Article

Genetic Algorithm Based Feature Selection Technique for Optimal Intrusion Detection

Sydney Mambwe Kasongo [1,2]

[1] Stellenbosch University; Department of Industrial Engineering, 145 Banghoek Rd, Stellenbosch Central, Stellenbosch, 7600
[2] Stellenbosch University; School of Data Science and Computational Thinking, 44 Banghoek Rd, Stellenbosch Central, Stellenbosch, 7599
* Correspondence: sydneyk@sun.ac.za

Abstract: In recent years, several industries have registered an impressive improvement in technological advances such as Internet of Things (IoT), e-commerce, vehicular networks, etc. These advances have sparked an increase in the volume of information that gets transmitted from different nodes of a computer network (CN). As a result, it is crucial to safeguard CNs against security threats and intrusions that can compromise the integrity of those systems. In this paper, we propose a machine mearning (ML) intrusion detection system (IDS) in conjunction with the Genetic Algorithm (GA) for feature selection. To assess the effectiveness of the proposed framework, we use the NSL-KDD dataset. Furthermore, we consider the following ML methods in the modelling process: decision tree (DT), support vector machine (SVM), random forest (RF), extra-trees (ET), extreme gradient boosting (XGB), and naïve Bayes (NB). The results demonstrated that using the GA algorithm has a positive impact on the performance of the selected classifiers. Moreover, the results obtained by the proposed ML methods were superior to existing methodologies.

Keywords: Intrusion Detection, Genetic Algorithm; Machine Learning

1. Introduction

The recent advances of technologies such as wireless networks, the Internet of Things (IoT), Industrial Internet of things (IIoT) have sparked an increase in various threats that can compromise the security, integrity, and reliability of those systems [1]. There exist several mechanisms that organizations can use to guard themselves against systems intrusions including, firewalls (considered as the first line of defense), anti-malware, and anti-various software systems, intrusions detection and prevention systems, etc. [2]. Among all the safety measures, an intrusion detection system (IDS) is considered as one of the most crucial mechanisms capable of defending computer networks against potential intrusions [3].

At the top level of classification, IDSs are categorized as network-based and host-based IDSs [4]. These systems can further be classified into knowledge-based IDS, anomaly-based IDS, and hybrid-based IDS [5]. Network-based IDS is an IDS that is deployed throughout a computer network's system. A host-based IDS is deployed on a node within a computer network. A knowledge-based IDS, sometimes called a signature-based IDS, uses existing patterns of attacks to detect current intrusions. One major disadvantage of a knowledge-based IDS is that it has to be constantly updated with new attack signatures. Anomaly-based IDSs are more dynamic in comparison to knowledge-based IDSs because they don't depend on existing attack patterns to flag intrusions. A hybrid-based IDS combines an anomaly-based and signature-based IDS [6]. In his research, we present an anomaly-based intrusion detection using Machine Learning (ML). ML is a branch of Artificial Intelligence (AI) that allows software applications or programs to perform predictions tasks without an explicit programming instruction [7].

 doi:10.20944/preprints202106.0710.v1

In this work, we consider the following ML methods: decision tree (DT) [8], random forest (RF) [9], extra-trees (ET) [10], extreme gradient boosting (XGB) [11], support vector machine (SVM) [12], and baive Bayes (NB) [13]. The SVM and NB are used as the baseline models. These models serve as a point of departure in the modelling process. Moreover, to assess the performance of the models presented in this paper, we use the NSL-KDD dataset [14]. This dataset is often employed as a benchmark dataset in developing anomaly-based IDS.

Furthermore, IDS benchmark datasets generally have a high feature dimension that can have a negative impact during the modelling procedure of a given classifier. Thus, it is critical to implement a Feature Selection (FS) or Feature Extraction (FE) method. FS is a crucial process in developing ML based IDS systems because it makes sure that only the most optimal (best) features are selected for the modelling procedure [15]. There exist two categories of FS methods, namely filter-based FS and wrapper-based FS. A filter-based FS select attributes based on the intrinsic nature of the data [16]. On the other hand, a wrapper-based FS method uses a model to select the most optimal attributes subset. In this instance, the best feature subset is selected using the performance of a model [17]. In this research, we propose a wrapper-based FS method that is based on the Genetic Algorithm (GA) [18]. Moreover, the fitness function that is implemented in the GA uses the Extra-Trees (ET) classifier [19]. In the experiments, the GA generated 7 feature vectors for the binary and multiclass classification procedures. These attributes vectors were then used in the modelling process. We evaluated the performance of the models using the selected features and we compared the results with existing ML methods. The results demonstrated that using GA for FS leads to an increase in performance for the classifiers considered in this research.

The remainder of the paper is organized as follows. Section 2 presents an account of related work. Section 3 provides an overview of the NSL-KDD dataset. Section 4 presents a background on the ML algorithms that are used in this research. In Section 5, the proposed intrusion detection framework is presented. Section 6 provides the experimental setup and a discussion of the results. Finally, Section 7 concludes this paper.

2. Related Work

In [20], the authors implemented a Deep Learning (DL) method for intrusion detection and prevention for software-defined networking (SDN) systems. The researcher utilized the NSL-KDD dataset to assess the effectiveness of their proposed architecture. The classifier that was used to model the IDS is the deep neural network (DNN) algorithm. The DNN used in this research is of type *feed-forward* whereby the information flows in one direction. The experiments were carried using the binary classification configuration and the accuracy was the main performance metric that was considered. The results showed that the DNN achieved an accuracy of 75.75%. Although these results are significant, this research did not implement a feature selection method that would potentially increase the performance of the proposed DNN.

Hosseini [21] proposed a two-step IDS based on ML. The first step is the feature selection procedure whereby the author used the logistic regression (LR) algorithm along with the GA method. In the second phase, the authors utilized the artificial neural network (ANN) method for classification. The ANN was trained using several evolutionary-based algorithms including the particle swarm optimization (PSO) approach. The NSL-KDD dataset was used to validate the performance of the presented frameworks. The accuracy and the training time were some of the performance metrics that were considered. Moreover, the authors implemented the binary classification scheme. The outcomes demonstrated that the PSO-ANN achieved an accuracy of 88.90% and it was trained in 74 seconds. In contrast, the GA-ANN obtained an accuracy of 83.11% and it was trained in 134 seconds.

 doi:10.20944/preprints202106.0710.v1

Su et al. [22] implemented a bidirectional long short-term memory (BLSTM) method in conjunction with an attention mechanism (BAT-MC) for feature extraction using the NSL-KDD dataset. The attention algorithm was used to capture the most important attributes required for an optimal classification procedure. The experimental processes were carried out for the BAT and the BAT-MC. The results demonstrated that the BAT and BAT-MC obtained accuracy scores of 82.56% and 84.25 %, respectively. The authors claimed that the BAT and the BAT-MC are superior to other methods; however, in the comparison analysis, they did not contrast their approach with existing evolutionary-based algorithms used for feature selection.

In [23], the researchers proposed an ensemble ML algorithm for intrusion detection. In order to build this model, the authors used multiple DTs to create a MuliTree architecture. The MuliTree changes in size and structure when the size of the training data varies. Furthermore, several baseline models were implemented to validate the performance of the MuliTree method. The NSL-KDD dataset was used as a testbed for the proposed IDS. In the experiments, the accuracy that was obtained on test data was used as the main performance metric. The experimental results demonstrated that the MuliTree approach achieved an accuracy score of 85.2% for the binary classification task. Although these results were superior in comparison to the base line models, the authors conceded that more research needed to be conducted to implement a feature selection or extraction method that could increase the performance of the MultiTree method.

Zhang et al. [24] presented a deep learning-based IDS using the NSL-KDD dataset. In this research, the author utilized an autoencoder (AE) to extract the most important attributes that were used by the DNN for classification. To gauge the performance of the AE-DNN, the authors considered the accuracy, precision, recall, and F1-score. The experimental outcomes revealed that the AE-DNN obtained a classification accuracy of 79.74%, a precision of 82.22%, a recall of 79.74%, and an F1-score of 76.47%. In future work, the researchers intended to improve the detection accuracy by tuning the DNN or by exploring the use of other classifiers.

In [25], an IDS approach using an adaptive synthetic sampling (ADASYN) technique coupled with a convolutional neural network (CNN) was proposed. Firstly, The ADASYN methodology was used to reduce the sensitivity of the algorithm against any form of class imbalance. Secondly, The CNN algorithm utilized in this research is derived from the split convolution module (SPCCNN). The SPCCNN was used to reduce the impact of unwanted information during the training process. Lastly, the AS-CNN algorithm in conjunction with the ADASYN and SPCNN is used to conduct the modeling process. The NSL-KDD dataset was used to evaluate the performance of the proposed framework. Additionally, the authors also implemented the following baseline models: recurrent neural network (RNN) and simple convolutional neural network (CNN). The result revealed that the RNN obtained a detection accuracy of 69.73%. The CNN achieved an accuracy score of 68.66% and the AS-CNN achieved a remarkable accuracy of 80%.

Tama et al. [26] presented the TSE-IDS, a two-step model for intrusion detection. In the first phase, the TSE-IDS used several feature selection algorithms including particle swarm optimization (PS), GA, and ant colony algorithm (ACO). The fitness criterion used to select a feature set is the performance that was obtained on the pruning tree (REPT) algorithm. In the second phase, the TSE-IDS uses an ensemble of classifiers that includes the bagging trees and the rotation forest. The experiments were carried using the NSL-KDD and UNSW-NB15 datasets. Moreover, the binary classification scheme was considered. To assess the performance of their method, the authors considered some of the following metrics: accuracy, precision, and sensitivity. The experimental results demonstrated that TSE-IDS achieved an accuracy of 85.797%, the precision of 88.00%, and sensitivity of 86.80% for the NSL-KDD dataset. In the case of the UNSW-NB15, the TSE-IDS obtained the following results: an accuracy of 91.27%, a sensitivity of 91.30%,

 doi:10.20944/preprints202106.0710.v1

and a precision of 91.60%. In future work, the authors planned to test the TSE-IDS for the multiple classes modeling process.

In [27], the authors proposed a hybrid attribute selection approach based on the binary gravitational search algorithm (BGSA). The BGSA is coupled with the mutual information (MI) method to increase the performance of the standard BGSA. In this research, the BGSA is used as a wrapper-based algorithm and the MI is used as a filter algorithm. The BGSA-MI was evaluated on the NSL-KDD intrusion detection dataset. The support vector machine (SVM) classifier was in the fitness function of the BGSA-MI and the accuracy was the main performance metric. Moreover, the authors also implemented the ReliefF and the Chi-square methods as the baseline algorithms. The experimental results showed that the BGSA-MI obtained an accuracy score of 88.36%, the ReliefF obtained an accuracy of 84.60%, and the Chi-square approach achieved an accuracy 84.68%.

Almasoudy [28] implemented a wrapper-based attribute selection method using the differential evolution (DE) algorithm for intrusion detection. In this research, the extreme learning machine (ELM) classifier was used to assess the selected feature sets. The NSL-KDD dataset was used as the testbed for the DE-ELM. Additionally, the researchers considered the two-class and the five-class classification configurations. The experimental results showed that the DE-ELM obtained an accuracy of 80.15 % and 87.53 % for the binary and multiclass classification configuration, respectively. In future work, the authors planned to implement the DE-ELM on live traffic and to increase its accuracy on minority classes such as the U2R.

Sarumi et al. [29] conducted a comparison analysis between a rule-based algorithm called Apriori and the SVM ML technique. For each of those methods, the authors implemented a filter-based and a wrapper-based feature selection technique. The performance of the Apriori and the SVM were evaluated on the UNSW-NB15 and the NSL-KDD datasets. Focusing on the NSL-KDD dataset, the experimental results demonstrated that the filter-SVM achieved an accuracy of 77.17%, recall 95.38%, and precision of 66.34%. The filter-Apriori obtained an accuracy of 67.00%, recall of 57.89%, and precision of 85.77%. In contrast, the wrapper-SVM obtained an accuracy of 79.65%, recall of 98.02%, and precision of 68.41. The wrapper-Apriori attained an accuracy of 68%, recall of 58.81%, and precision of 85.79%.

3. NSL-KDD dataset

In this research, the NSL-KDD dataset [14] is utilized to assess the performance of the proposed IDS framework. The NSL-KDD dataset is widely accepted in the literature and it has been the basis for several kinds of research on ML-based IDS. The NSL-KDD dataset includes the following categories of network traffic traces: *normal*, *R2L*, *U2R*, *DoS* and *Probe*. Moreover, the NSL-KDD comprises two data subsets: the NSL-KDD-Train and NSL-KDDTest+. In this study, we split the NSL-KDD into two partitions, namely, the NSL-KDD-Val and the NSL-KDD-Train+. The NSL-KDD-Val represents 25% of the entire NSL-KDD-Train and the NSL-KDD-Train+ constitutes 75% of the original training data subset. The NSL-KDD-Train+ is used to train various ML models, the NSL-KDD-Val is utilized to validate the trained models and NSL-KDD-Test+ is employed for the testing procedure.The validation phase guarantees that the ML algorithms used in this research are not prone to overfitting [30]. Table 1 provides the details about all the attributes of the NSL-KDD dataset that includes 3 categorical attributes. The rest of the features are of numerical format. Table 2 outlines the values distribution per attack categories for each data subset.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Table 1. NSL-KDD attributes description.

No.	Name	Format	No.	Name	Format
f1	duration	numeric	f22	is_guest_login	numeric
f2	protocol_type	categorical	f23	count	numeric
f3	service	categorical	f24	srv_count	numeric
f4	flag	categorical	f25	serror_rate	numeric
f5	src_bytes	numeric	f26	srv_serror_rate	numeric
f6	dst_bytes	numeric	f27	rerror_rate	numeric
f7	land	numeric	f28	srv_rerror_rate	numeric
f8	wrong_fragment	numeric	f29	same_srv_rate	numeric
f9	urgent	numeric	f30	diff_srv_rate	numeric
f10	hot	numeric	f31	srv_diff_host_rate	numeric
f11	num_failed_logins	numeric	f32	dst_host_count	numeric
f12	logged_in	numeric	f33	dst_host_srv_count	numeric
f13	num_compromised	numeric	f34	dst_host_same_srv_rate	numeric
f14	root_shell	numeric	f35	dst_host_diff_srv_rate	numeric
f15	su_attempted	numeric	f36	dst_host_same_src_port_rate	numeric
f16	num_root	numeric	f37	dst_host_srv_diff_host_rate	numeric
f17	num_file_creations	numeric	f38	dst_host_serror_rate	numeric
f18	num_shells	numeric	f39	dst_host_srv_serror_rate	numeric
f19	num_access_files	numeric	f40	dst_host_rerror_rate	numeric
f20	num_outbound_cmds	numeric	f41	dst_host_srv_rerror_rate	numeric
f21	is_host_login	numeric			

Table 2. NSL-KDD subsets values distribution.

Dataset	Normal	DoS	Probe	R2L	U2R	Total
KDDTrain	67343	45927	11656	995	52	125973
KDDTrain+	50494	34478	8717	749	42	94480
KDDVal	16849	11449	2939	246	10	31493
KDDTest+ Full	9711	7458	2754	2421	200	22544

4. Background on Machine Learning Algorithms

4.1. Decision Tree

Decision Tree (DT) a supervised ML method that is employed to model regression and classification problems. DT is amongst the most popular ML algorithms because of its inherent simplicity. Some of the advantages of DTs include the fact that DT can be visualized, the prediction process of the DTs is relatively simple to interpret, DTs are capable of solving multi-output tasks and DTs do not require extensive data cleaning [31]. As depicted in Figure 1, a DT contains the following types of nodes [32]:

- **Root Node**: This is the initial node of the tree. The starting point of decision making process.
- **Intermediate or splitting nodes**: these nodes generate more nodes when the final prediction is not reached.
- **Leaf or Terminal nodes**: these nodes do not split and they represent the final predictions that are made by the DT.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

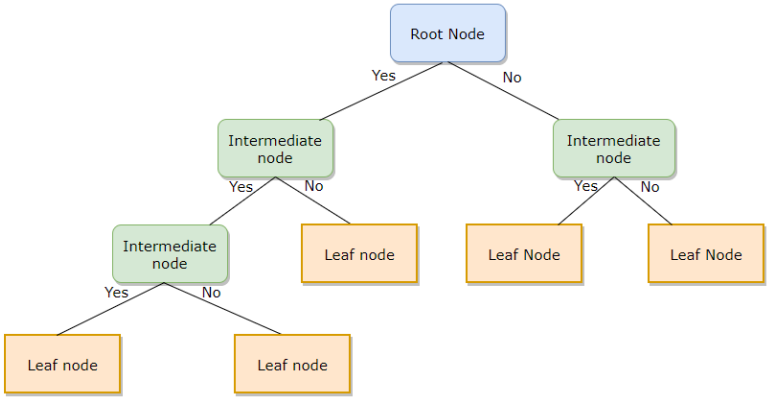


Figure 1. This figure shows the structure of a sample DT with the following types of nodes: root node, intermediate nodes and leaf nodes.

4.2. Ensemble tree-based Classifiers

In this study, the following ensemble tree-based classifiers are implemented: Random Forest (RF), Extra-Trees (ET), and Extreme Gradient Boosting (XGBoost). These algorithms use DTs as building blocks.

4.2.1. Random Forest

Random Forest (RF) is a supervised ML approach that uses multiple DTs to compute predictions. In the RF method, each DT computes a prediction, and the class that obtains the highest vote (the class that is predicted by the majority of DTs) is considered as the prediction of the RF model. Moreover, an RF model is trained using a method referred to as bootstrapping and it selects its split points using a greedy algorithm. The bootstrapping technique guarantees the uniqueness of each DT in the RF [33,34].

4.2.2. ExtraTrees

Extra-Trees (ET) or sometimes referred to as Extremely Randomized Trees is also a supervised ML algorithm that uses several DTs in its decision-making process. Furthermore, ET can be used for classification and regression problems. Unlike the RF that develops each DT from a portion of the training set; the ET fits individual DTs over the entire training set. Moreover, the ET method picks the nodes split points randomly [19,35].

4.2.3. Extreme Gradient Boosting

The Extreme Gradient Boosting (XGBoost) ML algorithm is an approach that allows for the tree algorithm to be boosted. Similar to the ET and RF algorithms, the XGBoost is also an ensemble-based ML method. The contrast between the XGBoost and the RF and ET lies in the fact in the XGBoost approach, the DTs are not independent. Each DT is an extension of an already created tree [36,37]. The mathematical formulation of XGBoost is as follows:

Let $G = \{(x_n, y_n) : n = 1...q, x_n \in \mathbb{R}^p, y_n \in \mathbb{R}\}$ represent dataset that contains q with q attributes denoted by y. Let $\hat{y}_n$ be the prediction of the XGBoost classifier expressed as follows:

$$\hat{y}_n = \vartheta(x_n) = \sum_{i=1}^{I} f_i(x_n) \tag{1}$$

where f_i represents a regression tree and $f_i(x_n)$ is the score value assigned to the i-th tree of the n-th record in the dataset. The ultimate aim is the minimization of Equation (2).

$$E(\vartheta) = \sum_{k} l(y_i, \hat{y}_n) + \sum_{m} \Phi(f_i) \tag{2}$$

 doi:10.20944/preprints202106.0710.v1

where l is denotes the loss over the entire dataset (or data subset). E is the loss function. Additionally, Φ acts as penalization parameter in Equation (2) and consequently, it decreases the model's overall complexity. Φ is expressed in Equation (3).

$$\Phi(f_i) = \sigma M + \frac{1}{2} \upsilon \|W\|^2 \tag{3}$$

where *sigma* and $upsilon$ are the regularization parameters that penalizes the number of tree leaves M and their associated weights W. Φ ensures that the XGBoost model doesn't overfit on the data while performing the tree boosting procedure.

4.3. Naive Bayes

Naive Bayes (NB) algorithm is a popular supervised ML method that is based on the *Bayes' Theorem* [38]. The NB approach makes the naive assumption of the independence between each pair of attributes within a dataset. The mathematical formulation of the NB is as follows [39]:

Let $G = (g_1, ..., g_n)$ denote a dataset with n attributes. To predict the label, C_r, a given instance in G, the NB algorithms computes p:

$$p(C_r|G) = \frac{p(G|C_r)p(C_r)}{P(G)} \tag{4}$$

The label, y, of the instance is therefore computed as follows:

$$y = \underset{k \in \{1,...,K\}}{\operatorname{argmax}} \, p(C_k) \prod_{i=1}^{n} p(G_i|C_k) \tag{5}$$

where y denotes the predicted class. In this research, we implemented the Bernoulli NB (BernoulliNB) [39] as expressed in Equation (6).

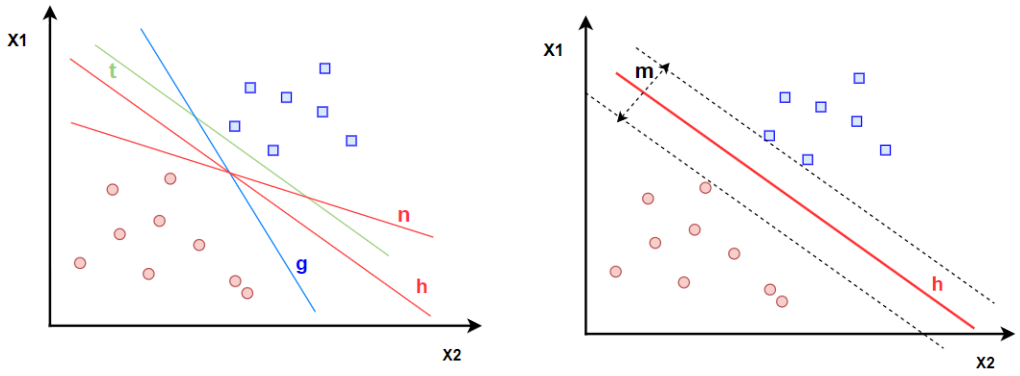
$$P(x_k|y) = P(k|y)x_k + (1 - P(i|y))(1 - x_k) \tag{6}$$

where x denotes a given feature and y is the class.

4.4. Support Vector Machine

Support Vector Machine (SVM) is a supervised ML technique that is widely used in the domain of Big Data and related subjects. The goal of an SVM method is to discover the most optimal hyperplane or hyperplanes that clearly classify data points within a given data space [40]. Lets consider a case whereby an SVM is tasked to classify two sets of data points as shown in Figure 2. In Figure 2a, hyperplanes that successfully splits the data. However, aim for the SVM algorithm is to maximise the value of m in 2b in order to find the optimal hyperplane h. However, in real-world applications, data points are not always separable using a line. In other words, some tasks require a non-linear solution. To overcome this issue, the SVM algorithm uses Kernels [41]. A kernel function converts a low-dimensional input space (input data) into a higher dimensional space in order for it to be separable. Some of the most popular kernels include the *linear kernel, polynomial kernel, the radial basis function (RBF) kernel* [42].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1



(a) Multiple hyperplanes **(b)** Optimal hyperplane

Figure 2. Plot **(a)** shows multiple hyperplanes (*n*,*g*,*t*, *h*) that successfully separated the circles from the squares. Plot **(b) depicts the optimal hyperplane,** *h* **that separates the data points.**.

5. The Proposed Intrusion Detection Framework

5.1. Feature Selection using the Genetic Algorithm

Genetic algorithms (GAs) form part of Evolutionary-based algorithms that were created in a bid to mimic the laws of biological natural genetics and natural selection. Moreover, GAs are heuristic optimization techniques. In other words, they are randomized search methods. Similar to the biological natural selection process GAs randomly pick individuals from an initial set referred to as the *population*. These individuals are considered to be the *parents* that will reproduce a new set of individuals that belong to the new generation [18,43]. The following are the most critical components (steps) in a GA [44,45]:

- GA starts its search process from an Initial *Population* and not a single individual.
- GA should be able to compute the *Fitness* function/measure to assess the performance of a candidate solution.
- GA should conduct a *Selection* process.
- GA should conduct a *Crossover* process.
- GA should perform a *Mutation* procedure.

In this work, the fitness function that was used in the GA was implemented using the ET method as described in Algorithm 1. The full implementation of GA that was utilized in this research is outlined in Algorithm 2. Moreover, the GA was implemented in two phases. In the first phase, the GA was applied to the NSL-KDD dataset to generate the list of features ($V = v_1, v_2, v_3, v_4, v_5, v_6, v_7$) that will be used in the binary classification process. Table 3 provides the details of each feature vector that was selected including the feature vector's length as well as the list of features that were produced. In the second phase, the GA was applied to the NSL-KDD to compute the list of attributes ($G = g_1, g_2, g_3, g_4, g_5, g_6, g_7$) that will be utilized in the multiclass classification process. Table 4 describes each feature vector present in *G*.

Algorithm 1 GA fitness function using the ET approach

Input: *X*, *y*; denotes the input and the label
Output: *Acc*; the Accuracy that is achieved by the ET algorithm.

1. Create the train and test data subsets: *X* and *y* in X_{train}, X_{test}, y_{test}, y_{test}
2. Instantiate the classifier: *et*.
3. Train and fit the model, *et*, using X_{train} and y_{train}
4. Compute predictions $y_{predictions}$ using X_{test}
5. Calculate the *Acc* using $y_{predictions}$ and y_{test}

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Table 3. Selected Features - Binary classification

Feature vector	vector length	List of features
v_1	20	service, flag, src_bytes, land, urgent, hot, logged_in, num_compromised, root_shell, su_attempted, num_root, num_shells, num_access_files, num_outbound_cmds, srv_count, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, dst_host_srv_diff_host_rate
v_2	21	service, src_bytes, dst_bytes, land, wrong_fragment, hot, logged_in,num_compromised, root_shell, num_file_creations, num_shells, num_access_files, is_host_login, is_guest_login, srv_count, serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate
v_3	17	duration, service, src_bytes, dst_bytes, land, num_compromised, num_root, num_outbound_cmds, count, serror_rate, srv_serror_rate,srv_rerror_rate, diff_srv_rate, dst_host_srv_count, dst_host_srv_diff_host_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate
v_4	25	protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, logged_in, num_compromised, root_shell, num_file_creations, num_shells, num_access_files, is_host_login, srv_count, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_diff_srv_rate, dst_host_serror_rate, dst_host_rerror_rate
v_5	20	service, flag, src_bytes, dst_bytes, wrong_fragment, num_failed_logins, num_compromised, num_file_creations, num_access_files, num_outbound_cmds, srv_count, serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, dst_host_count, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_srv_serror_rate, dst_host_srv_rerror_rate
v_6	20	duration, protocol_type, service, src_bytes, dst_bytes, urgent, logged_in, num_root, num_outbound_cmds, is_guest_login, srv_count, serror_rate, srv_serror_rate, rerror_rate, same_srv_rate, diff_srv_rate, dst_host_count, dst_host_diff_srv_rate, dst_host_srv_diff_host_rate, dst_host_srv_rerror_rate
v_7	21	duration, service, src_bytes, land, wrong_fragment, urgent, hot, logged_in, num_compromised, su_attempted, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_guest_login, count, srv_count, rerror_rate, same_srv_rate, srv_diff_host_rate, dst_host_srv_serror_rate

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Table 4. Selected Features - Multiclass classification

Feature vector	vector length	list of features
g_1	22	duration, protocol_type, service, src_bytes, dst_bytes, urgent, num_failed_logins, logged_in, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, srv_serror_rate, srv_rerror_rate, diff_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate
g_2	23	duration, service, flag, src_bytes, dst_bytes, land, hot, num_failed_logins, root_shell, su_attempted, num_root, num_file_creations, num_access_files, num_outbound_cmds, is_guest_login, count, serror_rate, srv_serror_rate, rerror_rate, diff_srv_rate, dst_host_same_srv_rate, dst_host_srv_serror_rate, dst_host_rerror_rate
g_3	21	duration, protocol_type, service, src_bytes, urgent, logged_in, root_shell, num_root, num_access_files, is_guest_login, count, srv_serror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_srv_serror_rate, dst_host_rerror_rate
g_4	18	service, src_bytes, dst_bytes, wrong_fragment, urgent, hot, num_compromised, su_attempted, num_root, num_file_creations, is_host_login, count, srv_count, rerror_rate, dst_host_srv_count, dst_host_diff_srv_rate, dst_host_serror_rate, dst_host_srv_serror_rate
g_5	21	duration, protocol_type, service, flag, src_bytes, dst_bytes, num_compromised, root_shell, su_attempted, num_root, num_shells, num_access_files, count, srv_count, rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate
g_6	20	duration, protocol_type, service, src_bytes, dst_bytes, urgent, logged_in, num_root, num_outbound_cmds, is_guest_login, srv_count, serror_rate, srv_serror_rate, rerror_rate, same_srv_rate, diff_srv_rate, dst_host_count, dst_host_diff_srv_rate, dst_host_srv_diff_host_rate, dst_host_srv_rerror_rate
g_7	21	duration, service, src_bytes, land, wrong_fragment, urgent, hot, logged_in, num_compromised, su_attempted, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_guest_login, count, srv_count, rerror_rate, same_srv_rate, srv_diff_host_rate, dst_host_srv_serror_rate

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Algorithm 2 Implementation of the GA on NSL-KDD Dataset

Require: S, the NSL-KDD dataframe
Require: G, a list (or array) containing the feature names of S.
Require: T, denotes the label
Require: F, an empty list that will temporarily store the feature subset
Require: ni, the total number of iterations (maximum)
START
1. Initialize the population Q, using G.
2. Compute the fitness measure using the ET
3. Compute the fitness using S, G, T and Q
4. calculate the optimal fitness score, b
5. Update F
for k in **range**(ni)
    6. Implement crossover
    7. compute mutations
    8. calculate the fitness
    9. generate the optimal fitness score, b
    10. Update F
end for
11. The GA has converged $\rightarrow F$ and b
STOP

5.2. IDS Architecture

The architecture of the proposed IDS is depicted in Figure 3. At the top level, there 4 major components namely, data preparation, feature selection, modelling and evaluation. In the data preparation or pre-processing phase, the numerical inputs of the NSL-KDD dataset are scaled (normalized) using the Min-Max scaling [46,47] methodology (Equation 7). This process ensures that the input data is kept within a range of $[0, 1]$.

$$v_{scaled} = \frac{v_n - min(v_n)}{max(v_n) - min(v_n)} \tag{7}$$

where v is the original value of a given attribute and v_{scaled} is the scaled value. Furthermore, the categorical attributes of the NSL-KDD (*service*, *protocol_type* and *flag*) are transformed (encoded) into numerical features using the *LabelEncoder* functionality that is found in Scikit-learn [48].

The second phase of the proposed IDS architecture involves the implementation of the GA for feature selection (as outlined in Section 5.1). This phase generates sets of feature vectors (V for the binary classification scheme and G for the multiclass classification task). In the modeling phase, the DT, RF, ET, XGBoost, NB, and SVM are trained for each vector contained in V and G using the NSLKDDTrain+. Finally, the evaluation phase is implemented in two steps. The initial step consists of validating the different models using the NSLKDDVal. This step is crucial because it ensures that the training process was carried out correctly. The second step is the validation phase whereby the various models are tested and evaluated using the performance metrics outlined in Section 6.2. The testing process is conducted using the NSLKDDTest. This data subset is fully independent of the NSLKDDTrain+ and the NSLKDDVal.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

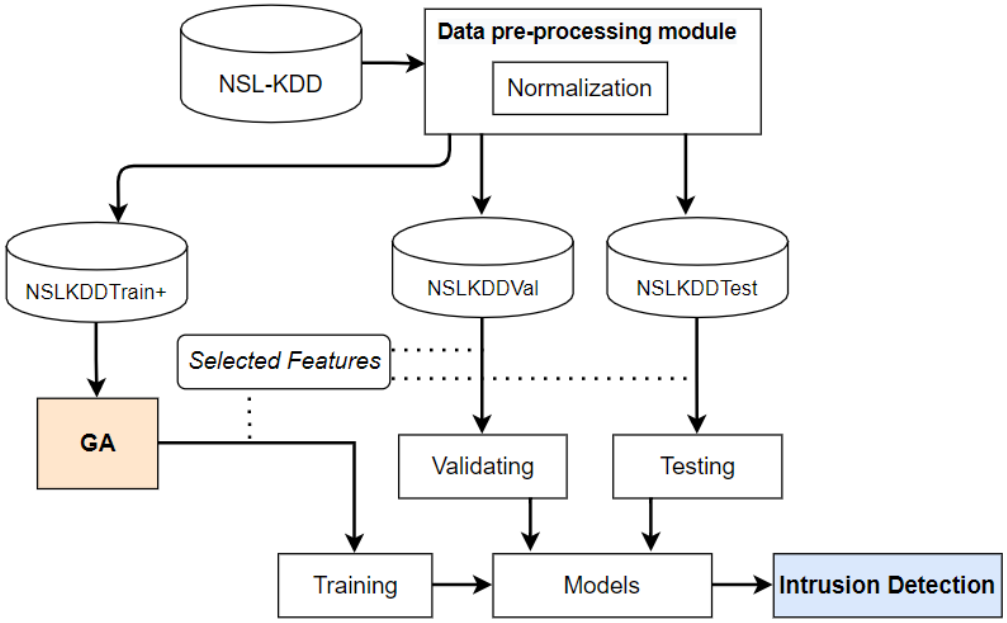


Figure 3. The architecture of the proposed intrusion detection system.

6. Experimental Setup

6.1. Hardware and Software settings

In this study, the experimental processes were carried out using a Notebook computer with the following specifications: DELL 153000 series, Windows 10 Operating System (OS), Intel Core i7-8568U-CPU processor, 1.8 GHz - 1.99 GHz. Moreover, to implement the ML models proposed in this research, we utilized a Python-based ML framework called Scikit-learn [48].

6.2. Performance Metrics

In the modeling phase described in Section 5.2, we framed the intrusion detection as an ML classification task. Therefore, the performance metrics that are employed to evaluate the performance of the models implemented in this study are accuracy (AC), precision (PR), recall (RC), and F1-Score (F1S) [49–51]. The F1S is the harmonic mean of the RC and PR. These metrics are derived from the following scenarios [51]:

- True positive (TP): intrusions that are correctly labelled as attacks.
- True Negative (TN): normal network traffic patterns that are correctly categorized as legitimate.
- False positive (FP): non-intrusive activities that are incorrectly classified as intrusions.
- False Negative (FN): malicious network traffic patterns that are labelled as normal.

$$AC = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

$$PR = \frac{TP}{TP + FP} \tag{9}$$

$$RC = \frac{TP}{TP + FN} \tag{10}$$

$$F1S = \frac{TP}{TP + FN} \tag{11}$$

In addition to the above performance metrics, for the binary classification scheme, we also plot the Receiver Operating Characteristic (ROC) curves of each model that was

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

implemented. A ROC curve allows us to compute the area under the curve (AUC). The AUC is a measure that determines the quality of the classification process. The AUC is always in the range of $[0, 1]$. A high AUC (close to 1) indicates an excellent classification process [52]. Moreover, in this study the test accuracy (TAC) is considered as the most important metric because it is a score that is computed on previously unseen data (Data that is independent from the training and validation data subsets). Additionally, we plot the confusion matrices (CMs) [53] for the multiclass classification process. A CM allows us to assess how a specific model performed for a each class in the dataset.

6.3. Results and Discussions

This section outlines the details of the experimental processes that were carried out in this research; it discusses the results and compares the outcome of our proposed methods with those of the existing systems. The experiments were conducted in two parts. Firstly, the binary classification process was implemented. In this phase, we only considered two classes (Intrusion: 1 and Normal: 0). Secondly, the multiclass classification task was carried out. In this step, we considered all five labels that are present in the NSL-KDD dataset (R2L: 1, U2R: 2, Probe: 3, DoS: 4 and Normal: 0).

6.3.1. Binary Classification

Table 5 - Table 11 depict the result that were obtained by the DT, SVM, RF, ET, XGB, and NB methodologies for the binary classification process using the optimal feature vectors v_1 - v_7. In each of the tables, the VAC is the accuracy that was obtained on validation data (KDDVal) and the TAC is the accuracy that was achieved on test data (KDDTest+). In terms of VAC, the classifier that achieved the highest performance is the RF with a VAC of 99.88% using v_4. In terms of TAC, the model that obtained the highest TAC is the DT with an accuracy score of 89.26% using v_3. Moreover, the DT (using v_3) achieved a VAC of 99.82%, a PR of 96.53%, an RC of 80.67%, and an F1S of 87.89%. Furthermore, in this experiment, we used the NB and SVM classifiers as the baseline models. In terms of VAC, the best SVM achieved a score of 95.22% using v_4. On the other hand, the most optimal SVM attained a TAC of 84.88% using v_2. With regards to VAC, the best NB model achieved a score of 89.52% using v_2. In respect of the TAC, the most optimal NB model obtained a score of 80.32%.

Table 5. Performance Results obtained by v_1 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_1	99.75 %	87.84 %	95.11 %	78.90 %	86.25 %
SVM	v_1	91.78 %	81.34 %	95.81 %	64.20 %	76.88 %
RF	v_1	99.76 %	86.48 %	95.20 %	75.86 %	84.44 %
ET	v_1	99.70 %	86.52 %	95.05 %	76.06 %	84.50 %
XGB	v_1	99.42 %	87.03 %	96.21 %	76.16 %	85.02 %
NB	v_1	86.06 %	79.74 %	78.68 %	62.40 %	74.85 %

Table 6. Performance Results obtained by v_2 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_2	99.76 %	88.69 %	95.21 %	80.65 %	87.33 %
SVM	v_2	91.22 %	84.88 %	89.82 %	77.50 %	83.21 %
RF	v_2	99.81 %	86.35 %	95.23 %	75.54 %	84.25 %
ET	v_2	99.77 %	86.45 %	95.41 %	75.61 %	84.36 %
XGB	v_2	99.54 %	86.80 %	96.40 %	75.51 %	84.68 %
NB	v_2	89.52 %	79.19 %	94.74 %	60.29 %	73.69 %

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Table 7. Performance Results obtained by v_3 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_3	99.82 %	89.26 %	96.53 %	80.67 %	87.89 %
SVM	v_3	93.48 %	80.07 %	90.23 %	65.91 %	76.17 %
RF	v_3	87.62 %	87.62 %	96.54 %	77.16 %	85.77 %
ET	v_3	99.72 %	87.75 %	96.45 %	77.50 %	85.94 %
XGB	v_3	99.62 %	86.89 %	96.29 %	75.80 %	84.83 %
NB	v_3	86.77 %	79.15 %	96.20 %	59.20 %	73.30 %

Table 8. Performance Results obtained by v_4 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_4	99.82 %	89.11 %	95.96 %	80.87 %	87.77 %
SVM	v_4	95.22 %	80.17 %	90.47 %	65.91 %	76.26 %
RF	v_4	99.88 %	86.59 %	96.05 %	75.35 %	84.45 %
ET	v_4	99.82 %	86.33 %	95.84 %	74.97 %	84.13 %
XGB	v_4	99.68 %	86.00 %	96.00 %	74.11 %	83.65 %
NB	v_4	88.98 %	79.11 %	93.22 %	61.23 %	73.91 %

Table 9. Performance Results obtained by v_5 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_5	99.80 %	88.60 %	95.51 %	80.18 %	87.17 %
SVM	v_5	95.13 %	82.17 %	90.49 %	70.53 %	79.27 %
RF	v_5	99.87 %	87.63 %	96.41 %	77.29 %	85.80 %
ET	v_5	99.81 %	88.25 %	96.69 %	78.36 %	86.57 %
XGB	v_5	99.55 %	87.90 %	96.40 %	77.87 %	86.15 %
NB	v_5	88.07 %	79.69 %	97.37 %	59.59 %	73.93 %

Table 10. Performance Results obtained by v_6 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_6	99.81 %	88.84 %	96.08 %	80.18 %	87.41 %
SVM	v_6	92.61 %	81.32 %	93.62 %	65.83 %	77.30 %
RF	v_6	99.86 %	86.51 %	95.89 %	75.31 %	84.36 %
ET	v_6	99.83 %	86.55 %	95.82 %	75.47 %	84.44 %
XGB	v_6	99.69 %	87.91 %	95.95 %	78.28 %	86.22 %
NB	v_6	87.83 %	79.35 %	96.08 %	59.72 %	73.66 %

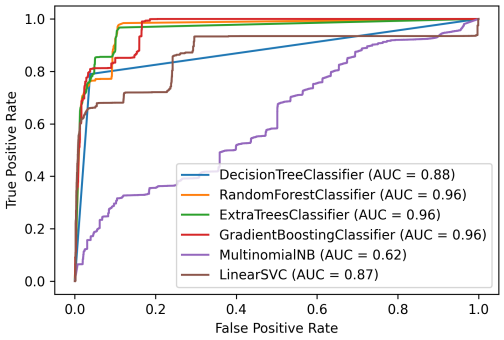
Table 11. Performance Results obtained by v_7 for the binary classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	v_7	99.77 %	86.81 %	93.59 %	78.06 %	85.12 %
SVM	v_7	89.91 %	79.74 %	93.68 %	62.27 %	74.81 %
RF	v_7	99.77 %	86.37 %	95.35 %	75.48 %	84.26 %
ET	v_7	99.73 %	86.29 %	95.21 %	75.42 %	84.17 %
XGB	v_7	99.47 %	86.73 %	96.46 %	75.32 %	84.59 %
NB	v_7	87.65 %	80.32 %	97.16 %	61.07 %	75.01 %

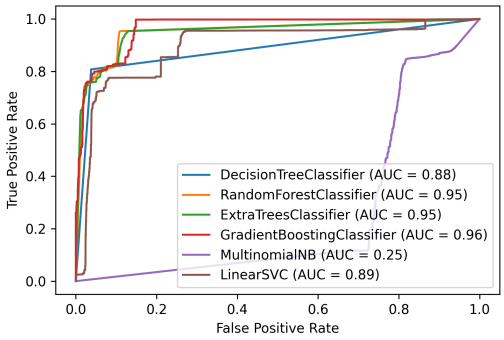
Moreover, for the binary classification process, we also plotted the ROC curves generated by each model (DT, SVM, RF, ET, XGB, NB). The ROCs curves were computed

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

in order to assess the quality of classification of each model using the AUC measure. Focusing on the tree-based models and as depicted in Figure 4a, the RF achieved an AUC of 0.96, the DT obtained an AUC of 0.89, the ET got an AUC of 0.96, and the XGB attained an AUC of 0.96. These results were obtained using v_1. In contrast, the baseline models (SVM and NB) obtained AUCs of 0.87 and 0.62, respectively. With regards to v_2 and as shown in Figure 4b, the DT, RF, ET and XGB achieved AUCs of 0.88, 0.95, 0.95, 0.96, respectively. Further, the SVM and the NB obtained AUCs of 0.89 and 0.25, respectively. In general, all the tree-based methods performed optimally of terms classification quality. This performance pattern is also supported by the AUCs measures obtained in Figure 5 - Figure 7.

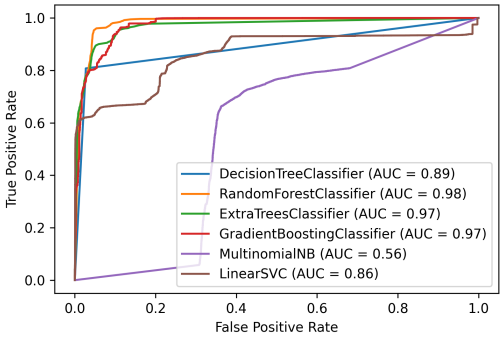


(a) ROCs for v_1

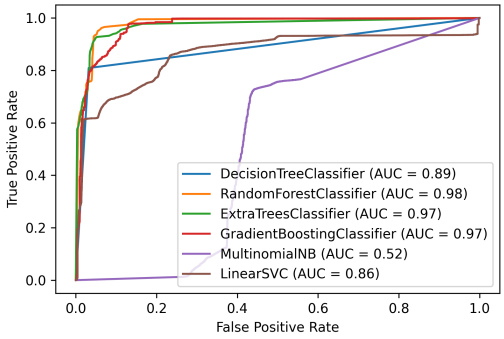


(b) ROCs Curves for v_2

Figure 4. Plot **(a)** shows the ROC curves of each model that was trained and tested using v_1. Plot **(b)** shows the ROC curves of each model that was trained and tested using v_2.



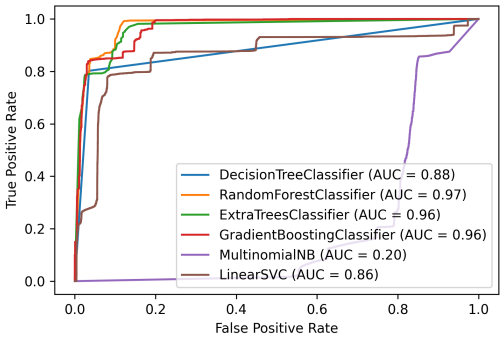
(a) ROCs for v_3



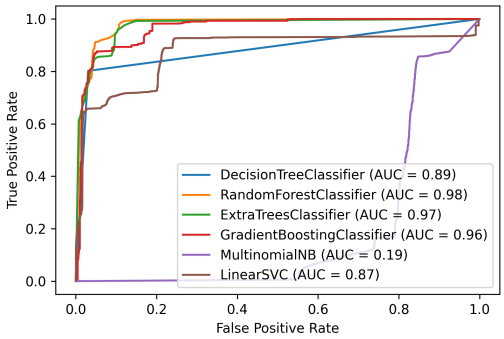
(b) ROCs for v_4

Figure 5. Plot **(a)** shows the ROC of each model that was trained and tested using v_3. Plot **(b)** shows the ROC curves of each model that was trained and tested using v_4.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1



(a) ROCs for v_5



(b) ROCs for v_6

Figure 6. Plot **(a)** shows the ROC of each model that was trained and tested using v_5. Plot **(b)** shows the ROC curves of each model that was trained and tested using v_6.

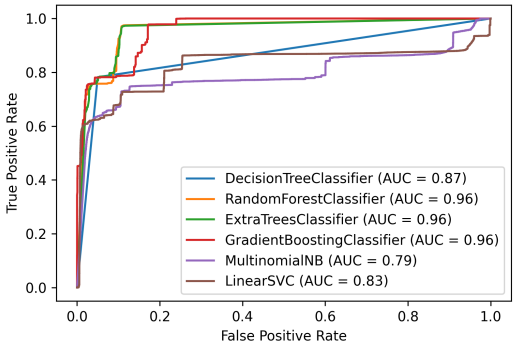


Figure 7. ROCs for v_7.

6.3.2. Multiclass Classification

Table 12 - Table 18 depict the outcomes that were achieved by the DT, SVM, RF, ET, XGB, and NB algorithms for the multiclass classification procedure using the optimal attribute vectors g_1 - g_7. Overall, in terms of TAC, the most optimal model was the XGB with a score of 87.26% using g_2. The same model achieved a VAC of 99.70%, a PR of 88.87%, and an F1S of 82.92%. Moreover, the baseline models, SVM and NB, achieved TACs of 77.61% and 65.98%, respectively. With regards to the VAC, the RF method obtained the highest score of 99.86% using g_5. The same classifier achieved a TAC of 86.84% and an F1S of 82.29%. In general, from g_1 to g_7, the XGB classifier performed optimally in terms of the TAC that was obtained. Moreover, Figure 8 depicts the confusion matrix (CM) that was generated using the DT method with g_2. This CM shows that the model was effective in detecting *Normal*, *DoS*, and *Probe* types of traffic patterns. However, the model unperformed with regards to detecting the *R2L* and *U2R* types of intrusions. This trend could also be observed in Figure 9 which depicts the CM generated by the DT algorithm using g_2. The detection accuracy of minority classes (*R2L* and *U2R*) is one of the major areas that we intend to investigate in our future works. The increase of the detection accuracy of minority classes will improve on the overall performance of the proposed models.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Table 12. Performance Results obtained by g_1 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_1	99.82 %	84.02 %	84.67 %	84.02 %	80.32 %
SVM	g_1	90.77 %	81.36 %	86.54 %	81.36 %	78.34 %
RF	g_1	99.84 %	85.97 %	81.17 %	85.97 %	80.77 %
ET	g_1	99.81 %	85.09 %	80.53 %	85.09 %	79.96 %
XGB	g_1	99.66 %	86.11 %	88.91 %	86.11 %	81.57 %
NB	g_1	80.63 %	61.53 %	61.53 %	62.53 %	59.29 %

Table 13. Performance Results obtained by g_2 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_2	99.82 %	87.14 %	88.92 %	87.14 %	82.92%
SVM	g_2	92.41 %	77.61 %	80.20 %	77.61 %	73.49%
RF	g_2	99.86 %	86.83 %	88.74 %	86.83 %	82.73%
ET	g_2	99.79 %	85.69 %	85.47 %	85.69 %	80.67%
XGB	g_2	99.70 %	87.26 %	88.87 %	87.26 %	83.02%
NB	g_2	81.87 %	65.98 %	78.99 %	65.98 %	64.36%

Table 14. Performance Results obtained by g_3 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_3	99.81 %	85.76 %	79.44 %	85.76 %	80.96 %
SVM	g_3	94.85 %	82.73 %	78.27 %	82.73 %	77.77 %
RF	g_3	99.85 %	86.33 %	84.50 %	86.33 %	81.10 %
ET	g_3	99.84 %	86.72 %	88.27 %	86.72 %	82.51 %
XGB	g_3	99.74 %	86.40 %	88.50 %	86.40 %	81.19 %
NB	g_3	40.34 %	33.26 %	50.97 %	33.26 %	23.00 %

Table 15. Performance Results obtained by g_4 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_4	99.74 %	86.05 %	88.05 %	86.05 %	81.91 %
SVM	g_4	93.92 %	78.76 %	72.35 %	78.76 %	74.62 %
RF	g_4	99.83 %	86.50 %	88.35 %	86.50 %	81.92 %
ET	g_4	99.75 %	86.47 %	88.29 %	86.47 %	82.12 %
XGB	g_4	99.77 %	86.50 %	82.88 %	86.50 %	81.32 %
NB	g_4	80.57 %	61.34 %	69.37 %	61.34 %	57.10 %

Table 16. Performance Results obtained by g_5 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_5	99.81 %	86.27 %	88.80 %	86.27 %	84.51 %
SVM	g_5	90.89 %	78.59 %	69.41 %	78.59 %	73.66 %
RF	g_5	99.86 %	86.84 %	84.84 %	86.84 %	82.29 %
ET	g_5	99.81 %	85.70 %	77.99 %	85.70 %	80.61 %
XGB	g_5	99.78 %	86.52 %	85.06 %	86.52 %	81.36 %
NB	g_5	81.54 %	62.53 %	66.02 %	62.53 %	58.08 %

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

Table 17. Performance Results obtained by g_6 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_6	99.78 %	85.81 %	87.67 %	85.81 %	81.04 %
SVM	g_6	90.10 %	79.65 %	76.22 %	79.65 %	76.30 %
RF	g_6	99.82 %	86.10 %	88.23 %	86.10 %	80.93 %
ET	g_6	99.74 %	85.70 %	87.26 %	85.70 %	80.65 %
XGB	g_6	99.79 %	87.03 %	88.80 %	87.03 %	82.57 %
NB	g_6	81.37 %	63.63 %	73.49 %	63.63 %	62.51 %

Table 18. Performance Results obtained by g_7 for the multiclass classification process.

Classifier	Feature Vector	VAC	TAC	PR	RC	F1S
DT	g_7	99.79 %	86.79 %	88.61 %	86.79 %	83.40 %
SVM	g_7	84.01 %	61.86 %	64.18 %	61.86 %	54.03 %
RF	g_7	99.87 %	86.25 %	84.46 %	86.25 %	81.03 %
ET	g_7	99.83 %	86.13 %	87.86 %	86.13 %	81.35 %
XGB	g_7	99.74 %	86.85 %	88.61 %	86.85 %	82.56 %
NB	g_7	81.55 %	66.10 %	79.45 %	66.11 %	63.81 %

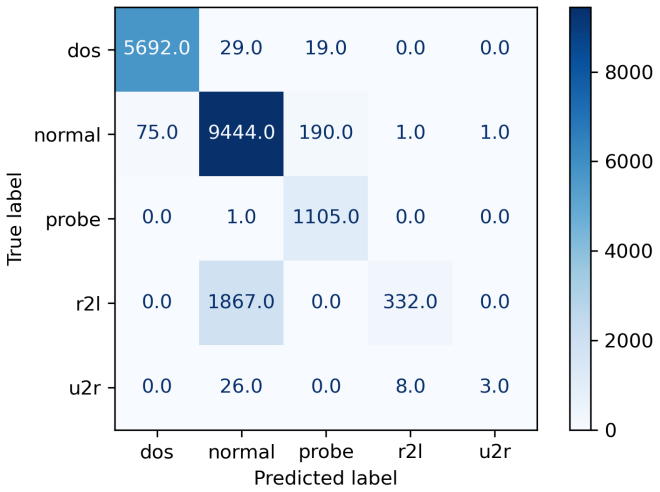


Figure 8. XGB model Confusion Matrix using g_2.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2021 doi:10.20944/preprints202106.0710.v1

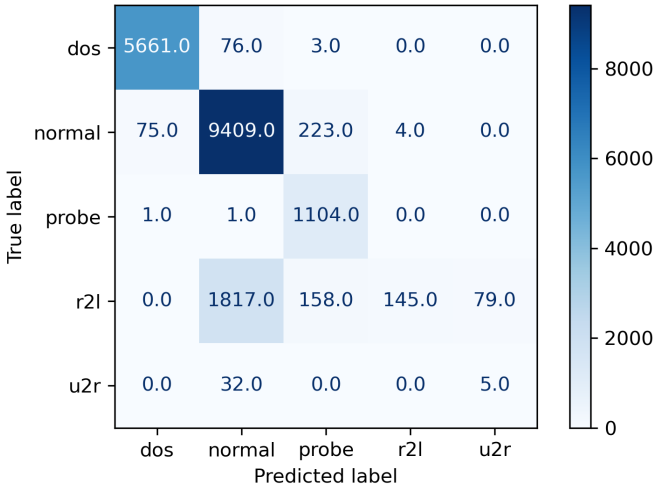


Figure 9. DT model Confusion Matrix us g_2.

6.3.3. Further Discussions

The results that were discussed in Section 6.3.1 and Section 6.3.2 demonstrate that the use of GA for feature selection on the NSL-KDD dataset has the potential to increase the performance of several classifiers. Moreover, Table 19 provides the details of the comparative analysis that was conducted between the results obtained in the experimental processes of the proposed research and some existing approaches. This contrast was conducted to put our results into perspective. In the proposed research, the GA-DT using v_3 was the best model in terms of TAC in the binary classification procedure. This model achieved an improvement of 6% of TAC in comparison to the GA-ANN in [21]. In contrast to BAT [22], the GA-DT (v_3) obtained an increase of 6.70%. Further, in comparison to the PSO-ANN [21] and the BAT-MC [22], the GA-DT (v_3) improved by 0.36% and 5.01%, respectively. With regards to the multiclass classification process, the GA-XGB using g_2 attained the highest TAC. In contrast to the work presented in [24], the GA-XGB ameliorated by 7.52 %. With regards to the AS-CNN in [25], the GA-XGB improved by 7.26 %. Moreover, in contrast to the DE-ELM in [28], Filter-SVM in [29] and the Wrapper-SVM in [29]; the TAC of the GA-XGB improved its performance by 7.11 %, 10.09 %, an 7.61 %, respectively.

 doi:10.20944/preprints202106.0710.v1

Table 19. Comparison with existing methods

Model	Binary Accuracy	Multiclass Accuracy
DNN [20]	77.75 %	-
PSO-ANN [21]	88.90 %	-
GA-ANN [21]	83.11 %	-
BAT [22]	82.56 %	-
BAT-MC [22]	84.25 %	-
MuliTree [23]	85.20 %	-
AE-DNN [24]	-	79.74%
AS-CNN [25]	-	80.00%
TSE-IDS [26]	85.79%	-
BGSA-MI [27]	88.36%	-
RelieF [27]	84.60%	-
DE-ELM [28]	87.83%	80.15%
Filter-SVM [29]	-	77.17%
Wrapper-SVM [29]	-	79.65%
GA-DT (Proposed)(v_3)	89.26%	-
GA-XGB (Proposed)(g_2)	-	87.26%

7. Conclusion

In this research, we implemented ML-based algorithms to develop efficient IDSs using the NSL-KDD dataset. A wrapper-based FS method using the GA was employed to select the most optimal features. The ET algorithm was used in the fitness function of GA. In the instance of the binary classification procedure, the GA selected 7 feature subsets. For the multiclass classification task, the GA selected 7 attribute subsets. The modelling process was conducted using the following ML approaches: DT, SVM, RF, ET, XBG. The experiments were carried out for both the binary and multiclass-classification schemes. Moreover, we compared the results obtained by our proposed methods with existing approaches. The outcome showed that GA-DT achieved a detection accuracy of 89.26% for the two-way classification scheme and an AUC of 0.89. The GA-XGB obtained an intrusion detection accuracy of 87.26% for multiclass classification process. These results were superior to those obtained by existing methodologies.

Further, the NSL-KDD dataset is an excellent benchmark dataset that is used for developing ML-based IDSs, however, it has some shortfalls regarding novel attacks. In our future work, we intend to apply our proposed framework to the following datasets, UNSW-NB15 and TON_IoT. The UNSW-NB15 is more advanced, and more complex than the NSL-KDD. This will allow us to assess the effectiveness of our method on complex datasets. The TON_IoT is a recently developed dataset that mainly contains traces of IoT-based network traffic patterns. Using this dataset has the potential to help us adapt our method to the IoT and Industrial-IoT (IIoT).

Funding: This research received no external funding.

Data Availability Statement: The data is available upon request.

Conflicts of Interest: The authors declare no conflict of interest

Abbreviations

The following abbreviations are used in this manuscript:

GA	Genetic Algorithm
IDS	Intrusion detection system
RF	Random Forest
DT	Decision Tree
SVM	Support Vector Machine
NB	Naive Bayes
XGB	Extreme Gradient Boosting
ET	Extra Trees
RC	Recall
PR	Precision
F1S	F1 Score
FS	Feature Selection
IoT	Internet of Things
IIoT	Industrial Internet of Things

References

1. Varga, P.; Plosz, S.; Soos, G. and Hegedus, C. Security threats and issues in automation IoT. In IEEE 13th Int. Workshop on Factory Commun. Sys. (WFCS), 2017 May 31, pp. 1-6.
2. Karatas, G.; Demir, O. and Sahingoz, O.K. Deep learning in intrusion detection systems. In Int. Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) IEEE, 2018 Dec 3, pp. 113-116.
3. Moorthy, M. and Sathiyabama, S. A study of Intrusion Detection using data mining. In IEEE Int. Conf. On Advances In Eng. Sci. And Management (ICAESM-2012) IEEE, 2012 Marc 30, pp. 8-15.
4. Ring, M.; Wunderlich, S.; Scheuring, D.; Landes, D. and Hotho, A. A survey of network-based intrusion detection data sets. Computers & Security, **219**, *86*,147–167.
5. Ananthakumar, A.; Ganediwal, T. and Kunte, A. Intrusion detection system in wireless sensor networks: a review. International Journal of Advanced Computer Science and Applications, **2015**, *6(12)*, 131–139.
6. Duhan, S. and Khandnor, P. Intrusion detection system in wireless sensor networks: A comprehensive review. In Int. Conf. on Electrical, Electronics, and Optimization Techn. (ICEEOT) IEEE, 2016 March, pp. 2707-2713.
7. Carleo, G.; Cirac, I.; Cranmer, K.; Daudet, L.; Schuld, M.; Tishby, N.; Vogt-Maranto, L. and Zdeborová, L. Machine learning and the physical sciences. Reviews of Modern Physics, **2019**, *91(4)*, p.045002.
8. Qawqzeh, Y.K.; Otoom, M.M.; Al-Fayez, F.; Almarashdeh, I.; Alsmadi, M. and Jaradat, G. A Proposed Decision Tree Classifier for Atherosclerosis Prediction and Classification. IJCSNS, **2019**, *19(12)*, p.197.
9. Noshad, Z.; Javaid, N.; Saba, T.; Wadud, Z.; Saleem, M.Q.; Alzahrani, M.E. and Sheta, O.E. Fault detection in wireless sensor networks through the random forest classifier. Sensors, **2019**, *19(7)*, p.1568.
10. Alsariera, Y.A.; Adeyemo, V.E.; Balogun, A.O. and Alazzawi, A.K. Ai meta-learners and extra-trees algorithm for the detection of phishing websites. IEEE Access, **2020**, *8*, pp.142532-142542.
11. Babajide Mustapha, I. and Saeed, F. Bioactive molecule prediction using extreme gradient boosting. Molecules, **2016**, *21(8)*, p.983.
12. Pisner, D.A. and Schnyer, D.M. Support vector machine. Machine Learning. Academic Press, **2020**, p. 101-121.
13. Saritas, M.M. and Yasar. Performance analysis of ANN and Naive Bayes classification algorithm for data classification. Int. Journ. of Intelligent Systems and Applications in Engineering, 7(2), pp.88-91.
14. Potluri, S.; Ahmed, S. ; Diedrich, C. Convolutional neural networks for multi-class intrusion detection system. In Int. Conf. on Mining Intelligence and Knowledge Exploration, 2018 Dec, pp. 225-238.
15. Chandrashekar, G. and Sahin, F. A survey on feature selection methods. Computers & Electrical Engineering, **2014**, *40(1)*, 16–28.
16. Rouhi, A. and Nezamabadi-pour, H. Filter-based feature selection for microarray data using improved binary gravitational search algorithm. In 3rd Conf. on Swarm Intelligence and Evolutionary Computation (CSIEC) IEEE, 2018 March 6, pp. 1-6.
17. Hui, K.H.; Ooi, C.S.; Lim, M.H.; Leong, M.S. and Al-Obaidi, S.M. An improved wrapper-based feature selection method for machinery fault diagnosis. PloS one, **2017**, *12(12)*, e0189143.
18. Sivanandam, S.N.; Deepa, S.N. Genetic algorithms. In Introduction to genetic algorithms, Springer, Berlin, Heidelberg, 2008, p 1537.
19. Ahmad, M.W.; Reynolds, J. and Rezgui; Y. Predictive modelling for solar thermal energy systems: A comparison of support vector regression, random forest, extra trees and regression trees. Journal of cleaner production, 2018, *203*, 810–821.
20. Tang, T.A.; Mhamdi, L.; McLernon, D.; Zaidi, S.A.R. and Ghogho. Deep learning approach for network intrusion detection in software defined networking. In international conference on wireless networks and mobile communications (WINCOM), Morocco, 2016 Oct 26, pp. 258-263.
21. Hosseini, S. A new machine learning method consisting of GA-LR and ANN for attack detection. Wireless Networks,**2020**, *26(6)*, 4149–4162.
22. Su, T.; Sun, H.; Zhu, J.; Wang, S. and Li, Y. BAT: deep learning methods on network intrusion detection using NSL-KDD dataset. IEEE Access, **2020**, *8*, 29575–29585.

23. Gao, X.; Shan, C.; Hu, C.; Niu, Z. and Liu, Z. An adaptive ensemble machine learning model for intrusion detection. IEEE Access, **2019**, *7*, 82512–82521.
24. Zhang, C.; Ruan, F.; Yin, L.; Chen, X.; Zhai, L. and Liu, F. A deep learning approach for network intrusion detection based on NSL-KDD dataset. In IEEE 13th Int. Conf. on Anti-counterfeiting Sec. Identification (ASID), 2019 Oct 25, pp. 41-45.
25. Hu, Z.; Wang, L.; Qi, L.; Li, Y. and Yang, W. A Novel Wireless Network Intrusion Detection Method Based on Adaptive Synthetic Sampling and an Improved Convolutional Neural Network. IEEE Access, **2020**, *8*, 195741–195751.
26. Tama, B.A.; Comuzzi, M. and Rhee, K.H. TSE-IDS: A two-stage classifier ensemble for intelligent anomaly-based intrusion detection system. IEEE Access, **2019**, *7*, 94497–94507.
27. Bostani, H. and Sheikhan, M. Hybrid of binary gravitational search algorithm and mutual information for feature selection in intrusion detection systems. Soft. Comput., **2017**, *21(9)*, 2307–2324.
28. Almasoudy, F.H.; Al-Yaseen, W.L. and Idrees, A.K. Differential Evolution Wrapper Feature Selection for Intrusion Detection System. Procedia Computer Science, 2020, *167*, 1230–1239.
29. Sarumi, O.A.; Adetunmbi, A.O. and Adetoye, F.A. Discovering computer networks intrusion using data analytics and machine intelligence. Scientific African, 2020, *9*, e00500.
30. Ieracitano, C.; Adeel, A.; Gogate, M.; Dashtipour, K.; Morabito, F.C.; Larijani, H.; Raza, A. and Hussain, A. Statistical analysis driven optimized deep learning system for intrusion detection. In Int. Conf. on Brain Inspired Cognitive Systems, 2018 Jul 7, pp. 759-769.
31. Safavian, S.R. and Landgrebe, A survey of decision tree classifier methodology. IEEE trans. sys man cybernetics, 1991, *21(3)*, 660–674.
32. Priyanka and Kumar, D. Decision tree classifier: a detailed survey. Int. Journ. Inf. Decision Sci., 2020 , *12(3)*, 246–269.
33. Izquierdo-Verdiguier, E. and Zurita-Milla, R. An evaluation of Guided Regularized Random Forest for classification and regression tasks in remote sensing. Int. Journ. Applied Earth Observation and Geoinformation, 2020 , *88*, 102051.
34. Oshiro, T.M.; Perez, P.S. and Baranauskas, J.A. How many trees in a random forest?. In Int. workshop on machine learning and data mining in pattern recognition, July 2012, New York, USA, pp. 154-168.
35. Alsariera, Y.A.; Adeyemo, V.E.; Balogun, A.O. and Alazzawi, A.K. AI meta-learners and extra-trees algorithm for the detection of phishing websites. IEEE Access, 2020, *8*, 142532–142542.
36. Devan, P. and Khare, N., 2020. An efficient XGBoost–DNN-based classification model for network intrusion detection system. Neural Computing and Applications,2020, 1–16.
37. Scikit-Learn: Ensemble Gradient Boosting Classifier. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (accessed on 21 May 2021).
38. Scikit-Learn: Naive Bayes. Available online: https://scikit-learn.org/stable/modules/naive_bayes.html (accessed on 21 May 2021).
39. Wongso, R.; Luwinda, F.A.; Trisnajaya, B.C. and Rusli, O. News article text classification in indonesian language. Procedia Computer Science, 2017, *116*, 137–143.
40. Ahmad I.; Basheri, M.; Iqbal, M.J. and Raheem, A. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. IEEE Access, **2018**, *6*, 33789-–33795.
41. Gopi, A.P.; Jyothi, R.N.S.; Narayana, V.L. and Sandeep, K.S. Classification of tweets data based on polarity using improved RBF kernel of SVM. Int. Journ. Inf. Tech., **2020**, 1–16.
42. Hussain, M.; Wajid, S.K.; Elzaart, A. and Berbar, M. A comparison of SVM kernel functions for breast cancer detection. In 8th IEEE Int. Conf. Comput., Graphics, Imaging and Visualization, Singapore, August 2011, pp. 145-150.
43. Whitley, D. A genetic algorithm tutorial. Statistics and computing, 1994, *4(2)*, 65–85.
44. Genlin, J. Survey on genetic algorithm. Computer Applications and Software, 2004, *2*, 69–73.
45. Natural Gas Liquefaction Cycle Enhancements and Optimization. Available online: https://doi.org/10.1016/B978-0-12-404585-9.00005-2 (accessed on 27 May 2021)
46. Jain, S; Shukla, S.; Wadhvani, R. Dynamic selection of normalization techniques using data complexity measures. Expert Systems with Applications, 2018, *106*, 252–262.
47. Cao, X.H.; Stojkovic, I.; Obradovic, Z. A robust data scaling algorithm to improve classification accuracies in biomedical data. BMC bioinformatics, 2016, *17(1)*, 1–10.
48. Scikit-learn: Machine Learning in Python. Available online: https://scikit-learn.org/stable/ (accessed on 28 May 2021)
49. Classification: Accuracy. Available online: https://developers.google.com/machine-learning/crash-course/classification/accuracy (accessed on 28 May 2021)
50. Classification: Precision and Recall. Available online: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (accessed on 28 May 2021)
51. Huang, H.; Xu, H.; Wang, X.; Silamu, W. Maximum F1-score discriminative training criterion for automatic mispronunciation detection. IEEE/ACM Trans on Audio, Speech, and Language Processing, 2015, *23(4)*, 787–797.
52. Fan, J.; Upadhye, S. ; Worster, A. Understanding receiver operating characteristic (ROC) curves. Canadian Journ. of Emergency Medicine, 2006, *8(1)*, 19–20.
53. Tharwat, A. Classification assessment methods. Applied Computing and Informatics, 2020.