ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

A COMPARATIVE ANALYSIS OF GRADIENT BOOSTING, RANDOM FOREST AND DEEP NEURAL NETWORKS IN INTRUSION DETECTION SYSTEM

Iftikhar Ahmad and Hadi S. AlQahtani
Faculty of Computing and Information Technology, King Abdulaziz University (KAU), Jeddah Saudia Arabia
E-Mail: iakhan@kau.edu.sa

ABSTRACT

The growing threat of advanced security attacks targeting enterprise information systems raises the need for novel security solutions that promptly identify and respond to these issues. These security strategies must automate threat detection and response in enterprise settings, enabling organizations to address emerging threats, ongoing attacks, and imminent risks adequately. Traditional security strategies that rely on rule-based approaches for intrusion detection systems are inefficient in achieving these objectives due to their limited capabilities in identifying new threats. As a result, machine learning strategies have been proposed to address these needs, offering an intelligent detection environment for novel threats. Classification algorithms such as random forest, gradient boosting and deep learning techniques like deep neural networks have been proposed in various studies. This paper examines the performance of these models, providing a comparative review of their detection capabilities based on precision, recall, accuracy, specificity, and sensitivity. The models are tested using a Python environment due to the extensive machine learning capabilities. These tests show that random forest is the ideal model for network-based intrusion detection systems.

Keywords: deep neural network, gradient boosting, IDS, machine learning, malware, random forest, security.

Manuscript Received 23 June 2023; Revised 19 August 2023; Published 30 August 2023

1. INTRODUCTION

ORGANIZATIONS, institutions, and agencies deploy diverse security strategies and solutions to handle emerging security breaches and increasing information risks. For instance, these companies implement firewall solutions to monitor and block malicious traffic, antivirus, antimalware services to scan system files, and security policies to entrench security culture. Although these initiatives reasonably provide a suitable security structure, they are inefficient in the growing security landscape. The rationale behind this situation is that hackers increasingly leverage advanced attack strategies such as advanced persistent threats, distributed denial of services, and ransomware-as-a-service approaches to bypass the established security mechanisms. These attack advancements compromise the efficiency of the existing tools due to their limited detection and response frameworks, rendering the security investments infeasible to meet enterprise IT goals.

Different strategies are developed to address these security needs, offering considerable remedies to the enterprise information infrastructure. For instance, there are security-as-a-service strategies managed by cloud service providers, enabling firms to implement managed security functionalities in their infrastructure [1]. These security solutions are efficient for the cloud-based information infrastructure, limiting their optimality for conventional computing systems. As a result, it becomes challenging for traditional information of enterprise architectures to rely on the solutions to secure their operations. The alternative to this security solution is the intrusion detection system (IDS), which strengthens the capabilities of the firewalls and antimalware services to enhance threat detection in enterprise information infrastructure. The IDS monitors corporate traffic, system activities, user behavior, and enterprise files to identify suspicious behavior that fails to conform to the expected security considerations [2]. This comprehensive monitoring enhances the detection of a breach before it compromises the corporate systems, notifying IT security regarding the imminent attack and potential mitigations to handle the incident.

This paper examines the development of an intelligent IDS based on gradient boosting, random forest, and deep neural networks. Reviewing these detection strategies offers a comparative assessment of the resulting IDS solution, enhancing understanding of their efficiency in handling enterprise security needs. The study's primary goal is to identify the best intelligent framework for developing IDS solutions, enabling them to handle emerging security needs. This goal is achieved by reviewing existing strategies for the IDS solutions, comparing the performance of different IDS, and selecting the suitable framework based on the performance metrics.
The rest of the paper is structured as a literature review, methodology, discussion, and conclusion. The literature review provides an extensive review of the related works, assessing other IDS frameworks and their inefficiencies in meeting security needs. This approach offers the research gaps that are addressed by the current study. The methodology develops the three models and tests their performance, while the discussion reviews the results obtained from these tests.

2. LITERATURE REVIEW

ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

A. Overview of IDS Solutions

The intrusion detection systems are implemented in two significant environments: host and network. The host-based intrusion detection system operates (HIDS) on a single server, providing dedicated monitoring of the target host machine [3]. This dedicated threat monitoring examines the network traffic, system logs, user activities, security policies implemented in the server, configuration information, and other details that determine the system's security posture. The solution manages the specific endpoint, enabling the security team to respond to emerging needs effectively. The network-based IDS (NIDS) solution expands its capabilities by monitoring different devices deployed over the enterprise network [3]. This phenomenon implies such solutions manage hybrid devices in a local area network (LAN), core network infrastructure, campus-wide network, or enterprise infrastructure. The solution obtains logs and traffic from diverse endpoints in the infrastructure, examining them to determine whether they are malicious or benign. The response to these incidents is based on the implemented security configuration, enabling the organization to secure its architecture effectively.

The major strategies used by IDS solutions are signature-based and anomaly-based techniques. The signature-based detection strategy uses a pattern approach to identify malicious activity or threat in the information infrastructure, notifying the system administrators or security team regarding this potential intrusion [3]. A notable example of such a solution is Snort and Suricata, which use rules to identify malicious traffic targeting the information infrastructure. These IDS solutions are configured with suitable rules that classify network traffic as malicious or benign [4]. This configuration enables them to notify system administrators regarding potential denial of service (DoS) or distributed denial of service attacks (DDoS) targeting the core devices. Network traffic that matches the rules triggers an alert on the system, initiating the automated response mechanisms to remediate the situation [4]. Anomaly-based IDS solutions classify network traffic, system logs, and user activities using machine-learning approaches. This classification is based on the dataset used to train the model, enabling the developed IDS solution to handle zero-day threats [3]. This feature resolves the limitations of the signature-based IDS solutions that are incapable of handling novel attacks due to the absence of rules to classify such attacks.

Diverse machine learning models are used to develop IDS solutions. These models are supervised, semi-supervised, or unsupervised frameworks. Supervised learning models require labeled data for the training, where the algorithm learns to classify new data based on the training dataset [3]. Unsupervised learning involves using unlabelled data for the training process. The trained model's objectives are to identify patterns or relationships in the data used to handle new datasets. The semi-supervised learning integrates supervised and unsupervised approaches to strengthen the algorithm's performance on new datasets [3]. These training models lead to diverse techniques for the IDS solutions, such as support vector machines, decision trees, Bayes classifiers, random forests, gradient boost, and neural networks, among others. These models provide diverse training experience and performance features for the IDS solutions, leading to divergent results in the developed solutions.

This study is limited to gradient boost, random forest, and deep neural network approaches for IDS solutions. Random forests use an ensemble learning technique where different decision trees are integrated to predict traffic as malicious or benign [5]. Deep neural networks use deep learning techniques in artificial neural networks to effectively predict data through multiple hidden layers. On the other hand, gradient boosting is an ensemble learning technique that uses ensemble learning technique by combining weak prediction models to offer a robust prediction framework [5]. The model improves its prediction through iterative training, minimizing errors in the subsequent models.

Limiting the research to these models offers a comprehensive assessment of their performance in handling security threats in enterprise computing environments. This feature helps identify the suitable IDS solution to enhance the security posture and improve the infrastructure's ability to address emerging threats. Further, this focus provides an efficient analysis environment as the models leverage diverse techniques during the training process. Random forest and gradient boost are classification strategies for machine learning, while the deep neural network is a deep learning technique. Thus, this limited focus enhances the assessment of the IDS solution's efficiency in meeting the security objectives.

B. Review of Related Works

ElSayed *et al*. (2021) propose a hybrid deep learning approach based on the convolutional neural network (CNN) to classify the flow traffic into normal or attack classes [11]. The proposed strategy is a regularizer called SD-Reg, a method based on the standard deviation of the weight matrix. This framework addresses the problem of overfitting and improves the capability of network-based intrusion detection systems in detecting unseen intrusion events. The evaluation results indicate that the SD-Reg outperforms the previous regularizer methods [11]. In addition, the proposed hybrid technique performs more in all the evaluation metrics than single deep learning models. Several datasets, including the InSDN - the most recent dataset for software-defined networking - are used to train and evaluate the performance of all techniques [11]. Furthermore, they suggest a lightweight NIDS by training the convoluted neural network CNN-based models using fewer features without causing a significant drop in the model performance.

Chandak *et al.* (2019) use the feature selection algorithm for the artificial neural network classifier proposed by Akashdeep et al. (2017) in separate research for IDS solutions [12]. The new feature selection is implemented in the original dataset, leading to 29 features for the subset. This feature reduces the features used for developing IDS systems, preventing oversampling of the

ARPN Journal of Engineering and Applied Sciences

target dataset. The researchers use Naive Bayes and Random Forest classification algorithms to compare these reduced feature sets' performance [12]. The use of these algorithms provides a comprehensive environment for evaluating the efficiency of classification algorithms in developing intelligent IDS solutions. However, they note that oversampling one class may deteriorate the performance of another class. This situation compromises the reliability and efficiency of the experimentation process, a phenomenon the researchers extensively address by reducing the features used for solution modeling [12]. The research evaluates random under-sampling or oversampling of a specific class to design an optimal training dataset. The results show that the classification models developed using this training dataset have a better detection rate for the minority classes.

Iqbal and Aftab (2019) use Feed-Forward and Pattern Recognition Neural Network algorithms to develop a NIDS solution. The researchers designed, implemented, and tested the developed system using the KDD CUP99 dataset with slight modifications to enhance experimental specificity [13]. They used Scaled Conjugate Gradient and Bayesian Regularization training functions to train the feed-forward artificial neural networks. The performance metrics used to evaluate the neural network models are the accuracy, Mathew's Correlation Coefficient, R-squared, Mean Square Error, False Alarm Rate (FAR), Detection Rate (DR), and Area under Receiver operating characteristic (ROC) curve [13]. These models demonstrate substantial performance measures when subjected to different attack vectors. This feature highlights the efficiency of neural networks in detecting diverse threats for IDS solutions.

Rajagopal *et al*. (2020) review the performance of eight two-class and three multiclass algorithms using UNSW NB-15, a modern intrusion detection dataset. Their research uses 82,332 testing samples to evaluate the performance of algorithms, providing a comprehensive test environment for the classifiers [14]. Additionally, they use the Microsoft Azure Machine Learning Studio (MAMLS) to learn algorithms, as running the experiment on the local systems is time-consuming due to resource constraints. The experiment examined accuracy, precision, recall, f1-score, Area under Curve (AUC), training, and execution times [14]. Reviewing these extensive metrics offered a comprehensive performance evaluation environment for the proposed models, enhancing the practical adoption of NIDS solutions. The two-class decision forest model shows 99.2% accuracy and takes 6 seconds to learn the 1,75,341 network instances. Similarly, the multiclass decision forest model identified generic, exploits, shellcode, and worm attacks with recall percentages of 99%, 94.49%, 91.79%, and 90.9%. The decision forest provided the best performance features for the experiments, highlighting its efficiency in handling security needs for NIDS. Also, the study notes that MAMLS offers a robust environment for managing large datasets during machine learning operations.

Tama and Rhee review the gradient-boosted machine's (GBM) performance against the deep neural networks, random forest, classification, regression tree, and support vector machines. Further, the authors examine the performance of the proposed model against the random tree + NBTree, NBTree, Adaboost + GA, GAR Forest, fuzzy classifier, DT, Two-tier model, and Discriminate Multinomial Naïve-Bayes. The review examines how these machine learning models perform in anomaly-based IDS solutions [16]. The researchers leverage a grid search to evaluate the algorithms' performance, with the models being trained using the supervised machine learning strategy. The metrics used for comparison are the accuracy, sensitivity, specificity, false positive rate (FPR), and area under the ROC curve. The datasets used to examine the data are the UNSW-NB15, NSL-KDD, and GPRS using hold-out or ten-fold cross-validation techniques [16]. The proposed model outperforms other strategies by demonstrating an accuracy of 95.08% and an FPR of 2.97% for the tenfold cross-validation, an accuracy of 91.31%, and an FPR of 8.60% for the hold-out technique.

Naseer *et al*. (2020) review the implementation of deep neural networks in IDS solutions for anomaly detection. The researchers provide a comparative assessment of thirteen models for supervised machine learning. These models are Vanilla autoencoder, Sparse autoencoder, Denoising autoencoder, Contractive autoencoder, Extreme Learning Machine, Deep Convolutional neural network (DCNN), RBF SVM, J48, long-short-term memory (LSTM), Multilevel perceptron, Naive Bayes, Quadratic Discriminant Analysis, and Random-Forest [17]. The attacks examined are denial of service, user-to-root, remote-to-local attacks, and probing attacks. The model performance is reviewed based on the AUC, precision-recall, ROC curve, accuracy, mean average precision, FPR, training, and testing time [17]. The DCNN and LSTM demonstrated accuracy performance of 85% and 89%. These scores show the promises of the algorithms in handling NIDS security needs. The table below illustrates the key features of the reviewed studies.

ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

Table-1. Related Works.

Ref#	Author	Year	Method			Performance metrics		Limitation(s)
			Preprocessing	Classification	Analysis Technique	ACC	DR	
11	Mahmoud Said ElSayed, Nhien-An Le-Khac, Marwan Ali Albahar, Anca Jurcut	2021	The network traffic is in a one-dimensional data	binary and multiclass classification	SD-Reg Signature-based NIDS and anomaly-based	99.28%, 98.92%	Mentioned its equation without a result	SD-Reg is not ideal for feature selection or feature reduction
12	Trupti Chandak, Sanyam Shukla, Rajesh Wadhvani	2019		Random Forest for performance comparison.	ANN, Naive Bayes			Didn't provide the result of ACC and DR of performance metrics
13	Ahmed Iqbal, Shabib Aftab,	2019	-	FF-NN, PR	BR, SCG	99 %, 98 %	99%, 97%	Proposed solution tested only on one dataset and need to be tested on another datasets.
14	Smitha Rajagopal, Katiganere Siddaramappa Hareesha, Poornima Panduranga Kundapur	2020	-	Averaged perceptron, Bayes point machine, Boosted decision tree, Decision forest, Decision jungle, support vector machine), Logistic regression	Machine learning as a service (MLaaS)= (MAMLS)	99.2%	99.3%	High false alarms for the new attacks
15	Ahmed Ahmim, Mohamed Amine Ferrag, Leandros Maglaras, Makhlouf Derdour, and Helge Janicke	2019	-	J48 (C4.5), ForestPA, Random Forest, REP Tree, Jrip, FURIA, Ridor, MLP, RBF, LIBSVM, SVM, Naïve Bayes	Supervised Machine Learning	99.75%	99.78	Used only machine learning methods
16	Bayu Adhi Tama and Kyung-Hyune Rhee	2019	-	Deep Neural Network, Random Forest, Gradient Boosted Machine, SVM, Classification and Regression Tree	Supervised machine learning	93.65%	96.51%	Study reduced FP rate successfully, but the results of accuracy metric was unsatisfied.
17	Sheraz Naseer, Yasir Saleem, Shehzad Khalid, M Khawar Bashir, Jihun Han, M Munwar Iqbal and Kijun Han	2018	-	Vanilla autoencoder, Sparse autoencoder, Denoising autoencoder, Contractive autoencoder, Extreme Learning Machine, Deep Convolutional neural network, RBF SVM, J48, LSTM, Multilevel perceptron, Naive Bayes, Quadratic Discriminant Analysis, and Random-Forest	Supervised machine learning	86% 89%	-	Poor in anomaly detection

3. METHODOLOGY

The design process for the intrusion detection system uses four steps that address specific design issues. The first activity is selecting the dataset for the experiment, where the dataset is obtained from the public repository. The dataset used on the IDS solution is the CIC-MalDroid

or Android Malware dataset [6]. This dataset is up-to-date regarding the conventional attacks, resembling real-world captured attacks. Further, it aggregates samples from different sources such as the AMD, Contagio Security blog, VirusTotal service, and MalDozer [7]. Additionally, the dataset is grouped into different categories, offering a robust classification environment for security threats. These attributes make the dataset suitable for machine learning operations.

The second step is preprocessing, where the dataset is cleaned and prepared for the model. Preprocessing is to convert the dataset into a format suitable for the models' inputs, optimizing the training process. Three activities are performed during this step: deleting null values, splitting data, and feature selection. Deleting the null and infinity values in the dataset enhances the accuracy and completeness of the data. This approach improves the optimality of the dataset used for the training process, enhancing its accuracy in classifying threats as malicious or benign. Columns containing null or infinity values were deleted to clean the dataset. Splitting involves segmenting the dataset into portions for practical training and testing processes. The training set comprised 80% of the dataset, while the testing was 20% of the dataset. This approach provided an effective environment for model training and testing.

Feature selection involves identifying the attributes used for the training process to refine the model. Although there are about 77 features in the dataset, such extensive features are challenging to capture in the model as they increase their complexity during the modeling process [7]. As a result, reducing the number of features used for the training process is essential while retaining the model's performance. This approach enhances the efficiency of the trained model without affecting its accuracy in handling novel datasets. Further, it reduces data dimensionality, providing a simple but powerful representation of the target dataset. The features selected in the dataset led to five categories: benign, riskware, SMS malware, banking malware, and adware. These labels had different features in the dataset, enabling the training model to classify the dataset during the learning process appropriately. The five categories and associated sizes are shown in the table below:

Table-2. Dataset Categories.

Dataset Categories	# of Samples (Values)
Benign	2273097
Riskware	240697
SMS malware	158930
Banking malware	128095
Adware	29924

The third step is modeling, where the detection models are configured to predict malicious and benign activities. This process involves setting the functionalities of the detection framework for the IDS solution and creating the respective approaches needed to identify the malware. Three models are developed in this stage: gradient boosting, random forest, and deep neural network. The gradient boosting classifier integrates regression and classification tasks to classify malicious activities, optimizing its performance through weak prediction ensembles. The resulting model enhances its performance by reducing prediction errors iteratively, leading to the reliability and accuracy of the detection engine [5]. For instance, the first iteration may have a high error value for the classification and regression tasks. However, these values are reduced in the subsequent iterations to improve its accuracy. This iterative error reduction and model reliability is illustrated in Figure-1 below.

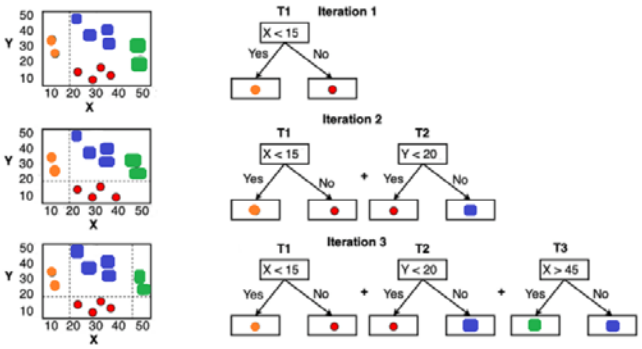

Figure-1. Visualizing gradient boosting.

The random forest classifier uses decision trees to classify activities as malicious or benign. This classification strategy offers a simple and efficient environment for threat detection, where data is split through iterative decision trees, as shown in Figure-2. The model comprises root, decision, and leaf nodes. The root node is the initial node in the decision tree, where data is segmented into different categories. This node provides the overall categorization for the dataset, as shown in Figure-3. The decision nodes are nodes resulting from the root node's classification. These nodes offer different sub-categories to the data, iteratively segmenting the dataset into smaller classification units. This iterative segmentation leads to the leaf nodes, where further separation is not feasible. The dataset categorization into these categories and sub-categories depends on differentiating features measured by entropy.

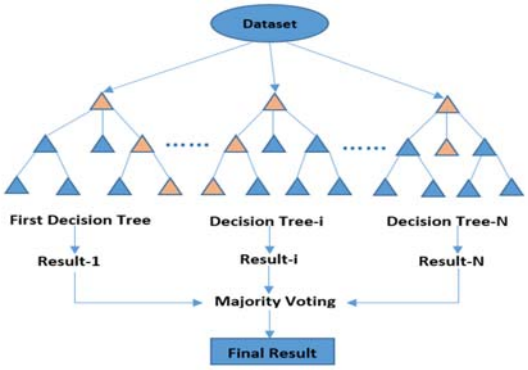

Figure-2. Random forest classifier model.

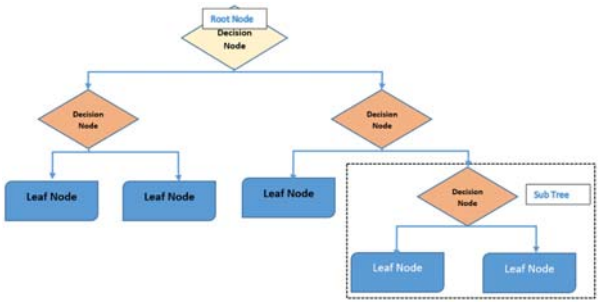

Figure-3. Nodes in random forest classifier.

The random forests for the decision-making process are constructed from the multiple random decision trees. These forests have two features that enhance randomization: random sampling of the original data and random selection of the features subset at each node during node construction. This construction of the decision trees is repeated until the stopping criterion is met. This criterion could be the lowest samples in the leaf nodes or the maximum depth. Ensembles are then created by combining the predictions from each tree. This feature provides the prediction for the classifier, enabling it to categorize new threats based on the decision trees.

The deep neural network model uses deep learning techniques for machine learning to classify activities as malicious or benign. The modeling technique uses an input layer that captures the inputs for the algorithm, a series of hidden layers, and the output layer. The hidden layers provide the model's transformation that helps it to predict data based on the inputs [5]. The model used in the IDS comprises three layers, where there is one input layer and two hidden layers. Additionally, there are 64 dense neurons leveraged in the model with a rectified linear unit (ReLU). The ReLU provides the activation function for the deep neural networks, resolving the diminishing gradient problem to provide enhanced output [8]. This feature improves the model's performance, offering enhanced prediction accuracy. Similarly, one dense neuron with a sigmoid activation is leveraged in the model. Figure-4 visualizes this deep neural network model for the IDS.

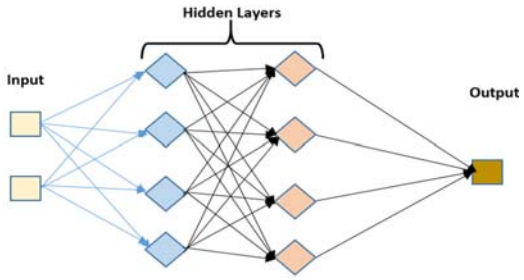

Figure-4. Visualizing deep neural networks.

The three models are implemented in Python programming language as they provide vast machine learning libraries. This feature speeds up the modeling process while optimizing the efficiency of the developed model [9]. Additionally, Python offers robust functionalities for dataset integration, preprocessing and feature selection, and model training and testing. Further, it provides powerful customization and automation features that speed up the training and testing operations. These functionalities optimize the machine learning operations by offering an integrated development environment [9]. The code used to develop the models and associated results are shown in the Appendix.

The last step is training and testing the model, where metrics are used to examine the performance. This testing is based on four parameters: accuracy, precision, sensitivity, and recall [10]. Accuracy denotes the overall correctness of the prediction model, while precision examines the model's reliability by comparing its true positives against all positives identified by the model. Sensitivity checks the true positives against the sum of the true positives and false negatives in the prediction. At the same time, specificity reviews the true negatives against the sum of the true negatives and false positives [10]. These metrics offer a comprehensive assessment of the prediction model, leading to the selection of the suitable IDS framework. The metrics are illustrated in the table below:

Table-3. Performance Metrics for the Models.

	Predicted Class		
Actual Class		**Class=Yes**	**Class=No**
	Class=Yes	a	b
	Class=No	c	d

Key:
a: TP (true positive)
b: FN (false negative)
c: FP (false positive)
d: TN (true negative)
Accuracy = TP+TN/(TP+FP+TN+FN)
Precision=TP/(TP+FP)
Recall=TP/(TP+FN)
Specificity = TN/(TN+FP)
Sensitivity = TP/(TP+FN)

ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

RESULTS AND DISCUSSIONS

A. Dataset Review

The CICMalDroid2020 dataset comprises 2,830,743 samples collected from different sources. This extensive dataset provides a comprehensive training environment for the machine learning models, refining their capabilities for intrusion detection. Additionally, the dataset is preprocessed and grouped into five categories. This preprocessing reduces the dimensionality of the dataset, improving the model's accuracy [18]. This situation enhances the model's reliability in handling novel attacks due to improved detection precision and prediction efficiency. Further, dimension reduction boosts the framework's performance, enabling it to function optimally for robust comparison with other algorithms [18]. This feature offers a suitable environment for selecting the ideal detection model for the IDS solution, as the framework is optimized like different algorithms in the prior studies. As a result, the network dataset used in the supervised machine learning strategy is one-dimension, offering a comprehensive training and testing environment for the models.

The preprocessed Android malware dataset comprises five categories. The categories address different malware groups for the Android platforms. For instance, adware malware provides annoying pop-ups, banking malware targets mobile banking platforms, and mobile riskware that malicious users exploit [7]. Further, there is SMS malware that leverages SMS to deliver the malware payload, and the benign instances do not fall in any malware category. These diverse groups extend the model's detection capabilities, enabling it to predict different malware categories based on the training dataset. The high number of samples in the categories strengthens the learning and testing operations by offering a reasonably large training dataset for malware detection in the individual category. Thus, the dataset enhances the efficiency of the trained model, improving its prediction and performance against different metrics.

B. Gradient Boosting Model

The gradient boosting model for classification was developed using the dataset, which was portioned into training and testing samples. There were 2,262,300 labels used during the training, where 1,817,049 entries were benign and 445,251 were malicious. The training process led to a precision of 99.774% for the harmless entries and 99.326% for the malware. Further, it had a recall of 99.835% for the benign entries and 99.077% for the malware. The accuracy for the model was 99.686%, while the sensitivity and specificity were 99.077% and 99.835%, as shown in Figure-5. The trained model is then subjected to the test sample, with 565,576 labels. The model demonstrates an accuracy of 99.3681%, sensitivity of 99.045%, and specificity of 99.836%. Further, the model's precision was 99.766% for the benign labels and 99.330% for the malicious entries. The recall results are 99.836% for the harmless entries and 99.045% for the malicious ones, as shown in Figure-6. This situation demonstrates the significant performance efficiency of the trained model in handling Android malware.

	precision	recall	f1-score	support
Benign	0.99774	0.99835	0.99805	1817049
Attack	0.99326	0.99077	0.99201	445251
accuracy			0.99686	2262300
macro avg	0.99550	0.99456	0.99503	2262300
weighted avg	0.99686	0.99686	0.99686	2262300

Sensitivity : 99.07670855766297
Specitivity : 99.83522733839318

Figure-5. Training results for gradient boosting classifier.

	precision	recall	f1-score	support
Benign	0.99766	0.99836	0.99801	454271
Attack	0.99330	0.99045	0.99187	111305
accuracy			0.99681	565576
macro avg	0.99548	0.99441	0.99494	565576
weighted avg	0.99680	0.99681	0.99680	565576

Sensitivity : 99.04496653339922
Specitivity : 99.8362211103064

Figure-6. Testing results for gradient boosting classifier.

C. Random Forest Model

The random forest classifier is developed using the 2,262,300 labels, segmented into benign and malware instances like the gradient boosting classifier. The training process resulted in a precision of 99.994% for benign cases and 99.889% for malware. Further, there is a recall value of 99.973% for the benign cases and 99.975% for the attacks. The training's accuracy is 99.973%, while the sensitivity is 99.975%. The specificity for the training process is 99.973%, as shown in Figure-7. The trained model is then tested on 565,576 labels, showing an accuracy of 99.833% and a specificity of 88.649%. The trained model demonstrates a precision of 99.930% for benign cases and 99.737% for intrusions. Also, there is a recall value of 99.936% for the harmless instances and 99.892% for the malware, as shown in Figure-8. This phenomenon demonstrates that the random forest model offers practical detection functionalities for intrusions.

ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

	precision	recall	f1-score	support
Benign	0.99994	0.99973	0.99983	1817049
Attack	0.99889	0.99975	0.99932	445251
accuracy			0.99973	2262300
macro avg	0.99941	0.99974	0.99958	2262300
weighted avg	0.99973	0.99973	0.99973	2262300

Sensitivity : 99.97484564885873
Specitivity : 99.97270299259954

Figure-7. Training results for random forest classifier.

	precision	recall	f1-score	support
Benign	0.99930	0.99936	0.99933	454271
Attack	0.99737	0.99714	0.99725	111305
accuracy			0.99892	565576
macro avg	0.99833	0.99825	0.99829	565576
weighted avg	0.99892	0.99892	0.99892	565576

Sensitivity : 99.71429854903194
Specitivity : 99.93550105553733

Figure-8. Testing results for random forest classifier.

D. Deep Neural Networks Model

The deep neural network model is developed using the existing dataset, with the training and testing labels being consistent with the other models. During the training process, the model demonstrates an accuracy score of 88.649%, a sensitivity score of 53.311%, and a specificity score of 97.713%. The model has a precision of 89.519% for benign cases and 85.099% for attacks. Further, it has a recall value of 97.713% for the benign instances, and 53.312 % for the intrusions, as shown in Figure-9. The testing model has a sensitivity of 53.3048%, a specificity of 97.7227%, and an accuracy of 88.660%. Similarly, it has a precision of 89.519% for benign cases and 85.153% for intrusions. It has a recall scoring of 97.723% for the benign cases and 53.305% for the attacks, as shown in Figure-10. These relatively low scores demonstrate the inefficiency of the model in handling security needs for the NIDS solutions.

	precision	recall	f1-score	support
Benign	0.89519	0.97713	0.93436	1817049
Attack	0.85099	0.53312	0.65556	445251
accuracy			0.88974	2262300
macro avg	0.87309	0.75512	0.79496	2262300
weighted avg	0.88649	0.88974	0.87949	2262300

Sensitivity : 53.31195213486326
Specitivity : 97.71260984156179

Figure-9. Training results for deep neural networks.

	precision	recall	f1-score	support
Benign	0.89519	0.97723	0.93441	454271
Attack	0.85153	0.53305	0.65566	111305
accuracy			0.88981	565576
macro avg	0.87336	0.75514	0.79504	565576
weighted avg	0.88660	0.88981	0.87955	565576

Sensitivity : 53.304882979201295
Specitivity : 97.72272498134373

Figure-10. Testing results for deep neural networks.

E. Comparative Assessment of the Models

A comparative review of the three models demonstrates the substantial performance of the random forest classifier. This model outperforms deep neural network and gradient boosting algorithms in most scores and equalizes in few ones. For instance, it outperforms gradient boosting and deep neural networks in the training process's precision, recall, and sensitivity parameters, as shown in Figure-11. This phenomenon is replicated in the testing process, where it excels in precision, recall, and sensitivity, as shown in Figure-12. The deep neural network performs the least in training and testing, demonstrating its weak functionality for the NIDS solutions. A review of the f1-scores for the three models shows the substantial performance of the random forest in the attack segments for the training and testing iterations, as shown in Figure-13. The model outperforms the deep neural networks and gradient-boosting algorithms in these considerations, demonstrating its comparative efficiency in handling novel incidents in NIDS scenarios.

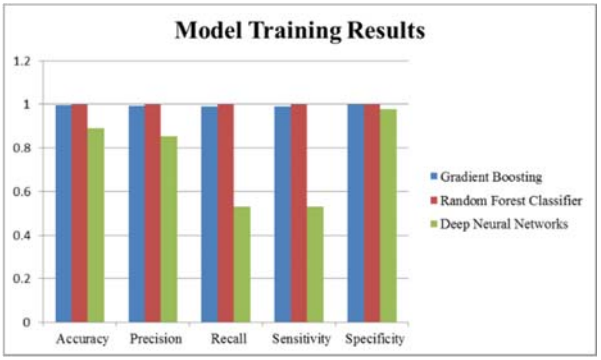

Figure-11. Model classification metrics results for training.

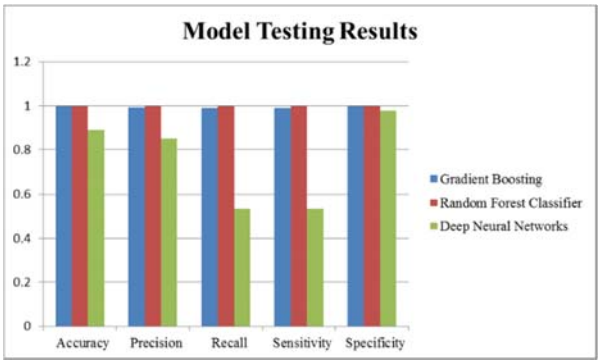

Figure-12. Model classification metrics results for testing.

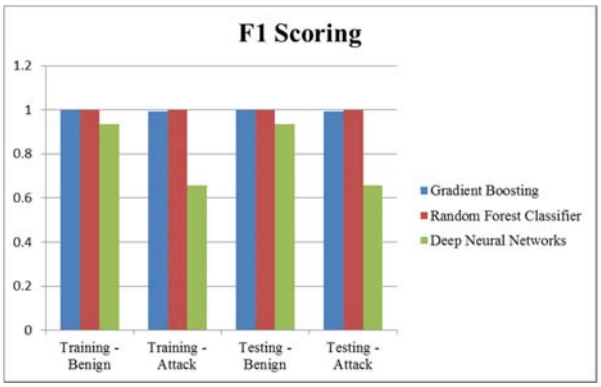

Figure-13. A comparison of F1 scores.

CONCLUSIONS

Although gradient boosting, random forest, and deep neural network models significantly perform in identifying intrusions, the random forest classifier outperforms the others. The experimental results indicate that the random forest classifier excels in the recall, sensitivity, and precision parameters for the training and testing instances. Further, it provides a higher f1-score value than the others, demonstrating its higher accuracy in handling threats. These findings indicate the practical relevance of the model in identifying novel intrusions for enterprise systems than the other models. Further, it shows the model's efficiency in handling the security needs of the NIDS solutions. However, there is a need for additional training iterations with other datasets. Such training initiatives provide an extended assessment environment that enhances understanding of these models, expanding the comparative review of the algorithms. Nevertheless, the current experimental results show that the random forest algorithm offers the best detection engine for network-based intrusion detection systems than deep neural networks and gradient boosting.

REFERENCES

[1] M. Hawedi, C. Talhi, and H. Boucheneb. 2018. Security as a service for public cloud tenants (SaaS). *Procedia computer science*. 130: 1025-1030.

[2] A. Khraisat, I. Gondal, P. Vamplew and J. Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets, and challenges. *Cybersecurity*. 2(1): 1-22.

[3] R. Vinayakumar et al. 2019. Deep learning approach for intelligent intrusion detection system. *IEEE Access* 7: 41525-41550.

[4] S. A. R. Shah and B. Issac. 2018. Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Generation Computer Systems*. 80: 157-170.

[5] Y. Lee *et al*. 2019. Retrieval of total precipitable water from Himawari-8 AHI data: A comparison of random forest, extreme gradient boosting, and deep neural network. *Remote Sensing*. 11(15): 1-18.

[6] S. Mahdavifar, *et al*. 2020. CICMalDroid. [Online]. Available: https://www.unb.ca/cic/datasets/maldroid-2020.html, accessed on: June 5, 2023.

[7] S. Mahdavifar *et al*. 2020. Dynamic android malware category classification using semi-supervised deep learning. in *Proc. DASC 2020*. pp. 515-522.

[8] Lin and W. Shen. 2018. Research on convolutional neural network based on improved Relu piecewise activation function. Procedia Computer Science. 131: 977-984.

[9] S. Raschka, J. Patterson and C. Nolet. 2020. Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information. 11(4): 1-44.

[10] J. Guo, Y. Fan, X. Ji, and X. Cheng. 2109. Matchzoo: A learning, practicing, and developing system for neural text matching. in. Proc. ACM SIGIR-RDIR 2019, pp. 1297-1300.

[11] M. S. ElSayed, N. A. Le-Khac, M. A. Albahar and A. Jurcut. 2021. A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique. Journal of Network and Computer Applications. 191: 1-18.

[12] T. Chandak, S. Shukla and R. Wadhvani. 2019. An analysis of 'A feature reduced intrusion detection system using ANN classifier' by Akashdeep et al. expert systems with applications (2017). Expert Systems with Applications. 130: 79-83.

[13] A. Iqbal and S. Aftab. 2019. A feed-forward and pattern recognition ANN model for network intrusion detection. International Journal of Computer Network and Information Security. 11(4): 19-25.

[14] S. Rajagopal, K. S. Hareesha and P. P. Kundapur. 2020. Performance analysis of binary and multiclass models using azure machine learning. International Journal of Electrical & Computer Engineering. 10(1): 978-986.

[15] A. Ahmim *et al*. 2019. A detailed analysis of using supervised machine learning for intrusion detection. in Proc. ICSIMAT 2019, 2019, pp. 629-639.

[16] B. A. Tama and K. H. Rhee, "An in-depth experimental study of anomaly detection using gradient boosted machine. Neural Computing and Applications, vol. 31, pp. 955-965, Apr. 2019.

[17] S. Naseer *et al*. 2018. Enhanced network anomaly detection based on deep neural networks. IEEE Access. 6: 48231-48246.

[18] D. Upadhyay, J. Manero, M. Zaman, and S. Sampalli. 2020. Gradient boosting machine, selection with machine learning classifiers for intrusion detection on power grids. IEEE Transactions on Network and Service Management. 18(1): 1104-1116.

Appendix: Python Implementation Code

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv("IDS-Android-DS.csv")
```

```
df
```

	Dest	chdir	chmod	dup	exit_u	exit_group	fchmod	fdatasync	flock	Adware	...	unlink	utimes	vAdware	exit	SMS malware.1	SMS malware.2	t
0	54866	3	2	0	12	0	8	8	8.0	0.00000	...	20	0.0	0.0	0	0	0.0	
1	55054	109	1	1	6	6	6	6	6.0	0.00000	...	20	0.0	0.0	0	0	0.0	
2	55056	52	1	1	6	6	6	6	6.0	0.00000	...	20	0.0	0.0	0	0	0.0	
3	46236	34	1	1	6	6	6	6	6.0	0.00000	...	20	0.0	0.0	0	0	0.0	
4	54863	3	2	0	12	0	8	8	8.0	0.00000	...	20	0.0	0.0	0	0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2830738	53	32215	4	2	112	152	28	28	28.0	0.00000	...	20	0.0	0.0	0	0	0.0	
2830739	53	324	2	2	84	362	42	42	42.0	0.00000	...	20	0.0	0.0	0	0	0.0	
2830740	58030	82	2	1	31	6	31	0	15.5	21.92031	...	32	0.0	0.0	0	0	0.0	
2830741	53	1048835	6	2	192	256	32	32	32.0	0.00000	...	20	0.0	0.0	0	0	0.0	
2830742	53	94939	4	2	188	226	47	47	47.0	0.00000	...	20	0.0	0.0	0	0	0.0	

2830743 rows × 79 columns

```
df.isnull().sum()
```

```
Dest            0
chdir           0
chmod           0
dup             0
exit_u          0
               ..
SMS malware.2   0
bind            0
brk             0
Idle Min        0
Label           0
Length: 79, dtype: int64
```

```
len(df)
```

```
2830743
```

```
cols = df.columns
num_cols = df._get_numeric_data().columns
list(set(cols) - set(num_cols))
```

```
[' Label']
```

```
df[" Label"].value_counts()
```

```
BENIGN             2273097
Banking malware     240697
Riskware            158930
Adware              128095
SMS malware          29924
Name:  Label, dtype: int64
```

```
df["Target"] = df[" Label"].apply(lambda x : 0 if x == "BENIGN" else 1)
```

```
df["Target"].value_counts()
```

```
0    2273097
1     557646
Name: Target, dtype: int64
```

```
df  = df.drop([" Label"],axis=1)
```

```
import numpy as np
def clean_dataset(df):
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

df = clean_dataset(df)
```

```
y = df["Target"].values
df = df.drop(["Target"],axis=1)
```

```
df
```

	Dest	chdir	chmod	dup	exit_u	exit_group	fchmod	fdatasync	flock	fork	...	ugetrlimit	uname	unlink	utimes	vfork	arm_nr_ca
0	54866.0	3.0	2.0	0.0	12.0	0.0	8.0	8.0	8.0	0.00000	...	-1.0	1.0	20.0	0.0	0.0	
1	55054.0	109.0	1.0	1.0	6.0	6.0	6.0	6.0	6.0	0.00000	...	256.0	0.0	20.0	0.0	0.0	
2	55056.0	52.0	1.0	1.0	6.0	6.0	6.0	6.0	6.0	0.00000	...	256.0	0.0	20.0	0.0	0.0	
3	46236.0	34.0	1.0	1.0	6.0	6.0	6.0	6.0	6.0	0.00000	...	329.0	0.0	20.0	0.0	0.0	
4	54863.0	3.0	2.0	0.0	12.0	0.0	8.0	8.0	8.0	0.00000	...	-1.0	1.0	20.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2830738	53.0	32215.0	4.0	2.0	112.0	152.0	28.0	28.0	28.0	0.00000	...	-1.0	3.0	20.0	0.0	0.0	
2830739	53.0	324.0	2.0	2.0	84.0	362.0	42.0	42.0	42.0	0.00000	...	-1.0	1.0	20.0	0.0	0.0	
2830740	58030.0	82.0	2.0	1.0	31.0	6.0	31.0	0.0	15.5	21.92031	...	0.0	0.0	32.0	0.0	0.0	
2830741	53.0	1048835.0	6.0	2.0	192.0	256.0	32.0	32.0	32.0	0.00000	...	-1.0	0.0	20.0	0.0	0.0	
2830742	53.0	94939.0	4.0	2.0	188.0	226.0	47.0	47.0	47.0	0.00000	...	-1.0	3.0	20.0	0.0	0.0	

2827876 rows × 77 columns

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(df,y,test_size=0.2,random_state=1)
```

```
from sklearn.metrics import confusion_matrix
def specandsens(y_true,y_pred):
    cm = confusion_matrix(y_true, y_pred)
    sensitivity = (cm[1][1] / (cm[1][1] + cm[1][0]))*100
    specificity = (cm[0][0] / (cm[0][0] + cm[0][1]))*100
    return sensitivity ,specificity
```

Gradinet Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
clf= GradientBoostingClassifier(random_state=1)
clf.fit(X_train,y_train)
```

```
GradientBoostingClassifier(random_state=1)
```

ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

[31]:
```
pred = clf.predict(X_train)
print(classification_report(y_train, pred, digits=5,target_names=["Benign","Attack"]))
sensitivity ,specificity = specandsens(y_train,pred)
print("Sensitivity : ",sensitivity)
print("Specitivity : ",specificity)
```

```
              precision    recall  f1-score   support

      Benign    0.99774   0.99835   0.99805   1817049
      Attack    0.99326   0.99077   0.99201    445251

    accuracy                        0.99686   2262300
   macro avg    0.99550   0.99456   0.99503   2262300
weighted avg    0.99686   0.99686   0.99686   2262300

Sensitivity :  99.07670055766297
Specitivity :  99.83522733839318
```

[32]:
```
pred = clf.predict(X_test)
print(classification_report(y_test, pred, digits=5,target_names=["Benign","Attack"]))

sensitivity ,specificity = specandsens(y_test,pred)
print("Sensitivity : ",sensitivity)
print("Specitivity : ",specificity)
```

```
              precision    recall  f1-score   support

      Benign    0.99766   0.99836   0.99801    454271
      Attack    0.99330   0.99045   0.99187    111305

    accuracy                        0.99681    565576
   macro avg    0.99548   0.99441   0.99494    565576
weighted avg    0.99680   0.99681   0.99680    565576

Sensitivity :  99.044966533399922
Specitivity :  99.83622111030064
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf= RandomForestClassifier(n_estimators=20)
clf.fit(X_train,y_train)
```

43]:
```
RandomForestClassifier(n_estimators=20)
```

```
pred = clf.predict(X_train)
print(classification_report(y_train, pred, digits=5,target_names=["Benign","Attack"]))
sensitivity ,specificity = specandsens(y_train,pred)
print("Sensitivity : ",sensitivity)
print("Specitivity : ",specificity)
```

```
              precision    recall  f1-score   support

      Benign    0.99994   0.99973   0.99983   1817049
      Attack    0.99889   0.99975   0.99932    445251

    accuracy                        0.99973   2262300
   macro avg    0.99941   0.99974   0.99958   2262300
weighted avg    0.99973   0.99973   0.99973   2262300

Sensitivity :  99.97484564885873
Specitivity :  99.97270299259954
```

```
pred = clf.predict(X_test)
print(classification_report(y_test, pred, digits=5,target_names=["Benign","Attack"]))

sensitivity ,specificity = specandsens(y_test,pred)
print("Sensitivity : ",sensitivity)

print("Specitivity : ",specificity)
```

```
              precision    recall  f1-score   support

      Benign    0.99930   0.99936   0.99933    454271
      Attack    0.99737   0.99714   0.99725    111305

    accuracy                        0.99892    565576
   macro avg    0.99833   0.99825   0.99829    565576
weighted avg    0.99892   0.99892   0.99892    565576

Sensitivity :  99.71429854903194
Specitivity :  99.93550105553733
```

Deep Neural Networks

[46]:
```
from sklearn.neural_network import MLPClassifier
```

[47]:
```
clf= MLPClassifier(random_state=1,hidden_layer_sizes=[64,32])
clf.fit(X_train,y_train)
```

Out[47]:
```
MLPClassifier(hidden_layer_sizes=[64, 32], random_state=1)
```

[48]:
```
pred = clf.predict(X_train)
print(classification_report(y_train, pred,digits=5, target_names=["Benign","Attack"]))
sensitivity ,specificity = specandsens(y_train,pred)
print("Sensitivity : ",sensitivity)
print("Specitivity : ",specificity)
```

```
              precision    recall  f1-score   support

      Benign    0.89519   0.97713   0.93436   1817049
      Attack    0.85099   0.53312   0.65556    445251

    accuracy                        0.88974   2262300
   macro avg    0.87309   0.75512   0.79496   2262300
weighted avg    0.88649   0.88974   0.87949   2262300

Sensitivity :  53.311952134486326
Specitivity :  97.71260994156179
```

[49]:
```
pred = clf.predict(X_test)
print(classification_report(y_test, pred,digits=5, target_names=["Benign","Attack"]))

sensitivity ,specificity = specandsens(y_test,pred)
print("Sensitivity : ",sensitivity)

print("Specitivity : ",specificity)
```

```
              precision    recall  f1-score   support

      Benign    0.89519   0.97723   0.93441    454271
      Attack    0.85153   0.53305   0.65568    111305

    accuracy                        0.88981    565576
   macro avg    0.87336   0.75514   0.79504    565576
weighted avg    0.88660   0.88981   0.87955    565576

Sensitivity :  53.304882979201295
Specitivity :  97.72272498134373
```