Knowledge-Based Systems*, vol. 136, pp. 130–139, 2017.

[32] E. Kabir, J. Hu, H. Wang, and G. Zhuo, "A novel statistical technique for intrusion detection systems," *Future Generation Computer Systems*, vol. 79, pp. 303–318, 2018.

[33] A. Pektaş and T. Acarman, "A deep learning method to detect network intrusion through flow-based features," *International Journal of Network Management*, vol. 29, no. 4, pp. 1–19, Article ID e2050, 2018.

[34] E. A. Shams and A. Rizaner, "A novel support vector machine based intrusion detection system for mobile ad hoc networks," *Wireless Networks*, vol. 24, no. 5, pp. 1821–1829, 2018.

[35] S. A. R. Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to snort system," *Future Generation Computer Systems*, vol. 80, pp. 157–170, 2018.

[36] R. S. M. Carrasco and M.-A. Sicilia, "Unsupervised intrusion detection through skip-gram models of network behavior," *Computers & Security*, vol. 78, pp. 187–197, 2018.

[37] A. M. Al Tobi and I. Duncan, "KDD 1999 generation faults: a review and analysis," *Journal of Cyber Security Technology*, vol. 2, no. 3-4, pp. 164–200, 2018.

[38] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–6, IEEE, Ottawa, Canada, July 2009.

[39] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[40] K. M. Hindi, "Fine tuning the Naïve Bayesian learning algorithm," *AI Communications*, vol. 27, no. 2, pp. 133–141, 2014.

[41] K. El Hindi, H. AlSalman, S. Qasem, and S. Al Ahmadi, "Building an ensemble of fine-tuned naive Bayesian classifiers for text classification," *Entropy*, vol. 20, no. 11, p. 857, 2018.