Journal of Autonomous Intelligence (2024) Volume 7 Issue 3
doi: 10.32629/jai.v7i3.1081

ORIGINAL RESEARCH ARTICLE

Network intrusion detection using ensemble weighted voting classifier based honeypot framework

Parvathi Pothumani*, E Sreenivasa Reddy

Department of Computer Science and Engineering, ANU College of Engineering, Acharya Nagarjuna University, Andhra Pradesh 522510, India

*** Corresponding author:** Parvathi Pothumani, parvathi047@gmail.com

ABSTRACT

The Internet of Things (IoT) is a new model that connects physical objects and the Internet and has become one of the most important technological developments in computing. It is estimated that by 2022, one trillion physical objects will be connected to the Internet. The poor accessibility and lack of interoperability of many of these devices in a vast heterogeneous landscape make it difficult to design specific security measures and implement specific defences mechanism in addition, IoT networks are still open and vulnerable to network disruption attacks. Therefore, there is a need for additional security tools related to IoT. Intrusion Detection System could serve this purpose. Intrusion detection is the process of monitoring and analyzing network traffic in order to detect potential security breaches and unauthorized access to a IOT network. It involves the use of various technologies and techniques to identify and respond to potential threats in real-time. Network intrusion detection helps organizations protect their valuable assets, including sensitive data, intellectual property, and financial resources, from cyberattacks. By detecting and responding to potential security breaches in a timely manner, network intrusion detection systems can help organizations prevent or mitigate the impact of security incidents, minimize downtime and financial losses, and maintain the integrity of their operations and reputation. Weighted soft voting is a technique used in network intrusion detection to improve the accuracy and reliability of the detection process. It involves combining the results of multiple intrusion detection systems (IDS) based on decision tree, random forest and XGBoost using a weighted approach that assigns different levels of importance to each system based on its performance and reliability. The basic idea behind weighted soft voting is to give more weight to the predictions of IDS that have higher accuracy and lower false positive rates, and less weight to those that have lower accuracy and higher false positive rates. The proposed approach can help reduce the impact of false alarms and increase the sensitivity and specificity of the intrusion detection process.

Keywords: intrusion detection systems; weighted soft voting; decision tree; random forest; XGBoost

ARTICLE INFO

Received: 1 August 2023
Accepted: 18 October 2023
Available online: 8 January 2024

1. Introduction

Honeypots are security mechanisms that are meant to detect, divert, or prevent efforts to make unauthorized use of information systems. Honeypots are also known as baiting devices. A honeypot is often a computer, a network, or a piece of data that gives off the appearance of being part of a susceptible system, but which is in fact isolated and monitored by professionals in the field of information security[1]. Honeypots are designed with the purpose of luring attackers to a computer system that is not currently being used. This allows the attackers to be observed and their methods to be scrutinized. This knowledge may subsequently be put to use to strengthen security measures, as well as to devise improved methods for detecting attacks and responding to them. Honeypots may be

used for the purposes of identifying new threats and vulnerabilities, as well as gathering intelligence on attackers and the tactics they employ[2]. They can also be used to trick attackers into thinking they have successfully compromised a real system when, in reality, they are interacting with a fake system that is designed to collect information about them. This can be accomplished by using a combination of social engineering and technical trickery.

A network intrusion detection system, often known as a NIDS, is a kind of security system that examines the data flow on a network for indications of inappropriate usage, illegal access, or hostile behavior. The primary objective of a network intrusion detection system (NIDS) is to detect and notify security workers of possible threats in real time[3]. A network intrusion detection system (NIDS) is often made up of several hardware and software components that, when combined, function to monitor and analyze network traffic[4]. These components may include sensors or agents that record and analyze network traffic, a central management console that receives alerts and offers centralized control, a database or repository that saves and analyses security data, and a centralized management console that receives alerts.

The Network Intrusion Detection System (NIDS) employs a wide range of methods, including as signature-based detection, anomaly-based detection, and behavior-based detection, to identify potentially harmful or suspicious activities on the network[5]. The process of matching network traffic to known patterns or signatures of recognized dangers is what is involved in signature-based detection. A method of network security known as anomaly-based detection searches for patterns of activity on the network that depart from what is considered normal or anticipated behavior. Analysis of user and network behavior is performed under the guise of behavior-based detection in order to spot anomalous or suspicious behavior[6].

Honeypots are often used as a component of network intrusion detection systems (NIDS), which are designed to identify and react to assaults on networks. Honeypots are designed to imitate genuine systems, services, and vulnerabilities on a network before being put there[7]. They are intended to entice attackers and serve as targets for such attacks. As attackers engage with the honeypot, they leave behind traces of their activity. These traces might include logs, network traffic, and modifications to the configuration settings of the system. Security professionals are able to examine these traces to acquire insight into the tactics, motivations, and tools used by the attackers. On the basis of this information, security professionals are able to create or update intrusion detection rules, signatures, and other steps to safeguard the actual systems that are connected to the network. Honeypots have a number of uses, one of which is to trick attackers into thinking they have successfully breached a genuine system while, in reality, they are dealing with a dummy system. This may be used to divert their attention and confuse them.

Honeypots are an effective method for detecting and warding off intrusions into IOT networks. Honeypots are a kind of decoy device that are meant to give the impression of being weak in order to draw in attackers. When an attacker interacts with the honeypot, their activities are recorded and then analyzed so that security flaws in the system may be identified and mitigated[8]. This knowledge may afterwards be put to use to strengthen the defense of the actual systems that are connected to the network.

Honeypots have a variety of applications for preventing intrusions into networks, including the following:

1) Early detection of attacks: The use of honeypots allows for the detection of assaults prior to their arrival at key systems. Administrators are able to monitor incoming traffic and identify unusual behaviour on the network if they place honeypots at key spots on the network.
2) Improved incident response: When an assault is discovered, the information that was acquired from the honeypot may be utilised to formulate a response that is more effective to the attack. The data may be used to determine the strategies and tools utilised by the attacker, which in turn can assist the security team in the development of countermeasures and the prevention of future attacks.

3) Deception and distraction: Honeypots are a useful tool for diverting the attention of potential attackers and convincing them to waste their time and resources on a fake system. This may be especially helpful in deterring targeted assaults, in which the perpetrators may have a very strong desire to break into a particular system.
4) Vulnerability testing: In order to evaluate the efficacy of the security measures implemented throughout the network, honeypots may also be deployed. Monitoring the activities of attackers allows security teams to locate vulnerabilities in their defences and determine what needs to be done to resolve such vulnerabilities.

2. Literature

Alhajjar et al.[9] discussed the adversarial aspect of the challenge faced by network intrusion detection systems (NIDS). The primary emphasis is on the assault viewpoint, which encompasses methods for producing adversarial instances that are able to circumvent a wide range of machine learning models. To be more explicit, the authors investigate the use of evolutionary computation (particularly, particle swarm optimization and genetic algorithm) and deep learning (especially, generative adversarial networks) as instruments for the development of adversarial examples. In order to evaluate the efficacy of these algorithms in avoiding a NIDS, they apply them to two data sets that are open to the public. These data sets are the NSL-KDD and the UNSW-NB15. Additionally, they compare the efficacy of these algorithms to the performance of a baseline perturbation method known as Monte Carlo simulation.

Dina and Manivannan[10] provided an in-depth analysis and critical review of machine learning (ML)-based intrusion detection methods that have been published in academic papers during the last 10 years. For researchers that are working on ML-based intrusion detection systems, this survey would serve as a complement to existing broad surveys on intrusion detection as well as a reference to current work done in the field.

Bangui et al.[11] proposed a brand new machine learning model that makes use of Random Forest and a posterior detection based on coresets as a means of enhancing the performance of intrusion detection systems (IDSs). This model aims to both improve the accuracy of intrusion detection and boost its overall efficiency.

da Costa et al.[12] focused on exhaustive research into the most recent state of the art literature pertaining to Machine Learning Methods and their applications in Internet-of-Things and Intrusion Detection for the purpose of protecting IOT networks. So, the purpose of this study is to do research that is both contemporary and in-depth on important works that deal with several intelligent approaches and their applicable intrusion detection architectures in computer networks, with a focus on the Internet of Things and machine learning. More than 95 publications on the subject were examined for this study, and those works covered a wide variety of topics relating to security concerns in IoT systems.

El Kamel et al.[13] demonstrates a technique to the prediction of network intrusion warnings that is based on deep learning. It has been shown that a deep learning model based on the Gated Recurrent Unit (GRU) is capable of learning dependencies in security warning sequences and producing probable future alerts when given a history of alerts from an attacking source. The model has been presented.

Guarascio et al.[14] suggested the creation of a platform for Orchestrated Information Sharing and Awareness, which would make it possible for various threat detection systems and other information awareness components to work together. ORISHA is supported by a decentralised Threat Intelligence Platform that is built on a network of linked Malware Information Sharing Platform instances. This configuration allows ORISHA to communicate with a variety of Threat Detection layers that are operated by a variety of companies. Threat Detection Systems within this ecosystem benefit from one another by exchanging information, which enables them to improve the prediction accuracy of their underlying models.

Doubtful situations, also known as examples with low anomaly scores, are presented to the knowledgeable individual who plays the role of the oracle in an Active Learning system. With its integration with a honeynet, ORISHA makes it possible to bolster the knowledge base with more examples of successful attacks, which in turn leads to the production of accurate detection models.

Danilov et al.[15] provided investigations with the intention of doing an analysis of the ways for creating synthetic data in order to fill honeypot systems. In the context of honeypot systems, the relevant target objects are divulged, which is followed by the selection of the produced data kinds. The currently available means of production are being looked at. Techniques for assessing the quality of the data produced by honeypot systems are also studied in this study.

Shahid et al.[16] provided a comprehensive online deception system with high levels of user engagement. This system is supported by a hybrid attack detection module, which is made up of a deep learning-based classifier mixed with a cookie analysis engine that assists in the profiling of potential attackers. Malicious HTTP (Hypertext Transfer Protocol) requests are sent from the detection module to a dockers-based deception system, which is managed and controlled by a docker controller. The containerized strategy that has been suggested makes the system more efficient, cuts down on latency, and improves runtime development. In addition to offering effective session management and scenario-based emulation, the essential characteristic of attacker profiling enables the proposed system to cope with attackers carrying zero-day attack payloads. This is a significant advantage. When evaluated in a real-time context, the suggested deception system has a high level of attacker involvement and can protect against all main types of attacks against web applications. In addition, the framework that has been suggested is scalable, flexible, and allows for simple framework update, which makes it appropriate for use even in IoT (Internet of Things) networks.

Lampe and Meng[17] presented an in-depth analysis of the various IDSs that are based on deep learning and are used in automotive networks. We compile a number of different deep learning strategies, classify them according to the topologies and methods that they use, and emphasise the unique contributions that each one makes. In addition to this, we investigate the assessment of every method with regard to the datasets, attack kinds, and metrics.

Kilincer et al.[18] examined the literature studies that make use of the CSE-CIC IDS-2018, UNSW-NB15, ISCX-2012, NSL-KDD, and CIDDS-001 data sets. These data sets are utilised extensively in the development of IDS systems. In addition, max-min normalisation was carried out on these data sets, and classification was carried out using support vector machine (SVM), K-Nearest neighbour (KNN), and Decision Tree (DT) algorithms. These are examples of traditional techniques to machine learning.

Tang et al.[19] created a novel dynamic security defence system by combining a TCP REPAIR-based dynamic honeypot selection architecture with a deep learning-based intelligent firewall. This should be your starting point. Encrypted or unencrypted attack traffic, as well as its many permutations, are distributed across the intelligent firewall in an exact manner by the system. The benign traffic is sent to the target system, while the traffic that has been designated as malicious is used to dynamically choose honeypots that will reply in accordance with the attack procedure.

Srinivasan and P[20] proposed an Ensemble Classifier Algorithm with Stacking Process (ECASP) to choose the best characteristics provided as input to the machine learning classifiers to evaluate the performance of botnet detection.

Matheen and Sundar[21] In order to transfer knowledge from source domain data to target domain and integrate it, researchers propose a deep transfer learning-based Lecun network P-Lenet technique in this work. This will result in an effective intrusion detection system (IDS) that will improve detection accuracy for any wireless multimedia sensors networks (WMSN). Anomaly detection was emphasised as the major

mechanism in the suggested solution to avoid attacks on the software defined network (SDN) platform in real time. In this article, the Lecun network (LeNet) was examined, and a new version of the network known as the Lenet was suggested.

Here Literature which was taken mainly focuses on the Detecting Intrusion with Machine Learning which classifies the action as Normal or as Intrusion when compared with deep learning as it is a part of machine learning and also research which was taken contributes Security of Internet of things.

3. Proposed ensemble voting classifier

An Ensemble Voting Classifier is a method of machine learning that makes a prediction by combining the results of many separate models into a single composite model. When you have multiple models that are trained on different aspects of the same data, or when you want to combine the strengths of multiple models to improve the overall accuracy of your predictions, this technique is particularly helpful. It is also useful when you have multiple models that are trained on different data sets. An Ensemble Voting Classifier works by having each individual model make a prediction based on the input data, and then combining those predictions to arrive at a final result for the classification. Combining the results of many models may be done in a number of different methods, including the following:

- Majority voting: The final forecast in this method is derived from the different models' predictions that are the most similar to one another. This method is also known as majority voting. For instance, if three models predict class A and two models predict class B, then the final forecast would be class A. This would be the case if three models predicted class A and two models predicted class B.
- Weighted voting: Weighted Voting is a methodology in which each individual model is given a weight depending on its performance on the training data, and the final forecast is a weighted average of the individual predictions. This method is also referred to as “voting with your feet”.

Figure 1 shows the example of voting classifier. Ensemble Voting Classifiers are adaptable and may be used for a broad variety of tasks related to machine learning, including as classification and regression. They are especially useful in situations in which the numerous models each have distinct advantages and disadvantages, or in which there is a high level of unpredictability in the data.

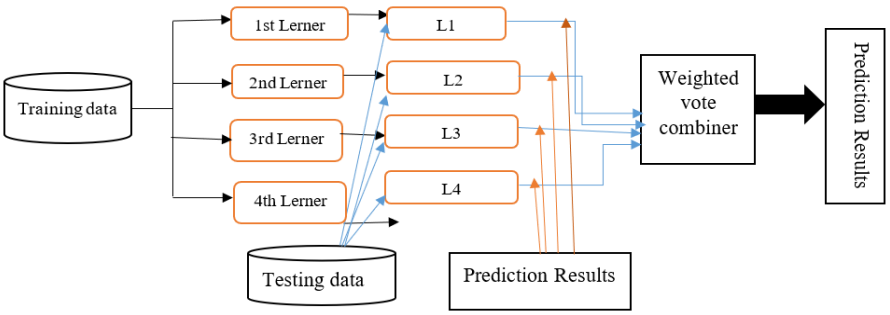

Figure 1. Voting classifier.

3.1. Ensemble weighted voting classifier

A weighted voting classifier is an ensemble technique used in machine learning that combines the predictions of multiple models to make a final prediction. Unlike simple majority voting where each model has an equal say in the final prediction. i.e., assigning an instance to the class that most base classifiers agree on.

In this type of voting, all classifiers have the same value of votes and they are all equal to 1. Let n be the number of classifiers in the ensemble. Assume that C_i is used to represent a classifier in the ensemble E such that i = 1, 2, ..., n and ensemble E = {C1, C2, ..., Cn}.

The decision of the *i*th classifier (C_i) is denoted by $d_{i,j} \in \{0, 1\}$, where $j = 1, 2, ..., k$ and k is the number of classes. The decision will produce $d_{i,j} = 1$, if ith classifier decides for class c_j, and $d_{i,j} = 0$ otherwise. The output of the ensemble, in majority voting, can be outlined with the following Equation (1).

$$\max_{i \le j \le k} \sum_{i=1}^{n} di, j \tag{1}$$

However, the base classifiers in an ensemble typically cannot perform equally well, therefore it may not be best to aggregate them equally. In this situation, weighing each classifier according to its performance is the best solution. How to properly estimate the weights of classifiers, which may significantly affect the performance of the ensemble, is the most crucial topic in weighting systems. A unique weighing technique is suggested in this work.

A weighted voting classifier assigns weights to each model, indicating their relative importance, and computes the weighted average of the model predictions to make the final prediction. The weights of the models are determined by the performance of the model on the training data. Typically, the better the model performs on the training data, the higher the weight assigned to it in the ensemble. The weights of the models can be set manually or can be learned from the training data using techniques such as cross-validation.

The process of building a weighted voting classifier involves the following steps:

1) As a result, the entire dataset is divided into training and test sets.
2) Train multiple models on the same training data: Typically, the models used in an ensemble are different from each other in terms of the learning algorithm used, the hyperparameters chosen, or the way the input data is processed. This diversity among the models helps to reduce the risk of overfitting and improves the overall accuracy of the ensemble.
3) Make predictions using each individual model: Once the models are trained, they are used to make predictions on the test data. Assume that there are m instances total in the testing set. **Table 1** displays the classifiers' correct and incorrect predictions for each case in the testing.
4) Assign weights to each model: The weights are assigned to each model based on its performance on the training data. The most common approach to assigning weights is to use the accuracy of each model as a proxy for its performance.
5) Compute the weighted average of the model predictions: The predictions of each model are multiplied by its weight, and the weighted predictions are summed up to compute the final prediction.

The proposed approach (Soft voting classifier) is illustrated with an example. A sample data is given in **Table 1**. In the example senario (**Table 1**), there are three classifiers (C1, C2, C3 and C4) in the ensemble and their predictions on five instances in the validation set are listed. The final column in the table shows the actual class labels of the instances. There two possible class labels: X and Y.

Table 1. An example dataset.

Classes Instances	**Predicted classes**			**Actual class**
	C1 prediction	**C2 prediction**	**C3 prediction**	
1	Y	Y	X	X
2	X	X	Y	Y
3	X	Y	X	Y
4	Y	X	X	Y

Table 2 shows the changes in the weights of the classifiers for each instance. To begin with, all weights are initialized to 1. In each iteration through instances in the testing set, the weights of correctly predicting classifiers are only incremented, so the weights of the others are not modified. The weights are incremented by the ratio of the number of incorrectly predicting classifiers to the whole number of classifiers (n = 3). For

example; there is one incorrect match for the first instance, so the weight of the 3rd classifier is updated by 3/4. This incremental process finishes with the last item.

Table 2. Changes in the weights for each instance.

Classes Instances	Prediction of classes		
	C1 prediction	**C2 prediction**	**C3 prediction**
Initial weights	1.0	1.0	1.0
1	1.0 + 1/3 = 1.33	1.0 + 0 = 1.0	1.0 + 0 = 1.0
2	1.33 + 0 = 1.33	1.0 + 0 = 1.0	1.0 + 1/3 = 1.33
3	1.33 + 0 = 1.33	1.0 + 1/3 = 1.33	1.33 + 0 = 1.33
4	1.33 + 1/3 = 1.66	1.33 + 0 = 1.33	1.33 + 0 = 1.33
Final weights	1.66	1.33	1.33

We have 3 models, A, B, and C, with weights 1.66, 1.33 and 1.33 respectively. If model A predicts class 1, model B predicts class 2, model C predicts class 1 then the final prediction would be:

$$\text{weighted average} = (1.66 \times 1) + (1.33 \times 2) + (1.33 \times 1) = 5.65 \quad (2)$$

if the weighted average is greater than threshold value then the final prediction would be class 1.

Figure 2 shows step_wise classification procedure. The advantages of using a weighted voting classifier include improved accuracy and reduced variance compared to using a single model. By combining the predictions of multiple models, the ensemble is able to capture more information about the data and make more accurate predictions. Additionally, assigning weights to the models helps to reduce the impact of poorly performing models on the final prediction, making the ensemble more robust to noise and outliers in the data.

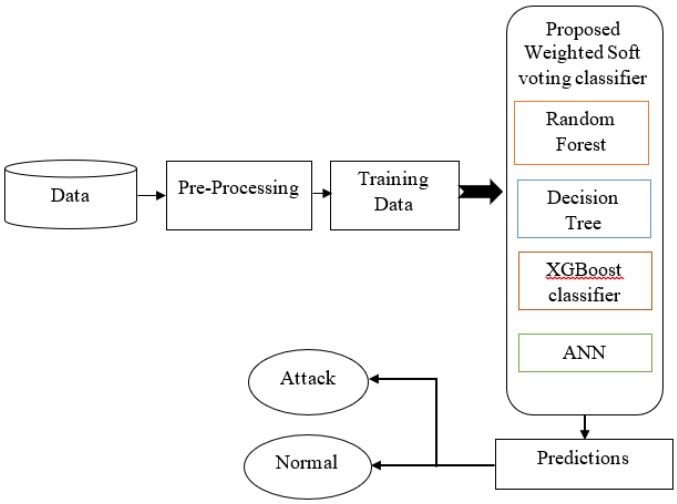

Figure 2. Proposed soft voting classifier method.

3.1.1. Random forest classifier

Random Forest Classifier is a supervised machine learning algorithm used for classification problems. It is an ensemble method that combines multiple decision trees to create a powerful classifier. The steps involved in building a Random Forest Classifier are as follows:

1) Step 1: Data preparation
   - Random Forest Classifier requires labeled data to train the model.
   - Split the labeled data into training and testing sets. Typically, the ratio of the training data to testing data is 70:30, 80:20, or 90:10 depending on the size of the dataset.

- Feature scaling is not required in Random Forest Classifier as it is based on decision trees.

2) Step 2: Building the forest
   - Random Forest Classifier builds a forest of decision trees.
   - The number of trees in the forest is a hyperparameter that needs to be set before training.
   - Each decision tree is trained on a random subset of the training data and a random subset of the features.
   - The random selection of samples and features helps to reduce overfitting and improve the generalization ability of the model.
3) Step 3: Training the decision trees
   - The decision trees in the forest are trained using the training data.
   - At each node of the tree, a split is made based on the value of a feature that maximizes the separation of the data into classes.
   - The process of selecting the best split continues recursively until the leaf nodes are reached.
   - The splitting criterion used in Random Forest Classifier is the Gini impurity, which measures the degree of impurity in a set of samples.
   - The objective of each decision tree is to create partitions in the feature space that are as pure as possible.
4) Step 4: Making predictions
   - Once the forest is built, it can be used to make predictions on the testing data.
   - For each test sample, the class labels predicted by all the decision trees in the forest are collected.
   - The final prediction is made by taking the majority vote of all the decision trees.
   - The class with the highest number of votes is chosen as the predicted class for the test sample.
5) Step 5: Evaluating the model
   - Once the predictions are made, the accuracy of the model can be evaluated using metrics such as accuracy, precision, recall, F1 score, etc.
   - If the model performance is not satisfactory, the hyperparameters can be tuned to improve the accuracy.
   - Common hyperparameters in Random Forest Classifier are the number of trees in the forest, the maximum depth of the trees, the minimum number of samples required to split a node, and the maximum number of features to consider when looking for the best split.

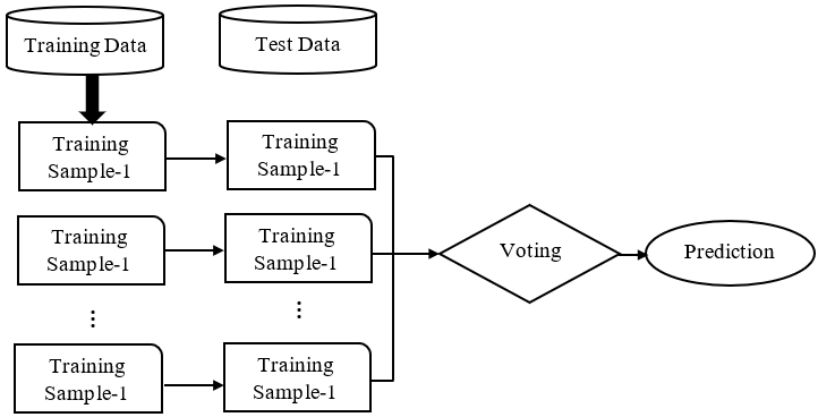

Figure 3. Random forest classifier.

Figure 3 shows algorithm steps for Random Forest Classifier can be summarized as follows:

a) Prepare the labelled data for training and testing.
b) Build the forest by training multiple decision trees on random subsets of the data and features.

c) Train the decision trees using the Gini impurity criterion.
d) Make predictions by taking the majority vote of all the decision trees.
e) Evaluate the accuracy of the model and tune the hyperparameters if necessary.

Random Forest Classifier is a powerful algorithm that is widely used in various applications such as image classification, text classification, and financial analysis. It is known for its high accuracy, robustness, and ability to handle large datasets.

3.1.2. Decision tree

Decision Tree is a supervised machine learning algorithm used for classification and regression problems. It creates a tree-like model of decisions and their possible consequences. The steps involved in building a Decision Tree algorithm are as follows:

1) Step 1: Data preparation
   - Decision Tree algorithm requires labeled data to train the model.
   - Split the labeled data into training and testing sets. Typically, the ratio of the training data to testing data is 70:30, 80:20, or 90:10 depending on the size of the dataset.
   - Feature scaling is not required in Decision Tree algorithm as it is based on the level of impurity in the features.
2) Step 2: Building the tree
   - Decision Tree algorithm builds a tree-like model of decisions and their possible consequences.
   - The root of the tree represents the feature that best splits the data into classes based on a criterion such as entropy or Gini impurity.
   - The decision tree is built recursively by splitting the data into subsets based on the best feature until the leaf nodes are reached.
   - The leaf nodes represent the class labels.
3) Step 3: Training the decision tree
   - The decision tree is trained using the training data.
   - At each node of the tree, a split is made based on the value of a feature that maximizes the separation of the data into classes.
   - The process of selecting the best split continues recursively until the leaf nodes are reached.
   - The splitting criterion used in Decision Tree algorithm can be entropy or Gini impurity, which measures the degree of impurity in a set of samples.
   - The objective of the decision tree is to create partitions in the feature space that are as pure as possible.
4) Step 4: Making predictions
   - Once the decision tree is built, it can be used to make predictions on the testing data.
   - For each test sample, the class label predicted by the decision tree is obtained by following the path from the root to the leaf node.
   - The class label corresponding to the leaf node is chosen as the predicted class for the test sample.
5) Step 5: Evaluating the model
   - Once the predictions are made, the accuracy of the model can be evaluated using metrics such as accuracy, precision, recall, F1 score, etc.
   - If the model performance is not satisfactory, the hyperparameters can be tuned to improve the accuracy.
   - Common hyperparameters in Decision Tree algorithm are the maximum depth of the tree, the minimum number of samples required to split a node, and the splitting criterion (entropy or Gini impurity).

The algorithm steps for Decision Tree can be summarized as follows:

a) Prepare the labeled data for training and testing
b) Build the tree by selecting the best feature that splits the data based on a criterion (entropy or Gini impurity)
c) Train the decision tree by recursively splitting the data into subsets based on the best feature until the leaf nodes are reached
d) Make predictions by following the path from the root to the leaf node for each test sample
e) Evaluate the accuracy of the model and tune the hyperparameters if necessary.

Decision Tree algorithm is a simple yet powerful algorithm that is widely used in various applications such as medical diagnosis, credit scoring, and customer churn prediction. It is known for its interpretability, versatility, and ability to handle both categorical and numerical data.

3.1.3. XGBoost classifier

XGBoost (Extreme Gradient Boosting) is a popular ensemble machine learning algorithm used for classification and regression problems. It is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. The algorithm steps for XGBoost classifier are as follows:

1) Step 1: Data preparation
   - XGBoost classifier requires labeled data to train the model.
   - Split the labeled data into training and testing sets. Typically, the ratio of the training data to testing data is 70:30, 80:20, or 90:10 depending on the size of the dataset.
   - Feature scaling is not required in XGBoost classifier as it is based on the gradient boosting technique.
2) Step 2: Building the initial decision tree
   - XGBoost classifier builds an initial decision tree as the base model.
   - The decision tree is built using the training data by recursively partitioning the data into subsets based on the best feature that minimizes the objective function.
   - The objective function used in XGBoost classifier is the sum of the loss function and the regularization term, which penalizes the complexity of the model to avoid overfitting.
3) Step 3: Boosting the decision tree
   - XGBoost classifier boosts the initial decision tree by adding more decision trees to improve the performance of the model.
   - Each subsequent decision tree is built using the training data by recursively partitioning the data into subsets based on the best feature that minimizes the objective function.
   - The difference between XGBoost and other boosting algorithms is the use of the gradient descent optimization algorithm to update the weights of the samples at each iteration.
4) Step 4: Making predictions
   - Once the XGBoost model is built, it can be used to make predictions on the testing data.
   - For each test sample, the class label predicted by the XGBoost model is obtained by summing the predictions of all the decision trees.
   - The class label corresponding to the highest sum is chosen as the predicted class for the test sample.
5) Step 5: Evaluating the model
   - Once the predictions are made, the accuracy of the XGBoost model can be evaluated using metrics such as accuracy, precision, recall, F1 score, etc.
   - If the model performance is not satisfactory, the hyperparameters can be tuned to improve the accuracy.

- Common hyperparameters in XGBoost classifier are the learning rate, the maximum depth of the decision trees, the number of decision trees, the regularization term, and the subsample ratio.

The algorithm steps for XGBoost classifier can be summarized as follows:

a) Prepare the labeled data for training and testing.
b) Build the initial decision tree by recursively partitioning the data into subsets based on the best feature that minimizes the objective function.
c) Boost the initial decision tree by adding more decision trees using the gradient descent optimization algorithm to update the weights of the samples.
d) Make predictions by summing the predictions of all the decision trees for each test sample.
e) Evaluate the accuracy of the XGBoost model and tune the hyperparameters if necessary.

XGBoost classifier is a powerful algorithm that is widely used in various applications such as speech recognition, natural language processing, and computer vision. It is known for its scalability, accuracy, and ability to handle large and complex datasets.

3.2. Honeypot using weighted voting classifier IDS

Weighted voting classifier IDS is a machine learning model that combines the output of multiple IDS models to improve the accuracy of intrusion detection. Each IDS model is assigned a weight based on its performance, and the final classification is determined by combining the output of each model, weighted by its assigned weight. To implement a honeypot using a weighted voting classifier IDS, the following steps are used:

Set up a honeypot system: Install a vulnerable operating system on a virtual machine or physical server, and configure it to mimic a production system. This can include setting up services, such as a web server, FTP server, and SSH server, and configuring weak or default passwords to attract attackers.

Collect training data: Record network traffic and system logs from the honeypot system when it is being attacked. This data will be used to train the IDS models.

Train IDS models: Train multiple IDS models using the collected training data. Some popular IDS algorithms include Snort, Bro, and Suricata. Each model should be trained on a separate subset of the training data.

Assign weights: Evaluate the performance of each IDS model using validation data, and assign a weight to each model based on its performance. Models with higher accuracy or fewer false positives should be assigned higher weights.

Combine output: When new network traffic is received by the honeypot system, pass it through each IDS model, and combine the output using the assigned weights. If the combined output indicates an attack, alert the administrator.

Evaluate and refine: Periodically evaluate the performance of the IDS models and adjust the weights as necessary. Also, update the training data to ensure that the IDS models remain effective.

4. Experimental results

In **Figure 4**, a honeypot intrusion detection system (IDS) is a security mechanism that is designed to detect unauthorized access to IOT systems or networks. It does this by employing a system or network that gives the impression of being vulnerable but is, in reality, a trap for those who would attempt to gain unauthorized access. In this part, we will show the results of the experimental study that was conducted in order to validate the suggested model.

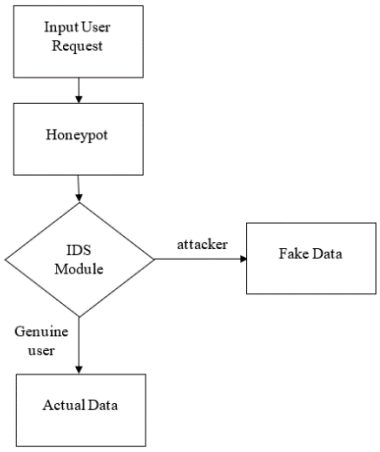

Figure 4. Proposed Honeypot based on weighted voting classifier.

4.1. Dataset

The Intrusion Detection Dataset (UNR-IDD) from the University of Nevada, Reno is used so that experimental study may be carried out. The bulk of the UNR-IDD is composed of the port information that was gathered from various networks. These phrases refer to the observed port metrics that are recorded in the router and switch ports, and they come up in the context of an environment including networking. In addition, the data set includes something that is referred to as "delta port statistics," which illustrates how the total magnitude of observed port statistics changed over the course of a certain time period. These port statistics are able to deliver a fine-grained analysis of network flows because decisions are made at the port level as opposed to the flow level. This is possible because choices are made at the port level. The process of recognizing any potential incursions might be sped significantly as a result of this. This Data set helps to protect network from port scanning, intrusions and identifying abnormal activity happens in a network.

4.2. Evaluation parameters

Table 3 shows the evaluation parameter to know the performance of each model. In the fields of machine learning and data analysis, some performance measures that are often used include accuracy, recall, precision, F1 score, and AUC (Area Under the Curve). The performance of a model in classification tasks, in which the objective is to assign labels (such as "positive" or "negative") to input data, may be evaluated with the use of these metrics by comparing them to the data.

Accuracy: The percentage of a model's right predictions relative to the total number of forecasts produced is what is meant by the term "accuracy". The formula for determining it is (TP+TN)/(TP+TN+FP+FN), where TP represents the number of true positives, TN represents the number of true negatives, FP represents the number of false positives, and FN represents the number of false negatives.

Recall (Sensitivity): The recall of a model is defined as the percentage of true positives that were properly detected among all true positives. This is also referred to as a model's sensitivity. It is determined by dividing the number of true positives by the sum of the number of true positives and false negatives. The formula for this is TP/(TP+FN).

Precision: The precision of a model is defined as the percentage of true positives that were accurately detected among all positive predictions made. This proportion is compared to the total number of positive predictions produced. It is determined by dividing the number of true positives by the total number of positive results, where the number of true positives is TP and the number of false positives is FP

The F1 score of a model is the harmonic mean of its accuracy and recall scores. It is a weighted average that takes into consideration both precision and recall, and it is a measure of precision. It is computed as 2 times (precision times recall) divided by (precision plus recall).

Area Under the Curve, or AUC for short: The area under the curve (AUC) is a statistic that assesses the overall performance of a model based on its capacity to differentiate between positive and negative classes. The area under the ROC (Receiver Operating Characteristic) curve, which displays the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds, is how it is computed. The FPR refers to the rate of incorrect positive results.

Table 4 shows the Comparative analysis between the classifiers like SVM, ANN, KNN, etc. and **Figures 5–9** show the comparative analysis of different evaluation parameters between models.

Table 3. Evaluation parameters.

Accuracy	97.7%
Recall	0.97
Precision	0.97
F1 score	0.97
AUC	0.98

Table 4. Comparative analysis.

	SVM	**ANN**	**KNN**	**Ad boost**	**Gaussian NB**	**Logistic regression**	**Proposed ensemble voting classifier**
Accuracy	49%	59%	85%	61%	67%	74%	97.7%
Recall	0.49	0.59	0.85	0.61	0.67	0.74	0.97
Precision	0.50	0.50	0.85	0.56	0.73	0.77	0.97
F1 score	0.48	0.49	0.85	0.53	0.66	0.73	0.97
AUC	0.80	0.81	0.91	0.85	0.87	0.88	0.98

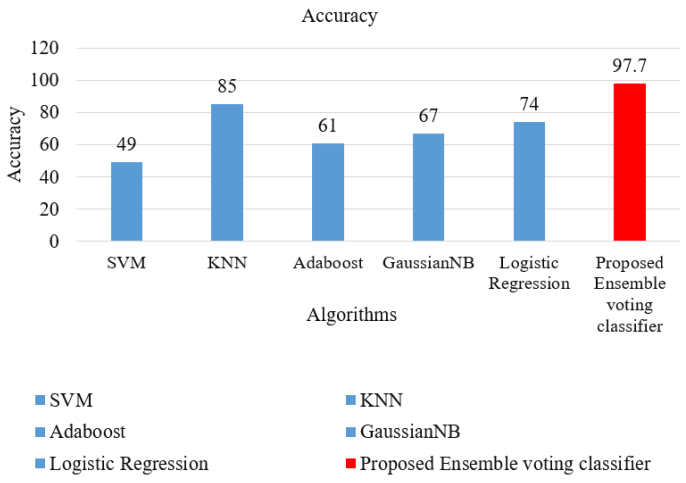

Figure 5. Comparative analysis of accuracy.

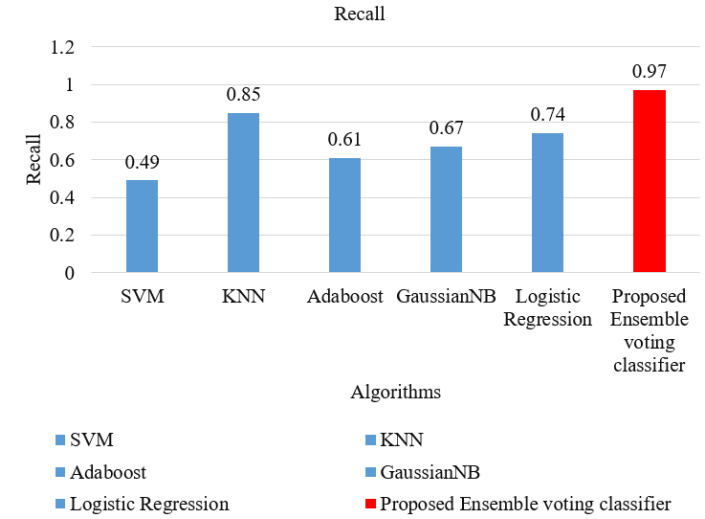

Figure 6. Comparative analysis of recall.

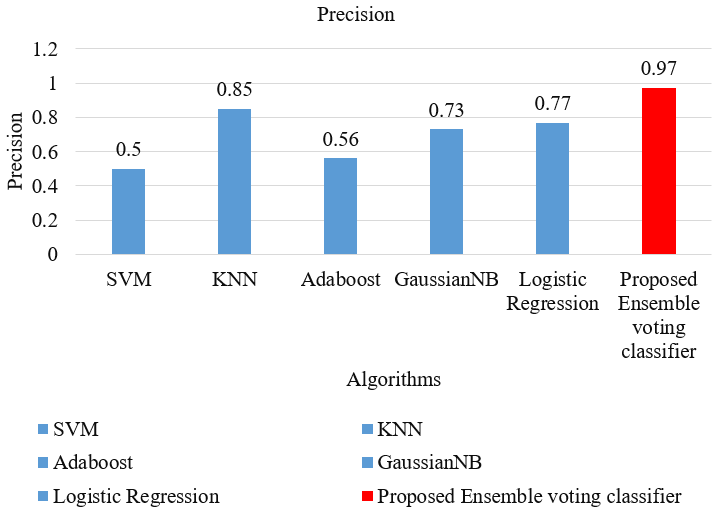

Figure 7. Comparative analysis of precision.

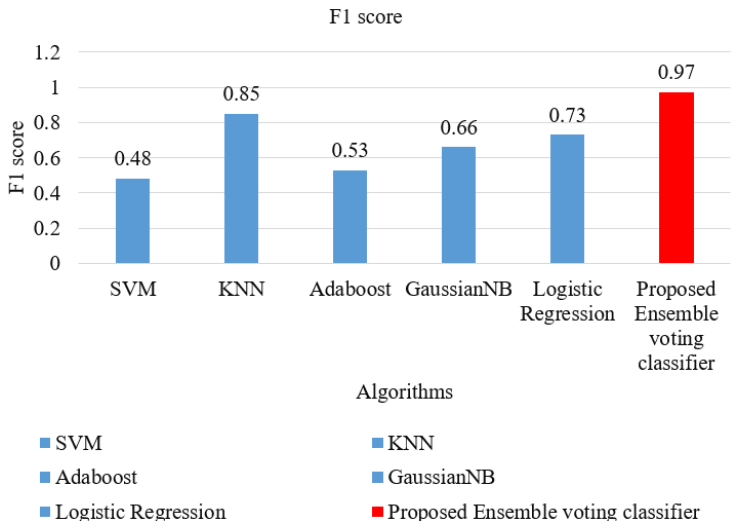

Figure 8. Comparative analysis of F1-score.

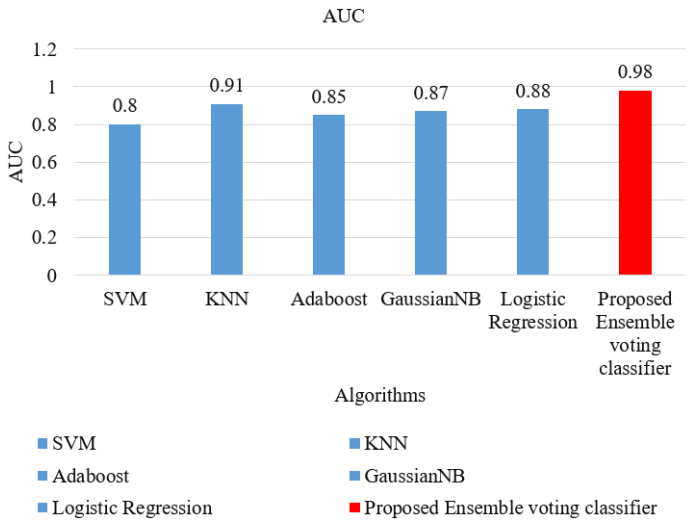

Figure 9. Comparative analysis of AUC.

5. Conclusion

Cowrie: Cowrie is a medium-interaction SSH and Telnet honeypot designed to log brute force and shell interaction attempts. It can be configured to mimic specific SSH and Telnet services. A honeypot can work in conjunction with an Intrusion Detection System (IDS) as its core by providing an additional layer of defense against potential attacks. The IDS monitors network traffic for suspicious activity and alerts security personnel if an attack is detected. The honeypot, on the other hand, acts as a decoy system that is designed to +lure attackers away from real systems and services. When an attacker tries to penetrate the honeypot, the IDS can detect and analyze their actions. Random Forest is a powerful ensemble learning method that combines multiple decision trees to produce highly accurate predictions. It is robust to overfitting, which can occur when a model is too complex and fits the training data too closely. Decision trees are computationally efficient and can handle large datasets with many features. They are also able to perform classification in real-time, making them suitable for applications that require fast response times. XGB is a powerful ensemble learning method that combines multiple decision trees to produce highly accurate predictions. XGB is designed to be highly efficient and scalable, making it suitable for large datasets and real-time applications. The proposed ensemble voting classifier produced superior Accuracy of 95%, recall of 0.94, precision of 0.94, F1 score of 0.94 and AUC of 0.97.

Author contributions

Conceptualization, PP; methodology, PP; validation, ESR; investigation, PP; writing—original draft preparation, PP; writing—review and editing, ESR; supervision, ESR; project administration, PP. All authors have read and agreed to the published version of the manuscript.

Conflict of interest

The authors declare conflict of interest.

References

1. Kumari P, Jain AK. A comprehensive study of DDoS attacks over IoT network and their countermeasures. Computers & Security. 2023, 127: 103096. doi: 10.1016/j.cose.2023.103096
2. Maesschalck S, Giotsas V, Green B, et al. Don't get stung, cover your ICS in honey: How do honeypots fit within industrial control system security. Computers & Security. 2022, 114: 102598. doi: 10.1016/j.cose.2021.102598
3. Khan SU, Eusufzai F, Azharuddin Redwan Md, et al. Artificial Intelligence for Cyber Security: Performance Analysis of Network Intrusion Detection. Explainable Artificial Intelligence for Cyber Security. Published online 2022: 113-139. doi: 10.1007/978-3-030-96630-0_6

4. Heidari A, Jabraeil Jamali MA. Internet of Things intrusion detection systems: A comprehensive review and future directions. Cluster Computing, 2022,11: 1-28.
5. Nazir A, Khan RA. A novel combinatorial optimization based feature selection method for network intrusion detection. Computers & Security. 2021, 102: 102164. doi: 10.1016/j.cose.2020.102164
6. ElSayed MS, Le-Khac NA, Albahar MA, et al. A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique. Journal of Network and Computer Applications. 2021, 191: 103160. doi: 10.1016/j.jnca.2021.103160
7. Mohammadzad M, Karimpour J. Using rootkits hiding techniques to conceal honeypot functionality. Journal of Network and Computer Applications. 2023, 214: 103606. doi: 10.1016/j.jnca.2023.103606
8. Al-Mohannadi H, Awan I, Al Hamar J. Analysis of adversary activities using cloud-based web services to enhance cyber threat intelligence. Service Oriented Computing and Applications. 2020, 14(3): 175-187. doi: 10.1007/s11761-019-00285-7
9. Alhajjar E, Maxwell P, Bastian N. Adversarial machine learning in Network Intrusion Detection Systems. Expert Systems with Applications. 2021, 186: 115782. doi: 10.1016/j.eswa.2021.115782
10. Dina AS, Manivannan D. Intrusion detection based on Machine Learning techniques in computer networks. Internet of Things. 2021, 16: 100462. doi: 10.1016/j.iot.2021.100462
11. Bangui H, Ge M, Buhnova B. A hybrid machine learning model for intrusion detection in VANET. Computing. 2021, 104(3): 503-531. doi: 10.1007/s00607-021-01001-0
12. da Costa KAP, Papa JP, Lisboa CO, et al. Internet of Things: A survey on machine learning-based intrusion detection approaches. Computer Networks. 2019, 151: 147-157. doi: 10.1016/j.comnet.2019.01.023
13. El Kamel N, Eddabbah M, Lmoumen Y, et al. A Real-Time Smart Agent for Network Traffic Profiling and Intrusion Detection Based on Combined Machine Learning Algorithms. Smart Innovation, Systems and Technologies. 2021, 301-309. doi: 10.1007/978-981-16-3637-0_21
14. Guarascio M, Cassavia N, Pisani FS, et al. Boosting Cyber-Threat Intelligence via Collaborative Intrusion Detection. Future Generation Computer Systems. 2022, 135: 30-43. doi: 10.1016/j.future.2022.04.028
15. Danilov VD, Ovasapyan TD, Ivanov DV, et al. Generation of Synthetic Data for Honeypot Systems Using Deep Learning Methods. Automatic Control and Computer Sciences. 2022, 56(8): 916-926. doi: 10.3103/s014641162208003x
16. Shahid WB, Aslam B, Abbas H, et al. A deep learning assisted personalized deception system for countering web application attacks. Journal of Information Security and Applications. 2022, 67: 103169. doi: 10.1016/j.jisa.2022.103169
17. Lampe B, Meng W. A survey of deep learning-based intrusion detection in automotive applications. Expert Systems with Applications. 2023, 221: 119771. doi: 10.1016/j.eswa.2023.119771
18. Kilincer IF, Ertam F, Sengur A. Machine learning methods for cyber security intrusion detection: Datasets and comparative study. Computer Networks. 2021, 188: 107840. doi: 10.1016/j.comnet.2021.107840
19. Tang J, Chen M, Chen H, et al. A new dynamic security defense system based on TCP_REPAIR and deep learning. Journal of Cloud Computing. 2023, 12(1). doi: 10.1186/s13677-022-00379-2
20. Srinivasan S, P D. Enhancing the security in cyber-world by detecting the botnets using ensemble classification based machine learning. Measurement: Sensors. 2023, 25: 100624. doi: 10.1016/j.measen.2022.100624
21. Matheen MA, S Sundar. Mitigation of network security attacks in wireless multimedia sensors networks using intrusion detection system. 2024, 7(1). doi: 10.32629/jai.v7i1.751