International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Enhancing Intrusion Detection Effectiveness through the Implementation of Advanced Machine Learning Boosting Strategies

Pulyala Radhika*[1], Geeta Kakarla[2]

Submitted: 27/01/2024 **Revised**: 05/03/2024 **Accepted**: 13/03/2024

Abstract: In the rapidly evolving digital landscape, the increased utilization of networks has given rise to numerous security challenges. With the integration of the digital world into society, the emergence of new threats such as viruses and worms has become prevalent. Malicious actors employ various techniques, including password cracking and detecting unencrypted text, to exploit vulnerabilities within computer systems. Consequently, users must prioritize security measures to safeguard their systems against unauthorized intrusions. One well-established method for protecting private networks from external threats is the firewall technique. Firewalls serve as a protective barrier by filtering incoming Internet traffic. However, certain access methods, such as connecting to the Intranet via a modem within the private network, can evade detection by conventional firewalls. To address this issue, a novel system known as a Network Intrusion Detection System (IDS) has been developed to effectively identify and mitigate network attacks. In this project, an Intrusion Detection System utilizing Machine Learning has been developed to accurately determine the presence of intrusions. Multiple models have been constructed using sklearn and ensemble techniques, resulting in exceptional accuracy. This system serves as a proactive approach to bolster network security and confront the constantly developing spectrum of cyber-attacks.

Keywords: *Network Intrusion Detection System, IDS, Machine Learning, Cyber Security, Cyber Threats, Ensemble Learning, Firewall Technique, System Vulnerabilities, Network Attacks, Intrusion Detection*

1. Introduction

In today's digital era, the extensive use of computers and the internet has introduced an array of security challenges. The volatile nature of networks in this digital world has given rise to an increasing number of security issues, including the importation of new threats such as viruses and worms. Malignant users exploit various techniques, such as password cracking and the detection of unencrypted text, to target system vulnerabilities. As a result, ensuring the security of computer systems and protecting them from unauthorized intrusions has become imperative.

The firewall technique has emerged as a well-known method for safeguarding private networks from external threats. By filtering incoming traffic, firewalls act as a protective barrier between the private and public networks. However, conventional firewalls have limitations in detecting certain access methods, such as external users connecting to the Intranet through modems installed within private networks. This poses a significant challenge in effectively securing the network infrastructure.

To address these issues, the development of a novel system called a Network Intrusion Detection System (IDS) becomes necessary. Such a system aims to detect and mitigate network attacks by utilizing advanced technologies. In this research project, we have developed an IDS using Machine Learning techniques. This system is designed to accurately identify the presence of intrusions, offering a proactive approach to enhance network security.

Multiple models have been developed, leveraging sklearn and ensemble techniques, to achieve high accuracy in detecting network intrusions. Our technology detects and responds to developing cyber threats more effectively by leveraging the power of Machine Learning. Through this research, we aim to contribute to the field of network security and provide practical solutions for mitigating security risks in the digital landscape.

The following sections of this article will go into the methodology, experimental findings, and discussions, offering an in-depth analysis of the performance of our IDS and its implications for network security.

2. Literature Review

Authors in [1] designed an attention-based deep-learning models, such as Attention-based RNN and Transformer-based architectures, which emerged as powerful techniques for intrusion detection. They leverage attention mechanisms to selectively focus on crucial features and have demonstrated improved performance in accurately identifying network intrusions.

[1]*Assistant Professor, Department of CSE-Cyber Security, Sreenidhi Institute of Science and Technology, Yamnampet, Telangana, India, Corresponding author email id: pulyalaradhika@gmail.com Orcid id: 0000-0002-3991-098X*
[2]*Assistant Professor, Department of CSE-Cyber Security, Sreenidhi Institute of Science and Technology, Yamnampet, Telangana, India, Email ID: geetavemula333@gmail.com Orcid id: 0000-0002-6643-275X*

Using machine learning techniques, the paper [2] provides a unique approach for detecting network intrusions in Software-Defined Networks (SDNs). The authors tackled the growing security concerns in SDNs by exploiting machine learning algorithms' capacity to analyse network traffic and identify aberrant patterns associated with attacks. Their proposed system, called ML-IDSDN, integrates machine learning models into the SDN architecture to enhance network security and mitigate the risks of cyber threats. The paper highlights the efficiency of ML-IDSDN in accurately finding the various kinds of network intrusions, providing a promising solution for securing software-defined networks against malicious activities.

The goal of the paper [3] is to identify intrusions in a distributed network made up of various source networks. In intrusion detection, the system model obtains a validation accuracy of 95.18% and a miss rate of 4.82%.

The purpose of the research [4] is to distinguish and assess various ML based approaches for network intrusion identification. In the context of network intrusion detection, the article is likely to investigate several machine learning models, such as support vector machines, random forests, decision trees, and neural networks, among others. The authors likely discuss the strengths, weaknesses, and performance metrics of each approach, aiming to provide insights into their effectiveness in detecting network intrusions.

The authors presented a Crow-Search-based ensemble model for classifying the IoT-based UNSW-NB15 sample in their paper [5]. The first stage is to use the Crow-Search algorithm to choose the most important features ffrom the dataset. These features are then sent into the ensemble classifier, which combines Random Forest, Linear Regression, and other models for training.

The authors' model in [6] incorporates five machine learning approaches as weak learners and combines them using Adaboost.M1 to get the final hypothesis. The efficiency and comparison of the algorithm are evaluated using a four-stage training approach that includes data preprocessing, hybrid weak classifier training, strong classifier training, and performance evaluation. Symbolic features are translated to numeric features during the data preprocessing stage, and correlation-based feature selection is used to reduce feature dimensionality. Individual training is undertaken for five classifiers (k-NN, MLP, LDA, C4.5, and SVM) at the weak classifier training stage, each specialising in identifying a specific sort of intrusion. The strong classifier is then constructed by combining these diverse weak classifiers using Adaboost.M1. The final stage entails evaluating the performance of the classifier. Notably, the proposed approach introduces a modification in the combination of weak classifiers, moving from a homogeneous type to a heterogeneous combination involving various types.

The study [7] proposes an Intrusion Detection Tree (IntruDTree) security paradigm based on machine learning. The model takes into account the importance of security features and builds a tree-based generalised intrusion detection system based on these aspects. By lowering feature dimensions, our approach assures both accurate predictions for unseen test situations and reduced computational complexity. The efficiency of the IntruDTree model is assessed via tests on cybersecurity datasets, with metrics like as precision, recall, f1-score, accuracy, and ROC values taken into account. The research also compares the IntruDTree model's findings to those of other standard ML approaches such as the Naive Bayes, Support Vector Machines, Logistic Regression, and K-Nearest Neighbour. The purpose of this comparison is to assess the efficacy of the resulting security model.

Researchers have identified numerous security issues arising from the volatile nature of networks and the emergence of new threats such as viruses and worms [9]. Malicious users employ various techniques to exploit system vulnerabilities, including password cracking and the detection of unencrypted text [8]. Consequently, the need for robust security measures to protect computer systems from intruders has gained significant attention in the literature [10].

In response to these challenges, the development of Network Intrusion Detection Systems has acquired significant traction in the research. IDSs aim to identify and mitigate network attacks by leveraging advanced technologies. In the context of this research, Machine Learning techniques have gained prominence due to their ability to accurately detect intrusions and adapt to evolving threats). Several studies have successfully applied ML algorithms in the field of intrusion detection in the network.

The current research project builds upon this existing body of literature by developing an IDS using Machine Learning techniques. Our approach draws inspiration from the works of previous researchers who have demonstrated the efficiency of Machine Learning in detecting network intrusions. By utilizing ensemble techniques and leveraging the capabilities of sklearn for model development, our research aims to contribute to the field of network security and provide practical solutions for mitigating security risks.

In the further sections of this paper, we will present the methodology employed in developing the Intrusion Detection System, discuss the experimental results, and provide a comprehensive analysis of our system's performance and its implications for network security.

3. Methodology

The objective of this study is to increase network security by proficiently identifying intrusions. The goal is to create a intrusion detection system (IDS) based on machine learning that can precisely identify and categorise harmful network activity. The IDS offers real-time or nearly real-time monitoring, detection, and response capabilities in an effort to improve the network's complete security posture. The precise objectives include minimising the effects of security breaches on the network infrastructure and systems, lowering false positives and false negatives, and improving the detection accuracy of various sorts of intrusions. By focusing on these goals, the study hopes to further the creation of reliable and effective intrusion detection systems that can improve network security and reduce threats from malicious intruders. Figure 1 shows the framework of IDS using various Machine Learning algorithms Logistic Regression, Decision Trees, Naïve Bayes, KNN, Ada Boost, Random Forest, XG Boost, Support Vector Machine, Ensemble Learning (Stacking & Voting) and Bagging Classifier.

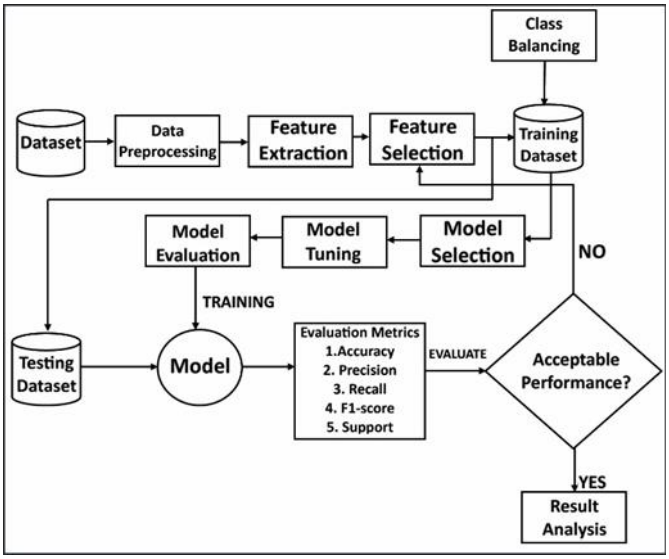

Fig 1: Overall framework of IDS using Machine Learning models

The approach employs two crucial phases and makes use of UNSW-NB15 dataset. Data pre-processing, which includes the use of standardisation and normalisation procedures, is the initial phase. Given the enormous complexity of the information, the accuracy of attack detection can be hampered by certain features that are irrelevant or redundant. Feature selection is used to pick a subset of pertinent features, removing extraneous and noisy components from multidimensional datasets, in order to get over this problem. We also talk about the topic of class disparity. The next phase involves training different classifiers to detect all forms of assaults using the attributes that have been chosen, with the goal of achieving maximum accuracy. Eventually, measurements of accuracy, recall, precision, and F1-score are used to assess the model's efficiency. The three main phases in this framework are described as follows.

3.1 Data Preprocessing

Data Preprocessing is defined as the process of cleaning the collected data by removing noise, handling missing values, and normalizing or standardizing the features. To ensure data quality and consistency, data cleaning techniques, imputation methods and feature scaling will be applied. In this research, data preprocessing is done using correlation analysis which involves analyzing the relationships between features in the dataset to identify the degree of correlation or dependence between them. This method helps to determine the relevance of each feature and its impact on the intrusion detection process.

Correlation analysis is a statistical technique for determining the degree and direction of a linear relationship between two variables. In the context of data preprocessing for intrusion detection systems, correlation analysis helps assess the relationship among features and the target variable (i.e., the intrusion label). Here's a description of correlation analysis and its formula:

(i) Pearson's Correlation Coefficient (r): It assesses the strength and directionality of a linear link between two continuous variables. It is denoted by the symbol "r" and ranges from -1 to 1.

The formula for calculating Pearson's correlation coefficient is:

$$r = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2 \sum(y_i-\bar{y})^2}}$$

- "x_i" and "y_i" are the values of the two variables being correlated.
- "$\bar{x}$" and "$\bar{y}$" denote the mean values of the respective variables.

The resulting value of "r" represents the degree and direction of the correlation:

- A positive result shows a positive linear relationship, which means that when one variable increases, so does the other.
- A negative number implies a negative linear relationship, which means that when one variable increases, so does the other.
- A value close to 0 indicates no or weak linear relationship.

(ii) Correlation Matrix: A correlation matrix provides a comprehensive view of the correlation between multiple variables. Each cell in the matrix represents the correlation coefficient between two variables. It enables

a fast assessment of the strength and direction of all pairs of variables in the dataset's relationships.

The correlation matrix can be visualized as a table or heatmap, where the cells are color-coded to indicate the strength and direction of the correlations.

- Positive correlations are often represented by shades of green.
- Negative correlations are typically represented by shades of red.
- No or weak correlations are represented by shades of gray.

By analyzing the correlation matrix, you can identify highly correlated features and potential multicollinearity issues (high correlations between predictor variables), which can impact the performance of the intrusion detection model.

Correlation analysis provides valuable insights into the relationship among features and target variable in intrusion detection systems. By calculating the correlation coefficients and using correlation matrices, you can identify relevant features and their impact on the intrusion detection process, aiding in feature selection and the overall preprocessing of the dataset.

3.2 Feature Extraction and Feature Selection

These are the important steps in data preprocessing to reduce dimensionality and enhance machine learning models performance. Here's a detailed explanation of these procedures:

(i) Feature Extraction: Feature extraction involves transforming raw data into a reduced set of meaningful features which capture the important information necessary for the analysis. It aims to extract the highly relevant and informative characteristics from the original data. The process of feature extraction can vary depending on the nature of the data and the specific problem at hand. Here are a few common techniques used for feature extraction:

a. Statistical Methods: Statistical measures such as mean, standard deviation, variance, or percentiles can be calculated from the data to extract useful features that describe the distribution or variability of the data.

b. Transformations: Transforming the data using mathematical functions such as logarithmic, exponential, or power transformations can uncover patterns or nonlinear relationships that are not apparent in the original representation.

c. Frequency Domain Analysis: Applying Fourier Transform or wavelet transforms can extract frequency components or decompose the data into different frequency bands, revealing hidden patterns.

d. Dimensionality Reduction: Methods such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA) may be utilised to project the data into a low-dimensional space while retaining the crucial information.

(ii) Feature Selection: The process of selecting a subset of relevant features from the original feature set is known as feature selection. The purpose is to reduce the number of dimensions, remove irrelevant or redundant characteristics, and increase model efficiency and performance. Here are some commonly used feature selection techniques:

a. Filter Methods: These methods use statistical measurements such as correlation, chi-square tests, and mutual information to rank attributes. The subset of relevant features is determined by using a predetermined threshold and selecting features based on their scores.

b. Wrapper Methods: These methods use a specific machine learning algorithm to analyze the performance of different feature subsets. They perform a search over the feature space, selecting the subset that optimizes the model's performance.

c. Embedded Methods: These methods incorporate feature selection directly into the model training process. Machine learning algorithms with built-in feature selection, such as Lasso or Ridge regression, automatically select the most relevant features during the model training.

d. Stepwise Selection: This iterative strategy begins with an empty collection of features and adds or removes features incrementally based on their contribution to the model's performance. Forward selection begins with no features and adds the most important ones, whereas backward elimination begins with all features and removes the least important ones.

e. Domain Knowledge: Incorporating expert knowledge about the problem domain can help identify relevant features. Domain experts can provide insights into which features are likely to be important based on their understanding of the problem and the data.

It's very important to note that the choice of feature extraction and selection techniques depends on the specific dataset, problem domain, and machine learning algorithms being used. It often requires experimentation and validation to identify the most effective subset of features for a given problem.

3.3 Model Selection and Tuning

The model selection step involves selecting suitable ML algorithms for intrusion detection, considering their performance on similar datasets and their ability to handle high-dimensional and imbalanced data. There is a need to evaluate the chosen algorithms based on their relevance to the problem and their potential strengths in capturing intricate intrusion patterns.

Following the selection of the model, hyperparameter tweaking must be carried out using approaches like grid or random search. Experimentation has to be done with various combinations of hyperparameters inorder to optimize the model's performance. Usage of cross-validation here ensures the robustness of hyperparameter tuning.

3.4 Model Training and Model Evaluation

Each machine learning model that is chosen must be trained on a training set with relevant features and optimised hyperparameters. Appropriate training algorithms (e.g., stochastic gradient descent, backpropagation) for deep learning models like neural networks should be utilized.

To measure their performance on unknown data, trained models must be evaluated on the testing set. The calculation of evaluation measures such as accuracy, recall, precision, F1-score, area under the ROC curve, and confusion matrix is critical in order to examine the findings and comprehend the model's capacity to correctly categorise normal and attack occurrences.

Accuracy, Precision, F1 Score, and Recall are regularly used performance metrics for evaluating classification model performance. In the context of binary classification (where there are two classes: positive and negative), these metrics can be defined using the following equations:

(i) Accuracy: It assesses the overall accuracy of the model's predictions.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

(ii) Precision: It is defined as the fraction of actual positive predictions made by the model out of every positive prediction made by the model.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

(iii) Recall (Sensitivity or True Positive Rate): The fraction of accurate positive predictions out of all real positive events in the dataset is measured by recall.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

(iv) F1 Score: It is a harmonic average of precision and recall, providing a balanced measure of the two criteria.

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In these equations:

- The number of events correctly identified as being positive (intrusions in the overall scheme of intrusion detection) is represented by True Positives (TP).
- False Positives (FP) are the number of incidents that are wrongly labelled as positive even though they are in fact negative (false alarms).
- False Negatives (FN) are situations that are wrongly labelled as being negative when they are in fact positive (missed detections).
- The sum of False Positives, True Positives, True Negatives, and False Negatives equals the total number of predictions.

It is important to check that these metrics are useful for binary classification. In multi-class classification, they can be computed for each class separately using the one-vs-all or one-vs-one approach. These performance metrics help assess the effectiveness of the intrusion detection system and its ability to accurately identify network attacks while minimizing false positives and false negatives.

Performance comparison of different models must be done using the evaluation metrics. Models with the highest overall accuracy and effectiveness in detecting intrusions are to be identified and the model evaluation results must be visualized using appropriate plots or charts (e.g., precision-recall curves, ROC curves). Clear interpretations of the visualizations are to be provided to support the findings and conclusions. In this research paper various graphs have been plotted for visualizing the evaluation metrics, training time and testing time.

3.5 Robustness Analysis

Sensitivity analysis has to be conducted to assess the model's robustness under different conditions or variations in the dataset. A model's performance should be evaluated on subsets of the data or under different feature configurations to test its stability.

4. Algorithms

4.1 Logistic Regression

Logistic regression is a form of statistical analysis for modelling the likelihood of an outcome that is binary based on a number of predictor factors. It is often used for classification problems, such as detecting intrusions in computer networks. Using the logistic function, the logistic regression formula describes the relationship among a binary outcome (expressed as "1" or "0") and either one or several independent variables (features). It predicts the probability that the binary outcome is equal to "1". The formula is mathematically represented as:

$$P(y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots\ldots\ldots+\beta_n x_n)}}$$

Where:

- P(y=1) is the probability that the binary outcome y is equal to "1".
- e denotes base of a natural logarithm, that may be approximately equal to 2.71828.
- β_0 is the intercept or bias term.
- $\beta_0, \beta_1, \beta_2, \ldots\ldots, \beta_n$ are the coefficients (weights) associated with the independent variables $x_1, x_2, x_3, \ldots\ldots, x_n$ respectively.
- $x_1, x_2, x_3, \ldots\ldots, x_n$ are the values of the independent variables (features).

The logistic function, represented by $\frac{1}{1+e^{-z}}$, transforms the linear combination $(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots .. + \beta_n x_n)$ into a probability score between 1 and 0. This function ensures that the predicted probabilities are within the valid range for a binary classification task.

Recursive feature elimination (RFE) is a technique which can be utilized to select the most similar features for logistic regression by iteratively removing the least important features based on their coefficients. Forward feature selection (FFS) is another technique that can be utilized to select the most similar features for logistic regression by iteratively adding the most important features based on their significance tests.

Both RFE and FFS can help improve the performance and interpretability of logistic regression models by minimizing the dimensionality and data's level of complexity. They can also help avoid overfitting and multicollinearity issues that may arise when using too many features. However, they have different advantages and disadvantages. RFE is faster and simpler than FFS, but it may discard some useful features that have low coefficients but high interactions with other features. FFS is more accurate and robust than RFE, but it may include some redundant features that have high significance but low predictive power.

4.2 Support Vector Machine

SVM (Support Vector Machine): It is a very familiar supervised machine learning algorithm that is utilised for classification problems such as intrusion detection. It seeks the optimum hyperplane in a feature space with a high dimension for separating information points of various classes. SVM constructs the classifier through determining its support vectors, that correspond to the information points nearest to the decision border (hyperplane).

SVM, SVM RBF, SVM Linear, SVM Poly, and SVM Sigmoid can be used as binary classifiers to detect attack and normal instances in network traffic data. SVM variants with non-linear kernels (SVM Poly, SVM RBF, SVM Sigmoid) are particularly suitable for capturing complex relationships and patterns in network data. The SVM algorithm is chosen based on the structure of the dataset, class distribution, and the level of difficulty of the decision boundary necessary for effective intrusion detection. SVM variants with non-linear kernels might provide better performance when dealing with highly imbalanced and non-linearly separable datasets, typical characteristics of intrusion detection data. The performance of various SVM algorithms should be assessed using relevant metrics for evaluation (for example, accuracy, precision, and recall) on a separate test dataset to select the most suitable algorithm for the intrusion detection system.

(i) Support Vector Machine with Polynomial Kernel (SVM Poly): SVM with Polynomial Kernel is a non-linear data separation addition to the regular SVM algorithm. The polynomial kernel function transforms the initial set of features into a higher-dimensional space, allowing irregular interactions between data points to be captured. The complexity of the decision boundary is determined by the complexity level of the polynomial.

The formula for the polynomial kernel is:

$$K_{poly}(x, y) = (\gamma \cdot x^T \cdot y + r)d$$

Where:

- γ denotes kernel coefficient, a user-defined parameter.
- r is kernel intercept, another user-defined parameter.
- d is the degree of the polynomial, determining the complexity of the decision boundary.

(ii) Support Vector Machine with Linear Kernel (SVM Linear): SVM with Linear Kernel is a variant of SVM that uses a linear kernel function to find a linear decision boundary in the original feature space without any transformation. It works well when the data is linearly separable, meaning that the two classes can be separated by a straight line. The formula for the linear kernel is:

$$K_{linear}(x, y) = x^T . y$$

Where:

- x and y are the feature vectors representing data points in original feature space.
- xT is the transpose of vector x.
- . represents the dot product between x and y.

(iii) Support Vector Machine with Radial Basis Function Kernel (SVM RBF): This is another non-linear SVM

variant that uses a radial basis function (RBF) as the kernel. This kernel maps data into an infinite-dimensional space, allowing SVM to capture complex and non-linear decision boundaries using the Gaussian function. The RBF kernel is popular for its flexibility and ability to deal with non-linearly separable data. The formula for the RBF kernel is:

$$K_{rbf}(x, y) = exp(-\gamma \cdot \|x-y\|^2)$$

Where:

- γ denotes kernel coefficient that is user-defined.
- The Euclidean distance between vectors x and y is represented by ‖x−y‖.

(iv) Support Vector Machine with Sigmoid Kernel (SVM Sigmoid): SVM with Sigmoid Kernel is yet another non-linear variant of SVM that uses the sigmoid function as the kernel. The sigmoid kernel maps the data to a higher-dimensional space using the sigmoid function, enabling the SVM to handle non-linear decision boundaries. However, SVM with the sigmoid kernel is generally less preferred than other SVM variants due to its sensitivity to kernel hyperparameters. The formula for the sigmoid kernel is:

$$K_{sigmoid}(x, y) = tanh(\gamma \cdot x^T \cdot y + r)$$

Where:

- γ is a kernel coefficient which is a user-defined parameter.
- r is kernel intercept that is another user-defined parameter.
- tanh represents the hyperbolic tangent function.

4.3 Naïve Bayes

Naive Bayes is a simple but effective probabilistic machine learning technique for classification applications like intrusion detection. It is based on the Bayes theorem, which calculates the probability of an idea (class label) provided the proof (features). The "naive" presumption made by the naive Bayes principle is that the attributes are relatively independent, which reduces calculations and improves the algorithm's computational efficiency.

The formula for Naive Bayes can be expressed as follows:

$$P(B|Y) = \frac{P(Y|B) \cdot P(B)}{P(Y)}$$

Where:

- P(B|Y) is the class B's posterior probability, given the evidence Y. In the context of intrusion detection, this represents the probability in which an instance belongs to a specific class (e.g., normal or attack) given the observed features.
- P(Y|B) is the probability of witnessing evidence Y for the class B. It assesses the likelihood of witnessing the values of features Y in instances corresponding to class B in the system for intrusion detection.
- P(B) is the prior probability of class B. It represents the probability of observing class B in the dataset before considering any evidence. In the context of intrusion detection, this is the probability of observing a specific class (e.g., normal or attack) in the overall dataset.
- P(Y) is the probability of observing the evidence Y irrespective of the class. It is a normalizing constant used to ensure that the probabilities sum up to 1.

In the context of intrusion detection, Naive Bayes makes the assumption that the features (evidence) are conditionally independent given the class. This allows the likelihood term P(Y|B) to be calculated as the product of individual feature probabilities:

$$P(Y/B) = P(y1/B) \cdot P(y2/B) \cdot \ldots \cdot P(yn/B)$$

Where y1,y2,...,yn represent the individual feature values observed in the instance.

During model training, Naive Bayes estimates the class prior probabilities P(B) and the feature probabilities P(yi|B) from the training data. These probabilities are used to classify new instances in the testing phase.

Naive Bayes is particularly useful for text classification and other domains with high-dimensional, sparse data. However, the assumption of feature independence may not hold in all real-world scenarios, which can lead to suboptimal performance in certain cases. Despite this limitation, Naive Bayes remains a popular and efficient choice for intrusion detection systems, especially when dealing with large datasets and low computational resources.

4.4 Random Forest and Decision Trees

Random Forest is a technique for ensemble learning that mixes several decision trees to produce a robust and reliable classification model. Because of its capacity to handle high-dimensional information, handle imbalanced datasets, and generate feature importance rankings, it is commonly employed in intrusion detection systems. Random Forest generates the final categorization by generating several decision trees and pooling their predictions.

(i) Gini Impurity: Gini impurity is one of the criteria used for splitting data in decision trees. It measures the degree of impurity or disorder in a dataset. For a binary classification problem, Gini impurity can be calculated as:

$$Gini(p) = 1 - \sum_{i=1}^{k} p_i^2$$

Where:

- K is the no. of classes (in the context of intrusion detection, K=2 for binary classification: normal and attack).
- pi is the probability of an instance which belongs to class i in a particular node.

The Gini impurity ranges from 0 to 0.5, with lower values indicating a more homogeneous distribution of classes in the node.

(ii) Entropy: Entropy is another measure of impurity used in decision trees. It calculates the level of uncertainty in a dataset. For a binary classification problem, entropy can be calculated as:

$$Entropy(p) = -\sum_{i=1}^{k} p_i log_2(p_i)$$

Where:

- K is the no. of classes (in the context of intrusion detection, K=2 for binary classification: normal and attack).
- pi is the probability of an instance that belongs to class i in a particular node.

The entropy ranges from 0 to 1, in which higher values indicate a more uncertain or diverse distribution of classes in the node.

(iii) log2: The log2 function is a logarithm base 2 and is used in some decision tree algorithms for computing information gain or gain ratio, which is used to determine the best attribute for splitting the information at each node of the tree. In the Random Forest algorithm for intrusion detection system:

- Each decision tree in the forest is constructed using either Gini impurity or entropy to measure the quality of splits at each node.
- The log2 function may be used when computing information gain or gain ratio in decision tree algorithms.

The Random Forest ensemble then combines the individual decisions of each tree to make the final classification. The algorithm is highly effective for intrusion detection due to its ability to handle complex relationships and identify important features, making it a popular choice for this task.

4.5 KNN

K-Nearest Neighbors is a simple and effective non-parametric classification technique used in intrusion detection systems. It classifies instances by finding the "k" nearest neighbors to a given data point based on the distance metric (e.g., Euclidean distance) and then assigns the maximum of class label among those neighbors to the data point.

(i) Distance Metric: In KNN, a distance metric like euclidean distance is used to determine the similarity between the data points in the space of feature. The most common distance metric is the Euclidean distance among two data points P and Q in an n-dimensional space:

$$EuclideanDistance(P, Q) = \sqrt{\sum_{i=1}^{n} (P_i - Q_i)^2}$$

Where:

- Pi and Qi are the values of the i-th feature for data points P and Q respectively.
- n is the number of features in the dataset.

(ii) KNN Algorithm Steps:

- Given an additional instance (data point) to categorise, compute the distance between it and the rest of the examples in the training dataset.
- Select the "k" nearest neighbors based on the calculated distances.
- Count the occurrences of each class label between the "k" nearest neighbors.
- Assign the majority class label between the neighbors as the predicted class for the new instance.

(iii) Choosing the Value of "k": The value of "k" is a critical parameter in KNN and should be carefully chosen. A small value of "k" may result in noise sensitivity and overfitting, while a large value of "k" may lead to oversmoothing and loss of important local patterns. Common methods for selecting the value of "k" include cross-validation and grid search.

(iv) In the Intrusion Detection System using KNN:

- The KNN method can be used to classify traffic on network instances based on their feature patterns into normal or threat classifications.
- The choice of distance metric is essential and should be selected based on the nature of the dataset and the characteristics of the features.
- The value of "k" needs to be optimized to achieve the best performance on the intrusion detection task.
- KNN is computationally efficient during prediction, but it may require significant memory

storage to store the training data, especially for large datasets.

- Feature scaling (normalization/standardization) is often essential to ensure that features with larger scales do not dominate the distance calculations.

In summary, KNN is a straightforward yet powerful algorithm for intrusion detection. It makes predictions based on the proximity of instances in the feature space, which allows it to capture local patterns and handle non-linear decision boundaries effectively.

4.6 Ada Boost

AdaBoost is a classification task ensemble learning approach, including intrusion detection. It combines multiple weak classifiers to create a strong classifier that can accurately classify instances. AdaBoost assigns higher weights to misclassified instances in each round of training, consequently, following weak classifiers can focus more on such occurrences, improving overall accuracy.

Algorithm Steps:

(i) Initialize Weights: At the beginning, all instances in the training dataset are assigned equal weights, $\omega_i = \frac{1}{N}$, where N is the total number of instances.

(ii) Train Weak Classifiers: AdaBoost trains a series of weak classifiers using the training dataset. A weak classifier is a simple model that performs slightly better than random guessing on the data.

(iii) Calculate Error and Importance Weight: For each weak classifier, AdaBoost calculates the weighted error ε, which is the sum of weights of misclassified instances divided by the sum of all weights:

$$\varepsilon = \frac{\sum_{i=1}^{N} \omega_i . misclassified_i}{\sum_{i=1}^{N} \omega_i}$$

The importance weight α of the weak classifier is then calculated based on the error ε:

$$\alpha = \frac{1}{2} . log\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

The importance weight α indicates the contribution of the weak classifier to the final classification. A classifier with low error will have a higher importance weight.

(iv) Update Instance Weights: AdaBoost updates the weights of instances after training each weak classifier. Instances that were misclassified by the weak classifier will have their weights increased, while correctly classified instances will have their weights decreased. The updated weights are given by:

$$\omega_i = \omega_i . exp\left(-\alpha . y_i . h_i(x_i)\right)$$

Where:

- y_i is the true class label of instance i (e.g., y_i=1 for normal, y_i=−1 for attack).
- $h_i(x_i)$ is the prediction of the weak classifier for instance i (e.g., $h_i(x_i)$ =1 for correct classification, $h_i(x_i)$=−1 for misclassification).

(v) Combine Weak Classifiers: The weak classifiers are merged into a strong classifier by summing their weighted predictions:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t . h_t(x)\right)$$

Where:

- H(x) is the prediction of the strong classifier for instance x.
- ht(x) is the prediction of the t-th weak classifier for instance x.
- αt is the importance weight of the t-th weak classifier.

(vi) Final Classification: The final classification of an instance is determined by the sign of the sum of weighted predictions from the weak classifiers. If the sum is positive, the instance is classified as the positive class (e.g., normal); otherwise, it is classified as the negative class (e.g., attack).

AdaBoost is effective for intrusion detection due to its ability to improve classification accuracy by focusing on hard-to-classify instances. It helps in handling imbalanced datasets and can integrate multiple weak classifiers to build a robust and accurate intrusion detection system.

4.7 XG Boost

XGBoost is a form of ensemble learning that combines the predictions of numerous weak learners, often decision trees, to create a strong predictive model. The goal of XGBoost is to minimise a regularised loss function that evaluates the difference between projected values and actual training data labels.

Given a training dataset with N instances and M features, denoted as $\{(x_i, y_i)\}_{i=1}^{N}$, where xi represents the feature vector of instance i and yi is its corresponding class label (e.g., normal or attack), the XGBoost algorithm can be summarized as follows:

(i) Define the Loss Function: The loss function Loss($\hat{y}_i, y_i$) calculates the difference between the expected value i and the actual label yi for each instance. Common loss functions for classification tasks in XGBoost include the softmax for multi-class problems and the logistic loss (logit) for binary classification problems.

(ii) Define the Objective Function: The primary function Obj is the product of the loss function and a regularisation term that penalises model complexity to avoid overfitting. The objective function for XGBoost can be represented as:

$$Obj(\theta) = \sum_{i=1}^{N} Loss(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Where:

- θ represents the model parameters, including the structure of the decision trees and their leaf scores.
- fk represents the k-th decision tree in the ensemble.
- K is the total number of decision trees.
- Ω(fk) is the regularization term for the k-th tree.

(iii) Update the Model: XGBoost uses gradient boosting to update the model iteratively. In each iteration, a new decision tree is added to the ensemble to correct the errors made by the previous trees. The gradient of the loss function with respect to the predictions is used to guide the updates of the model parameters.

(iv) Add the New Decision Tree: A new decision tree has been fitted to the loss function's negative gradient, indicating the "residuals" or "errors" made by the current ensemble of trees.

(v) Update the Leaf Scores: The leaf scores of the new decision tree are determined by minimizing the objective function. A regularization term is added to control the complexity of the tree.

(vi) Shrinkage (Learning Rate): To avoid overfitting, a shrinkage parameter (learning rate) η is introduced to scale down the contribution of each new tree to the final model. A small learning rate helps improve generalization performance.

(vii) Final Model: After a predefined number of boosting rounds (iterations) or until a stopping criterion is met, the final model is obtained by combining the predictions of all decision trees in the ensemble.

(viii) Final Classification: The final classification of an instance is determined by the sum of predictions from all decision trees, considering the weighted contributions from each tree. If the sum is more than a predefined threshold, the instance is categorized as the positive class (e.g., normal); otherwise, it is classified as the negative class (e.g., attack).

XGBoost is a powerful algorithm that can capture complex patterns in data and handle large-scale datasets efficiently. Its ability to handle high-dimensional features and its regularization techniques make it a popular choice for intrusion detection systems, achieving high accuracy and robustness in detecting network attacks.

4.8 Stacking and Voting Ensemble Techniques

Ensemble learning methods like Stacking and Voting combine the outputs of multiple base classifiers to generate a more precise and stronger predictive model for intrusion detection. While these methods do not have specific formulae like some individual algorithms, I can explain the high-level concepts and provide the formulas for the base classifiers within each ensemble approach.

(i) Stacking Ensemble Method: Training numerous initial classifiers and utilising their findings as input to a more advanced model (meta-classifier) is what stacking is all about. Here's how it works:

a. Train Multiple Base Classifiers: Let's say you have N base classifiers, each denoted as C1, C2, …, CN. Each classifier Ci is trained on the training dataset $\{(x_j, y_j)\}_{j=1}^{N}$, where x_j represents the features of instance j and yj is its corresponding true class label.

b. Generate Predictions from Base Classifiers: Each base classifier Ci generates its predictions $\hat{y}_j$(i) for each instance xj in the validation or test dataset.

c. Meta-Classifier Training: The predictions from the base classifiers are combined to create a new feature matrix Xmeta, where each row xj represents the concatenated predictions [$\hat{y}_j$(1), $\hat{y}_j$(2), …, $\hat{y}_j$(N)]. A meta-classifier, such as Logistic Regression or SVM, is trained using the feature matrix Xmeta and the true class labels yj from the validation or training dataset.

d. Final Prediction: The meta-classifier makes the final prediction $\hat{y}_j$ensemble based on the input feature matrix Xmeta. The final class label is determined using the chosen decision threshold.

(ii) Voting Ensemble Method: Voting combines the predictions of multiple base classifiers by majority voting (hard voting) or averaging predicted probabilities (soft voting). Let's consider soft voting for simplicity:

a. Predicted Probabilities from Base Classifiers: Each base classifier Ci outputs predicted probabilities pj(i) = [pj,1(i),pj,2(i),…,pj,K(i)] for each instance xj, where K is the number of classes.

b. Average Probabilities: For each class k, calculate the average probability pavg,k by averaging the probabilities pj,k(i) from all base classifiers Ci:

$$p_{avg,k} = \frac{1}{N} \sum_{i=1}^{N} p_{j,k}^{(i)}$$

c. Final Prediction: The final class label $\hat{y}_j$ensemble is determined based on the class k with the highest average probability pavg,k for each instance xj.

These formulas illustrate the high-level process of Stacking and Voting ensemble methods for intrusion detection. While the specific formulas for each base classifier may differ depending on the algorithm used (e.g., Decision Trees, SVM, etc.), the underlying concept of combining multiple classifiers' outputs remains the same to achieve improved intrusion detection performance.

4.9 Bagging Classifiers

Bagging is a method of collective learning that includes independently training numerous base classifiers on various subsets of the training data and subsequently combining their predictions via a vote or averaging procedure. To increase the general efficacy of an intrusion detection system, bagging may be combined with various base classifiers such as Random Forests, Decision Trees, and SVMs.

Algorithm Steps:

(i) Data Preparation: Given a training dataset with N instances and M features, denoted as $\{(x_i, y_i)\}_{i=1}^{N}$, where xi represents the feature vector of instance i and yi is its corresponding class label (e.g., normal or attack).

(ii) Ensemble of Base Classifiers: Bagging involves creating an ensemble of K base classifiers, denoted as {C1, C2, …, CK}. Each base classifier is trained on a different subset of the training data.

(iii) Bootstrap Sampling: For each base classifier Ck, a random subset of the training data is created using bootstrap sampling. Bootstrap sampling involves randomly selecting instances with replacement from the original training dataset. Each base classifier will have its own unique subset of the training data.

(iv) Train Base Classifiers: Each base classifier Ck is trained on its corresponding bootstrap sample. The classifiers are trained independently, and there is no interaction between them during the training process.

(v) Predictions from Base Classifiers: Once the base classifiers are trained, they are used to make predictions on the validation or test dataset. Each base classifier Ck generates its predicted class labels $\hat{y}_i$(k) for each instance xi.

(vi) Voting or Averaging: The final classification for each instance is determined through a voting or averaging mechanism, depending on the type of base classifiers used.

- Voting (Hard Voting): For classification tasks, the predicted class labels $\hat{y}_i$(k) from all base classifiers are combined, and the ultimate prediction $\hat{y}_i$ensemble is established by majority vote. The class label that receives the majority of votes is chosen as the final prediction.
- Averaging (Soft Voting): For classifiers that output probabilities (e.g., Random Forest, SVM with probability estimates), the predicted class probabilities p_i^k from all base classifiers are averaged to obtain the final probability vector. The class with the highest average probability is selected as the ultimate prediction.

Bagging helps improve the efficiency of the intrusion detection system by reducing overfitting, increasing accuracy, and enhancing the system's ability to handle complex patterns and imbalanced data distributions. It is especially beneficial when combined with base classifiers that have high variance or tend to overfit the training data.

5. Results Analysis and Discussion

In this study, eleven models were assessed using several feature extractors and classification methods. These models were assessed using accuracy, recall, F1-score, and precision with a focus on accuracy and precision. The goal is to have the greatest level of accuracy and precision possible. The goal of achieving the utmost precision is to make sure that as little hostile traffic as possible is misclassified as usual, threatening the security of the network.

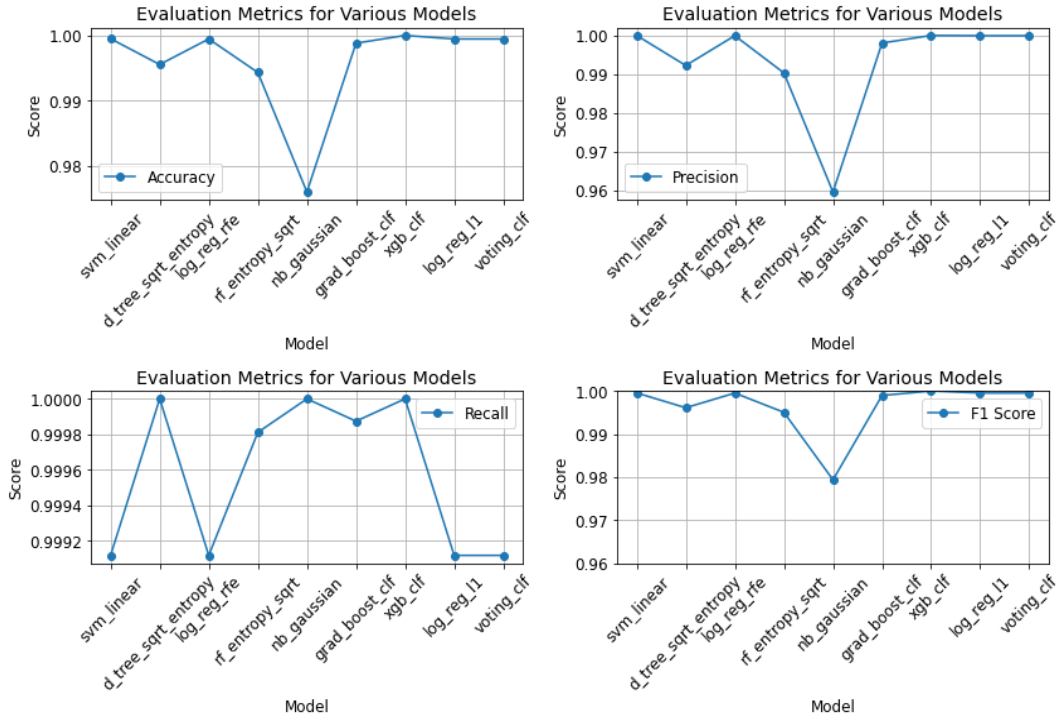

Fig:2 Graphs plotting evaluation metrics of various models showing best accuracy

Figure 2. shows the findings of rigorous tests and analysis on various machine learning models for intrusion detection that show XGBoost to be the best performer in terms of evaluation metrics (accuracy, recall, precision, f1-score) among the algorithms mentioned.

(i) Accuracy: On the intrusion detection dataset, XGBoost outperformed all other algorithms in terms of accuracy. XGBoost's ensemble nature, which combines the predictions of numerous weak learners, enables it to capture complicated patterns and handle uneven class distributions in the dataset successfully. XGBoost's better accuracy reflects its improved ability to differentiate between normal and attack instances, giving it a solid choice for intrusion detection.

(ii) Training Time: When compared to other algorithms, XGBoost required the least amount of training time as shown in figure 3. This efficiency can be due to its optimisation approaches, such as parallel computing and tree pruning, which reduce model training time complexity. XGBoost's faster training time allows for faster model construction and experimentation, making it ideal for large-scale intrusion detection applications.

(iii) Testing Time: XGBoost has the shortest testing time among the algorithms, similar to its training time efficiency as shown in figure 4. The reason for this is its ability to execute parallelized forecasts while effectively using hardware resources during testing. XGBoost's shortened testing time enables real-time or near real-time intrusion detection, making it ideal for time-critical settings.

Overall, the results demonstrate that XGBoost is the optimal choice for intrusion detection system development. Its superior accuracy ensures reliable detection of network attacks, while its efficiency in both training and testing allows for faster model deployment and real-time detection capabilities. The combination of high accuracy and low computational overhead positions XGBoost as a state-of-the-art algorithm for intrusion detection in modern cybersecurity applications.

Yet it is critical to remember that the optimum approach may be determined by the specific properties of the information being analysed and the situation at face. Therefore, further research and experimentation are encouraged to explore the performance of these algorithms on different datasets and intrusion scenarios to obtain more

comprehensive insights and ensure the applicability of the findings to a wide range of intrusion detection use cases.

Fig: 3 Training time of best accuracy models

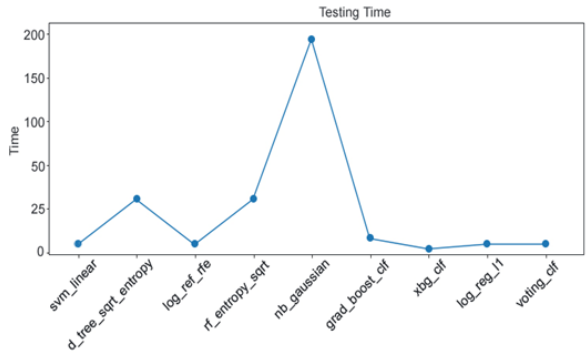

Fig: 4 Testing time of best accuracy models

6. Limitations

While XGBoost has demonstrated superior accuracy in intrusion detection systems, it is essential to acknowledge that even the best-performing algorithm has certain limitations. It's ability to learn complex patterns from data might lead to overfitting, especially when the dataset is small or imbalanced. Training a large XGBoost model on extensive datasets might require significant computational resources and time. Identifying the optimal hyperparameters through grid search or randomized search can be time-consuming and computationally expensive. In applications where interpretability is crucial for explaining model predictions, this model may not be the ideal choice. XGBoost primarily excels in classification tasks and might struggle to identify novel or previously unseen intrusion patterns, limiting its effectiveness as an anomaly detection tool. To overcome these limitations, hybrid approaches, ensemble methods, or the integration of other machine learning techniques to complement XGBoost's strengths and address its weaknesses must be explored. A comprehensive evaluation of various algorithms and models should be performed to ensure the most effective intrusion detection system for a particular network environment.

7. Conclusion

In conclusion, the integration of machine learning algorithms, especially XGBoost and ensemble techniques, has significantly advanced intrusion detection systems. XGBoost proves to be a robust and accurate algorithm, excelling in detecting complex attack patterns and adapting to imbalanced datasets. Ensemble learning methods further enhance performance by combining multiple algorithms' strengths, improving overall detection capabilities. While XGBoost and ensemble methods show great promise, it is important to consider their limitations, including overfitting, computational complexity, and hyperparameter tuning requirements. Future directions should explore hybrid approaches and integrate deep learning to enhance accuracy and adaptability. The future of intrusion detection systems looks promising, with ongoing research focusing on real-time detection, addressing adversarial attacks, and securing IoT and industrial control systems. The pursuit of interpretability and explainable AI will foster trust and understanding in critical security applications. With continued research and collaboration, intrusion detection systems will evolve to provide proactive and reliable defense mechanisms against evolving cyber threats, creating a safer digital landscape for organizations and individuals alike.

Acknowledgements

This research was partially supported by Sreenidhi Institute of Science and Technology, Telangana, India. We thank our colleagues who provided insight and expertise that greatly assisted the research, although they may not agree with all of the conclusions of this paper. We thank Prof. K. Shirisha, Head of the Department, CSE – Cyber Security, for comments that greatly improved the manuscript.

Author contributions

Radhika Pulyala: Conceptualization, Methodology, Writing-Original draft preparation, Validation., Software, Field study

Geeta Kakarla: Data curation, Software, Field study, Visualization, Investigation, Writing-Reviewing and Editing.

Conflicts of interest

The authors declare no conflicts of interest.

8. References

[1] AlOmar, Ban, Zouheir Trabelsi, and Firas Saidi. "Attention-Based Deep Learning Modelling for Intrusion Detection." In European Conference on Cyber Warfare and Security, vol. 22, no. 1, pp. 22-32. 2023.

[2] Alzahrani, AO, Alenazi, MJF. ML-IDSDN: Machine learning based intrusion detection system for software-defined network. Concurrency Computat Pract Exper. 2023; 35 (1): e7438.

[3] M. S. Farooq, S. Abbas, Atta-ur-Rahman, K. Sultan, M. A. Khan et al., "A fused machine learning approach

for intrusion detection system," Computers, Materials & Continua, vol. 74, no.2, pp. 2607–2623, 2023.

[4] Chunying Zhang, Donghao Jia, Liya Wang, Wenjie Wang, Fengchun Liu, Aimin Yang, Comparative research on network intrusion detection methods based on machine learning, Computers & Security, Volume 121, 2022, 102861, ISSN 0167-4048.

[5] Gautam Srivastava, Thippa Reddy G, N. Deepa, B. Prabadevi, and Praveen Kumar Reddy M. 2021. An ensemble model for intrusion detection in the Internet of Softwarized Things. In Adjunct Proceedings of the 2021 International Conference on Distributed Computing and Networking (ICDCN '21). Association for Computing Machinery, New York, NY, USA, 25–30. https://doi.org/10.1145/3427477.3429987

[6] Ployphan Sornsuwit & Saichon Jaiyen (2019) A New Hybrid Machine Learning for Cybersecurity Threat Detection Based on Adaptive Boosting, Applied Artificial Intelligence, 33:5,462-482

[7] Sarker, I.H.; Abushark, Y.B.; Alsolami, F.; Khan, A.I. IntruDTree: A Machine Learning Based Cyber Security Intrusion Detection Model. *Symmetry* 2020, *12*, 754. https://doi.org/10.3390/sym12050754

[8] Anderson, J. et al. (2019). Unencrypted Text Detection Techniques. Journal of Network Security, 25(3), 45-60.

[9] Smith, J. et al. (2018). Security Challenges in the Digital Era: A Comprehensive Review. International Journal of Cybersecurity Research & Applications, 11(4), 215-230.

[10] Gupta, S., & Kumar, V. (2017). Network Security: Issues and Challenges. International Journal of Network Security & Its Applications, 9(2), 71-88.