Sustainable Machine Intelligence Journal

Machine Learning for Intrusion Detection: A Reproducible Baseline is All You Need

Salma A. Walli[1], Karam Sallam[2]

[1] Faculty of Computers and Informatics, Zagazig University, Zagazig, Sharqiyah 44519, Egypt; 20912019200851@fci.zu.edu.eg.
[2] School of IT and Systems, Faculty of Science and Technology, University of Canberra, Canberra, Australia; karam.sallam@canberra.edu.au.

Abstract: Ensuring Responsible AI practices is paramount in the advancement of systems founded upon machine learning (ML) principles, particularly in sensitive domains like intrusion detection within cybersecurity. A fundamental aspect of Responsible AI is reproducibility, which guarantees the reliability and transparency of research outcomes. In this paper, we address the critical challenge of establishing reproducible for intrusion detection utilizing ML techniques. Leveraging the NSL-KDD dataset and the Edge-IIoTset, we carry out extensive experiments to evaluate the efficacy of our approach. Our study prioritizes meticulous experiment design and careful implementation setups, aligning with the principles of Responsible AI. Through rigorous experimentation and insightful discussions, we underscore the importance of reproducibility as a cornerstone in ensuring the resilience and reliability of intrusion detection systems. Our findings offer valuable insights for researchers and practitioners striving to develop Responsible AI solutions in cybersecurity and beyond. The source code is publicly accessible at https://github.com/Salma-00/Machine-Learning-for-Intrusion-Detection.

Event	Date
Received	22-02-2024
Revised	26-04-2024
Accepted	18-05-2024
Published	20-05-2024

Keywords: Anomaly Detection, Cyber-Attacks, Machine learning, Reproducibility, Baseline, Cybersecurity, Evaluation metrics, Security threats.

1. Introduction

In the realm of cybersecurity, the integration of intrusion detection with machine learning (ML) has emerged as a potent avenue for enhancing defensive capabilities against evolving cyber threats. Notable research endeavors have emphasized the indispensable contribution of ML in augmenting the effectiveness of intrusion detection systems. Drawing upon widely recognized benchmark datasets, scholars have conducted comprehensive comparative analyses aimed at assessing the performance of various ML algorithms [1]. These empirical investigations offer valuable perspectives into the capabilities of different ML approaches. Through meticulous analysis, researchers have uncovered a wealth of knowledge regarding the efficacy of various approaches. As they scrutinized a diverse array of algorithms, ranging from traditional methods to cutting-edge technologies, intriguing trends began to emerge. Showcasing promising potential in discerning network patterns and identifying complex behaviors [2].

Ensemble learning methods emerge as frontrunners with superior predictive capabilities compared to individual algorithms. Their ability to combine multiple models enhances detection accuracy and resilience against diverse attack vectors [3]. Beyond numerical metrics, studies shed light on the nuanced strengths and limitations of each algorithm. While some algorithms may excel in recognizing known attack patterns, others demonstrate resilience in identifying

novel threats. This underscores the importance of a multifaceted approach to intrusion detection, leveraging the complementary strengths of various ML techniques [2],[3].

The primary contributions of this study can be encapsulated as follows:

- Our work contributes to the body of knowledge by developing reproducible baselines to advocate for the establishment of open-source benchmarks that provide a solid foundation that facilitates comparison between existing and upcoming research methodologies in cybersecurity research.
- Inclusive and fair experimentation on two distinct datasets, the NSL-KDD dataset, and the Edge-IIoTset, is performed to assess the effectiveness of different ML algorithms across diverse network environments. This ensures the robustness, transparency, and comparability of our results, contributing to the advancement of the subject matter of this research.

The remaining of our work is structured as follows. First, we begin by reviewing the existing literature in Section 2, providing insights into the current state-of-the-art techniques and methodologies in the field. In Section 3, we delve into the details of our proposed ML-based intrusion detection algorithm. Section 4 outlines our meticulous experiment design, delineating the datasets used, feature engineering strategies, and evaluation metrics. The findings of our experiments are discussed in Section 5, where we analyze the experimental results. Finally, in Section 6, we present our conclusions.

2. Background and Literature

In the past few years, there has been a notable increase in research efforts focusing on intrusion detection systems (IDS) bolstered by ML (ML) techniques, particularly for safeguarding critical infrastructure. A comprehensive survey conducted by Pinto et al. provides valuable insights into the landscape of IDSs tailored for protecting critical infrastructure [1]. The study emphasizes the significance of ML-based IDSs in fortifying cybersecurity measures, especially within the domain of critical infrastructure vulnerabilities. Through analyzing various ML techniques applied to intrusion detection, the survey highlights the efficacy of supervised, unsupervised, and reinforcement learning approaches in identifying cyber threats. Furthermore, it delves into the challenges associated with real-world implementation and the need for adaptive and scalable IDS solutions to mitigate evolving cyber threats.

In a similar vein, Latif et al. embarked on an investigation into the efficacy of ML algorithms for network intrusion detection [2]. Focused on real-world applications, their study explores the performance of different ML techniques using benchmark datasets. Through empirical analysis, the research elucidates the strengths and limitations of various ML algorithms, focusing on their suitability for intrusion detection tasks. The study underscores the significance of feature scaling, reduction, and oversampling techniques in optimizing the performance of ML-based IDSs. Furthermore, Zhang et al. conducted a comparative study on network intrusion detection methods leveraging ML algorithms [3]. Through systematically analyzing the implementation of conventional ML, ensemble learning, and deep learning techniques, the research provides an exhaustive review of the advancements in intrusion detection technology. Comparative experiments conducted on datasets such as KDD CUP99 and NSL-KDD reveal the varying performance metrics, encompassing detection accuracy, F1 score, and area under the curve (AUC), across different ML algorithms. The comparative analysis shows the superiority of ensemble learning algorithms in terms of overall detection efficacy, while also acknowledging the nuanced advantages of Naive Bayes in recognizing novel attack types. The collective findings from these studies highlight the critical role of ML techniques in enhancing the capabilities of intrusion detection systems [4],[5]. Expressing the advantages and drawbacks of different ML algorithms and techniques opens the way for the development of more robust and adaptive IDS solutions to combat evolving cyber threats.

3. ML Baselines

3.1. Logistic regression

Logistic regression is a commonly utilized classification approach rooted in the principles of probabilistic statistics, employing binary data. Logistic regression predicts an outcome variable with two possible categories from one or more predictor variables, where the predicted variable takes values of 0 or 1. Logistic regression utilizes the logistic function, often referred to as the sigmoid function, to transform linear combinations of predicted variables into probabilities. This transformation facilitates Logistic regression's ability to model the probability of the binary outcome based on the predictors. Through maximum likelihood estimation, Logistic regression estimates parameters to establish the relationship between variables and the binary outcome [6]. The logistic function equation (1) used in Logistic regression can be expressed as:

$$g(z) = \frac{1}{1+e^{-z}} \quad (1)$$

Where z denotes the linear combination of predictor variables and model coefficients. The logistic function g(z) maps the linear combination z onto the interval [0, 1], representing the probability of the positive class. Through setting a threshold (typically 0.5), Logistic regression determines the class label based on whether the predicted probability exceeds this threshold [6],[7].

3.2. K-nearest neighbors (KNN)

K-nearest neighbors (KNN) classification, a fundamental approach in ML, is recognized for its simplicity and efficiency in numerous classification endeavors. Originating as one of the earliest algorithms in the field, KNN remains a cornerstone in the repertoire of classification techniques. Its utility spans diverse applications, including the discrimination between different categories, based on selected features [8]. The KNN approach applies the concept of determining class labels by a majority vote among the nearest neighbors of an unknown instance in the training set. Central to this process is the calculation of distances among data points, with the Euclidean distance benchmark frequently employed for its simplicity and effectiveness in quantifying dissimilarities between feature vectors [8]. Mathematically, the Euclidean distance d (x, x′) between two data points x and x′ in n-dimensional space is expressed as:

$$d(x, x') = \sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2} \quad (2)$$

where $x_i \; and \; x'_i$ expresses the respective feature values of the i-th dimension. This distance metric forms the basis for identifying the nearest neighbors of a given instance, thus informing the classification decision. The classification rule in KNN is straightforward. The class label of an unidentified instance is determined by the predominant class among its K nearest neighbors. This decision is encapsulated by the function $f\mathrm{KNN}(x)$, defined as:

$$f\mathrm{KNN}(x) = \begin{cases} 1, & if \; \sum_{i \in N_k(x)} y(i) \geq 0 \\ -1, & if \; \sum_{i \in N_k(x)} y(i) < 0 \end{cases} \quad (3)$$

where $N_k(x)$ denotes the set of indices of the K nearest patterns, and $y(i)$ represents the class label of the i-th nearest pattern [9]. This mechanism allows KNN to make predictions without the need for pre-processing or training data manipulation, thus showcasing its versatility and ease of implementation [8]. However, the effectiveness of KNN hinges significantly on the choice of the number of nearest neighbors (K). Optimal performance necessitates the selection of an appropriate K value, which often requires careful consideration and experimentation. Techniques such as cross-validation play a pivotal role in determining the optimal model parameters, ensuring robust and reliable classification outcomes [8]. In addition to its binary classification capabilities, KNN can be expanded to multi-class classification scenarios, where it predicts the class label based on the dominant class among the nearest neighbors in the data space.

This flexibility further enhances the applicability of KNN across a wide range of classification tasks, cementing its status as a foundational technique in ML [9].

3.3. Decision Tree Classifier

Decision trees are foundational in data mining, particularly for creating classifiers. They excel in managing vast amounts of data, allowing for categorical assumptions, classification based on training sets, and handling new data categorization. Among classification algorithms, decision trees stand out for their simplicity and effectiveness across various domains such as ML, image processing, and pattern recognition [10]. Operating through a series of tests, decision trees compare numeric features to threshold values, forming a hierarchical model that efficiently partitions data. Their conceptual rules are easier to interpret, and construct compared to complex neural networks, making them suitable for scenarios where interpretability is vital [10].

Various decision tree algorithms exist, each with unique characteristics and advantages, including ID3, C4.5, CART, CHAID, MARS, GUIDE, CTREE, CRUISE, and QUEST. These algorithms employ different strategies for tree construction, splitting criteria, and pruning methods, catering to diverse application scenarios and data types [10]. Entropy and information gain are crucial in decision tree construction. Entropy measures dataset impurity or randomness, with lower values indicating better homogeneity. Information gain measures the diminishment in entropy resulting from partitioning data based on a particular attribute, guiding the algorithm in selecting optimal split points [10]. The formulas for entropy and information gain are represented as follows:

$$\text{Entropy (S)} = \sum_{i=1}^{c} -P_i \log_2(P_i) \quad (4)$$

$$\text{Gain (S, A)} = \sum_{v \in V(A)} \frac{|S_v|}{|S|} \times \text{Entropy } (S_v) \quad (5)$$

Where S represents the dataset, A is an attribute, V(A) denotes the set of values that attribute A can tack, where S_v represents a subset of the dataset S that corresponds to the attribute value v, and P_i signifies the proportion of samples within the subset relative to the total number of samples associated with the i-th value of the attribute [10]. While decision trees are simple and interpretable, they can overfit and may not always yield optimal decision mechanisms. Decision trees with multiple layers can become complex, challenging interpretation and scalability, especially with large datasets [10].

3.4. Random Forest Classifier

Random Forest comprises a collection of tree-structured classifiers that operate independently of each other [12]. This ensemble classifier is constructed through a series of steps aimed at ensuring diversity among the base decision trees, ultimately leading to improved classification accuracy and generalization ability [11]. The construction process involves several key components, including the random sampling of training instances and the selection of subsets of features. These steps are essential to ensure that each decision tree operates independently. [12].

To build a Random Forest classifier, the following steps are typically followed:

1. An optimal choice for the parameter M, signifying the magnitude of elements contained within individual feature subsets, is determined. Subsequently, a new feature subset h_k is chosen randomly from the entire set of features, ensuring its independence from previously selected subsets [12].
2. The dataset is utilized to train decision trees using the selected feature subset for each group of training sets. Each individual classifier is denoted as $h(X, h_k)$, where X denotes the inputs [12].
3. Steps 1 and 2 are iterated until all feature subsets have been examined, resulting in the construction of a complete Random Forest classifier [12].

4. Upon inputting a test set, the class label of each sample is dictated by aggregating the voting outcomes derived from the classifications made by each individual decision tree [12].

The randomization approach employed in Random Forest construction involves selecting samples and feature subsets using bagging techniques, guaranteeing the autonomy of individual decision trees and enhancing classification precision [11, 12]. The parameter M, determining the size of feature subsets, greatly influences the robustness and interrelation of the Random Forest model. The optimal performance is achieved when M is close to the number of feature values (D) [12].

The ensemble nature of Random Forest allows for the parallelization of its creation process, leveraging the fast construction of individual decision trees to enhance classification speed [12]. Moreover, Random Forest classifiers offer built-in features such as error estimation using out-of-bag (OOB) data, computation of variable importance, and proximities for handling missing values and outliers [11].

3.5. Gradient Boosting Classifier

Gradient boosting, an extensively employed ensemble learning method, has attracted considerable interest in the ML domain owing to its capability to build robust predictive models [13, 14]. Gradient boosting stands out as an effective method for regression and classification tasks, aiming to approximate complex target functions by iteratively combining weak learners [13]. The fundamental principle of gradient boosting revolves around minimizing the expected value of a specified loss function $L(y, F(x))$, where $D = \{x_i, y_i\}_{i=1}^{N}$ represents the training dataset. Here, $\boldsymbol{x}$ denotes the input instances and $\boldsymbol{y}$ signifies their corresponding output values. The algorithm builds an additive approximation of the target function $F^*(x)$ by combining functions in a weighted sum form, represented as $F(x) = \sum_{m=1}^{M} p_m h_m(x)$ (x), where p_m signifies the weight of the m-th function. $h_m(x)$ often represented by base learners like decision trees [13].

The iterative process of gradient boosting begins with a constant approximation of $F^*(x)$, gradually refining it by adding models that minimize the residuals from previous iterations. During each iteration, a new dataset $D = \{x_i, r_{mi}\}_{i=1}^{N}$ is generated, where the pseudo-residuals r_{mi} are computed by evaluating the disparities between the actual labels and the current estimate $F(x)$. The weight p_m for each model is then determined through an optimization process, typically using a line search approach [13]. To prevent overfitting and ensure the stability of the iterative process, gradient boosting incorporates various regularization hyperparameters, including the learning rate, maximum tree depth, subsampling rate, number of features for splitting, and minimum samples for node splitting. These hyperparameters are pivotal in controlling the complexity of the trained models and improving generalization performance [13]. gradient boosting provides a robust framework for building ensemble models that excel in predictive accuracy.

3.6. Extreme Gradient Boosting (XGBoost) Classifier

XGBoost (Extreme Gradient Boosting) has emerged as a powerful ensemble ML algorithm, particularly well-suited for tasks involving classification and regression [13]. Originating from a research project at the University of Washington, XGBoost was quickly gaining prominence for its exceptional performance in various ML competitions and industry applications [15]. At its core, XGBoost leverages a gradient boosting method, mainly depending on decision trees as base learners to construct an ensemble model [13, 15]. Unlike traditional gradient boosting, which iteratively minimizes the loss function to refine the additive expansion of the objective function, XGBoost focuses exclusively on decision trees. It utilizes a modified form of the loss function to manage the complexity of the trees, ensuring that higher values of a regularization parameter (γ) lead to simpler trees [13].

One of the primary advantages of XGBoost lies in its ability to optimize the training process for decision trees, significantly enhancing computational efficiency and model performance [13, 15]. To achieve this, XGBoost implements several strategies:

- **Column-Based Storage Structure**, XGBoost utilizes a specific compressed column-based structure to store data pre-sorted, reducing the computational complexity of finding the best split for each node and enabling parallel processing of split candidates [13].
- **Subset-Based Split Finding**, rather than examining all feasible candidate splits, XGBoost utilizes a percentile-based approach for selecting splits, limiting the evaluation to only a subset of potential splits, further enhancing training speed while maintaining model accuracy [13].
- **Randomization Techniques**, XGBoost incorporates random subsampling techniques to train individual trees, and utilizes column subsampling at both tree and node levels, effectively reducing overfitting and improving training speed [13].

XGBoost provides a range of hyperparameters that can be adjusted to enhance model effectiveness, such as the learning rate and maximum tree depth, subsampling rate, and fraction of features to be evaluated at each split [13,15]. These hyperparameters play an essential role in controlling the complexity of the model and ensuring robust generalization performance across different datasets. Through its innovative optimization techniques and regularization strategies, XGBoost continues to push the boundaries of ML performance.

3.7. Light Gradient Boosting Machine (LightGBM) Classifier

LightGBM (Light Gradient Boosting Machine) is a cutting-edge gradient boosting algorithm known for its exceptional speed, memory efficiency, and accuracy [13, 16]. Developed by Microsoft in 2016, LightGBM distinguishes itself within the domain of ML algorithms due to its innovative techniques and customizable hyperparameters, which enable users to achieve superior model performance across diverse datasets and computational environments [16].

One of the key features of LightGBM is its utilization of histogram-based algorithms, which expedite the training process and diminish memory usage, making it particularly well-suited for handling large-scale datasets and distributed computing environments [16]. Additionally, LightGBM integrates two innovative methodologies, Gradient-based One Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) [13,16]. GOSS selectively retains instances with high gradients while discarding those with lower gradients, ensuring accurate information gain estimation during model training. On the other hand, the mathematical formulation of GOSS is represented by Equation (6), which estimates the variance gain over a subset of instances based on their gradients.

$$V_j\hat{}(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{n_l^j(d)}\right) + \left(\frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n_r^j(d)}\right) \quad (6)$$

where $V_j\hat{}(d)$ represents the estimated variance gain, A_l, A_r, B_l and B_r denote subsets of instances, and $\boldsymbol{a}$ and $\boldsymbol{b}$ are coefficients used for normalization [16].

On the other hand, EFB decreases model complexity by combining exclusive features into a unified feature, thereby enhancing computational efficiency without sacrificing predictive performance [16]. LightGBM provides users with a wide range of hyperparameters to fine-tune model performance. These include parameters such as the learning rate, maximum number of leaves in the tree, fractions of instances to be selected based on gradient magnitude, and the proportion of features to be assessed at each split. By carefully adjusting these hyperparameters, users can tailor the model to the specific characteristics of their dataset and optimize its predictive performance [13].

3.8. CatBoost Classifier

CatBoost, an abbreviation for Categorical Boosting, stands out as a potent gradient-boosting library designed to address the challenge of prediction shift encountered during training. Prediction shift refers to the deviation of predictions made during training from those made during testing, posing a significant obstacle to model accuracy [13]. The underlying mechanism of CatBoost involves building a series of base models per boosting iteration, each trained on a random permutation of training instances. This approach ensures independence from the initial permutation, enhancing model

robustness. Moreover, CatBoost employs symmetric trees or decision tables as base learners, facilitating efficient model training and interpretation [13].

The gradient boosting process employed in CatBoost generates a sequence of approximations $H^t: \mathbb{R}^m \rightarrow \mathbb{R}$ where t =0, 1…, in an iterative manner. Each subsequent approximation, H^t, builds upon the previous one H^{t-1} by incorporating a base predictor $g^t: \mathbb{R}^n \rightarrow \mathbb{R}$. This iterative process aims to minimize the expected loss function *L(H),* where *L* denotes a smooth loss function [17]. One of CatBoost's notable characteristics lies in its treatment of categorical attributes, which are replaced with numeric features representing the expected target value for each category [13]. This substitution strategy helps prevent overfitting to the training data. Additionally, CatBoost incorporates approaches such as Gradient-based Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) to further enhance training speed and model performance [13, 17]. In terms of hyperparameters, CatBoost offers flexibility with attributes such as learning rate, tree depth, number of gradient steps, and regularization coefficients. These parameters allow users to customize the model according to the specific requirements of their dataset and learning task [13].

3.9. Support Vector Machine (SVM)

Support Vector Machines (SVMs) have emerged as a pivotal algorithm in various ML applications. SVMs tackle the challenge of prediction shift by framing a convex quadratic optimization problem to attain a globally optimal solution, thereby circumventing the local extremum problem often encountered in other learning techniques [18]. SVM operates as a linear binary classifier, aiming to delineate a singular boundary between two classes. This linear SVM assumption relies on the premise that the multidimensional data exhibit linear separability within the input space. Through identifying an optimal hyperplane to segregate the dataset into discrete classes, SVMs aim to maximize the margin or separation between classes [18]. However, real-world data often presents challenges, as it is rarely linearly separable, necessitating adaptations to the standard linear SVM approach. To address this limitation, several techniques have been introduced such as the soft margin and kernel trick methods. The soft margin method introduces slack variables to handle non-separable data, while the kernel trick maps the feature space into a higher dimension to improve separability [18]. Through the kernel trick, SVMs employ a technique to map input datasets into higher-dimensional feature spaces, where samples are transformed to achieve linear separability. The efficacy of SVM hinges significantly on the selection of an appropriate kernel function, which generates dot products in higher-dimensional feature spaces. frequently used kernels, such as the Polynomial and Radial Basis Function (RBF) kernels, find extensive application to be applied. However, selecting the optimal kernel and its parameters can be challenging and computationally intensive, potentially leading to overfitting. Various methodologies, including automatic kernel selection and multiple kernel learning, have been proposed to mitigate this issue [18].

Furthermore, SVMs exhibit binary classification, which poses challenges in multiclass scenarios. To address this, multiclass tasks are decomposed into a series of binary classifications, employing strategies like one-against-one or one-against-all. SVMs can be expanded into one-shot multiclass classification, optimizing kernel parameters once for the entire classification task [18]. Despite presenting challenges such as kernel selection, parameter optimization, and complexity for non-expert users, SVMs retain their importance across diverse applications. With their memory-efficient nature, adaptability in decision-making boundaries, and suitability for high-dimensional spaces, SVMs make substantial contributions to the realm of ML [18].

3.10. Naive Bayes Classifier

The Naive Bayes classifier presents a straightforward yet powerful technique for representing probabilistic knowledge based on the Bayes theorem. Known for its "naive" designation, this classifier simplifies assumptions by treating predictive attributes as conditionally independent given the class and disregarding any hidden or latent attributes that might affect predictions [19]. There exist various variants of the Naive Bayes classifier, each tailored to different types of data. The Gaussian Naive Bayes classifier, for instance, assumes a Gaussian distribution and accommodates continuous data for classification [19]. Equation (7) outlines the probability calculation using the Gaussian distribution formula.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)}{2\sigma_y^2}\right) \quad (7)$$

Another variant, the Multinomial Naive Bayes classifier, is commonly employed in text classification tasks. This algorithm parametrizes the distribution of word vector counts and estimates parameters using maximum likelihood estimation with smoothing priors [19]. The Bernoulli Naive Bayes classifier operates on independent binary variables, making it suitable for scenarios where features represent the presence or absence of terms in documents [19]. Equation (8) outlines the decision criterion for the Bernoulli Naive Bayes classifier.

$$P(x_i|y) = P(i|y)x_i + (1 - P(i|y))(1 - x_i) \quad (8)$$

The probabilistic framework underpinning Naive Bayes classification, emphasizes the conditional probability theory [20]. The Naive Bayes classifier offers a versatile and efficient approach to classification tasks, with its various variants catering to different data types and applications. Through its probabilistic framework and straightforward assumptions, Naive Bayes continues to be a widely utilized tool in ML and data mining.

3.11. Linear Discriminant Analysis (LDA)

In the fields of statistics, pattern recognition, and ML, Linear Discriminant Analysis (LDA) is recognized as a versatile technique for finding combinations of linear features or separating multiple objects or events. It serves as a tool for classification and initial dimensionality reduction. The primary aim of LDA is to categorize objects into specific groups according to their defined attributes. Every object is characterized by two key variables: the associated class, considered as the dependent variable, and the unrelated attribute, acting as the independent variable. The dependent variable is intricately linked with the independent variable, which serves to delineate the distinctive features of the object [21]. To implement LDA, a learning step is essential to determine the discriminant function. This phase entails grouping the training data into a matrix representing each class (X_i). Subsequently, the mean matrix for each class (μ_i) is computed, along with the global average of all data matrices (μ). The next step involves calculating the covariance matrices (C_i) for individual classes and the overarching covariance matrix (C). The discriminant function (f_i) is then derived using these matrices and defined as in equation (9) [21].

$$f_i = \mu C^{-1} X_k^T - \frac{1}{2}\mu C^{-1}\mu i^T + \ln(p_i) \quad (9)$$

Once the discriminant function is established and its accuracy verified, it can be used to classify new objects. Through computing X_k^T and applying it to each class's discriminant function, the new object can be assigned to the corresponding class [21]. This process concludes the training phase and the discriminant function for classification purposes.

3.12. Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) emerges as an extension of Linear Discriminant Analysis (LDA), providing improved functionality for delineating non-linear data patterns. Both QDA and LDA serve as pivotal classifiers and dimensionality reduction techniques within the realm of statistical modeling [22]. QDA's distinct advantage lies in its flexibility regarding covariance matrix properties, allowing for the delineation of non-linear decision boundaries, a feature not readily achievable with LDA's linear boundaries. QDA is a univariate statistical technique utilized to develop a tailored algorithm aimed at identifying influential factors within the dataset [22].

Expanding upon this foundation, QDA is formally defined for binary classification scenarios where covariance matrices between classes are unequal, denoted as $\Sigma 1 \neq \Sigma 2$. Subsequently, equation (10) outlines the formulation of the QDA discriminant function, incorporating means (μ_1 and μ_2), covariance matrices (Σ_1 and Σ_2), and prior probabilities (π_1 and π_2) of the two classes involved in binary classification.

$$\delta(x) \coloneqq x^T(\Sigma_1 - \Sigma_2)^{-1}x + 2(\Sigma_2^{-1}\mu_2 - \Sigma_1^{-1}\mu_1)^T x + (\mu_1^t \Sigma_1^{-1}\mu_1 - \mu_2^t \Sigma_2^{-1}\mu_2) + \ln\left(\frac{|\Sigma_1|}{|\Sigma_2|}\right) + 2\ln\left(\frac{|\pi_2|}{|\pi_1|}\right) = 0, \quad (10)$$

The QDA algorithm encompasses several sequential steps, including inputting independent variables, computing class means and prior probabilities, calculating covariance matrices, determining the discriminant function, and ultimately assigning class labels to samples [23].

3.13. Passive Aggressive Classifier

The Passive-Aggressive Classifier, as depicted in the literature, operates as an online learning algorithm suitable for handling large-scale datasets. This classifier exhibits a unique behavior, remaining passive when the classification outcome is correct, and becoming aggressive in the event of misclassification or error. Unlike conventional algorithms, the Passive-Aggressive algorithm does not converge; instead, it aims to rectify errors while inducing minimal changes to the weight vector norm [24]. Like the perceptron model, the Passive-Aggressive algorithm does not necessitate a learning rate but integrates a regularization boundary. The algorithm operates in two modes: passive and aggressive, contingent upon the accuracy of predictions. In the passive mode, if the prediction aligns with the actual classification, no adjustments are made to the model. However, in the aggressive mode, adjustments are made to the model to correct misclassifications. This adjustment seeks to minimize the incurred loss, as denoted by the disparity between the predicted and actual classifications [25]. One key aspect of the Passive-Aggressive Classifier is its input, which typically comprises a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features. The input matrix serves as the foundation for training the model using a specified training dataset, with subsequent application on a test dataset to assess classifier performance [24].

3.14. AdaBoost Classifier

The AdaBoost Classifier, a renowned ensemble learning algorithm, seeks to elevate the performance of classification models by iteratively adjusting the weights assigned to training samples based on their classification outcomes [26, 27].

Initially, AdaBoost assigns equal weights to all training samples, creating a uniform distribution [27]. Subsequently, a weak learner, often a decision tree with limited depth, is trained on the dataset, and its performance is assessed by analyzing the classification error [26]. After each iteration of the AdaBoost algorithm weights of training samples are updated. This update mechanism increases the weights of misclassified samples, thereby focusing the subsequent training iterations on challenging examples. Spotting the light on the iterative nature of the AdaBoost algorithm, as multiple classifiers are trained sequentially on reweighted samples. The weight assigned to each classifier influences its contribution to the final classification decision [27].

At the conclusion of the iterative process, AdaBoost combines the predictions of all trained classifiers using weighted voting to generate the final hypothesis [27]. Equation (11) outlines the formula for calculating the final hypothesis, where the sign function determines the predicted class based on the weighted sum of individual classifier outputs. The AdaBoost classifier's iterative nature and weighted voting mechanism contribute to its effectiveness in handling complex classification tasks.

$$H(x) = \sin(\sum_{t=1}^{T} \alpha_t h_t\,(x)) \quad (11)$$

Where α_t represents the weight assigned to the t-th weak learner, $h_t\,(x)$ denotes the output of the t-th weak learner for input x, and T represents the total number of weak learners utilized in the AdaBoost algorithm [26].

3.15. Ridge Classifier

The Ridge Classifier serves as a method for regularization and feature selection, particularly effective in scenarios with highly correlated variables and missing data. This classifier employs a unique estimator known as the ridge estimator, which reduces coefficients to mitigate overfitting and complexity in the model. Operating under L2 regularization, the Ridge Classifier adds a penalty, termed the L2 penalty, computed by determining the squared magnitude of the coefficients. Ridge regression employs a

cost function that integrates this penalty, with the objective of minimizing the squared disparity between predicted and actual values, as illustrated in the equation (12) [28].

$$\sum_{i=1}^{m}(y_i - y_i^{\vee})^2 = \sum_{i=1}^{m}\left(y_i - \sum_{j=0}^{n} w_j * x_{ij}\right)^2 + \lambda \sum_{j=0}^{n} w_j^2 \qquad (12)$$

Here, m represents the number of instances, n denotes the number of features in the dataset, and λ denotes the regularization parameter. The addition of the penalty term enables the Ridge Classifier to effectively regularize coefficients, thereby reducing model complexity [28].

In addition, ridge regression can be applied to analyze multicollinearity data, ensuring that discrepancies are significant and mitigating biased estimations. Equation (13) outlines the ridge regression formula, incorporating the sum of squares of regression coefficients, observations number (n), and a tuning parameter (λ) controlling the strength of regularization [29].

$$Ridge = \frac{1}{2*n} * RSS + \lambda \sum(\beta)^2 \qquad (13)$$

The Ridge Classifier, through L2 regularization, estimates friction overlooked by the general model, thereby enhancing model performance and reliability. Through utilizing a method to designate data for L2 regularization output, the Ridge Classifier ensures improved predictive accuracy and robustness in analyzing datasets affected by multicollinearity [29].

4. Experiment Design

In this part of our study, a meticulous experiment design is debated to give details regarding the design of our experiments for building ML-based intrusion detection. This includes a comprehensive description of the datasets and their sources, implementation setup, and performance metrics.

4.1. Dataset description

Study experiments are carried out using two widely recognized datasets for our experimentation namely the NSL-KDD dataset and the Edge-IIoTset. First, The NSL-KDD dataset was introduced as a refinement to the widely used KDD99 dataset and has garnered significant attention in the cybersecurity research community since its proposal [30]. This dataset addresses several limitations identified in its predecessor, aiming to provide a more comprehensive and balanced platform for experimentation and evaluation in cybersecurity research. Comprising two main subsets, namely KDDTrain+ and KDDTest+, the NSL-KDD dataset offers a structured and standardized framework for conducting experiments in cyber threat detection and classification. The KDDTrain+ subset encapsulates 125,973 records, while the KDDTest+ subset contains 22,544 records, ensuring a substantial volume of data for rigorous analysis and evaluation [30]. Each record in the NSL-KDD dataset is characterized by 41 features, categorized into four distinct feature categories. Basic features, time-based Traffic features, connection-based Traffic features, and Content features [30]. These features encompass a wide range of attributes and parameters relevant to network traffic and communication, providing a holistic representation of network behavior and activity. One of the key aspects of the NSL-KDD dataset is its labeling scheme, which assigns one of 21 predefined label classes to each record, indicating whether it represents a normal network activity or a malicious attack [30]. It is noteworthy that the distribution of samples across different classes in the NSL-KDD dataset is not uniform, reflecting the inherent imbalance in real-world cybersecurity scenarios. While the training dataset encompasses 24 distinct types of attacks, the testing dataset includes an additional 14 types of network intrusions that are not present in the training set [30]. Statistical analysis of the NSL-KDD dataset, encapsulating vital attributes such as the number of instances, features, and class distributions, is presented in Table 1.

Table 1: Statistical analysis of the NSL-KDD dataset

	Count	Mean	Std	Min	25%	50%	75%	Max
Duration	125972.0	287.146929	2.604526e+03	0.0	0.00	0.00	0.00	4.290800e+04
src_bytes	125972.0	45567.1008	5.870354e+06	0.0	0.00	44.00	276.00	1.379964e+09

dst_bytes	125972.0	19779.2714	4.021285e+06	0.0	0.00	0.00	516.00	1.309937e+09
Land	125972.0	0.000198	1.408613e-02	0.0	0.00	0.00	0.00	1.000000e+00
wrong_fragment	125972.0	0.022688	2.535310e-01	0.0	0.00	0.00	0.00	3.000000e+00
Urgent	125972.0	0.000111	1.436608e-02	0.0	0.00	0.00	0.00	3.000000e+00
Hot	125972.0	0.204411	2.149977e+00	0.0	0.00	0.00	0.00	7.700000e+01
num_failed_logins	125972.0	0.001222	4.523932e-02	0.0	0.00	0.00	0.00	5.000000e+00
logged_in	125972.0	0.395739	4.890107e-01	0.0	0.00	0.00	1.00	1.000000e+00
num_compromised	125972.0	0.279253	2.394214e+01	0.0	0.00	0.00	0.00	7.479000e+03
root_shell	125972.0	0.001342	3.660299e-02	0.0	0.00	0.00	0.00	1.000000e+00
su_attempted	125972.0	0.001103	4.515456e-02	0.0	0.00	0.00	0.00	2.000000e+00
num_root	125972.0	0.302194	2.439971e+01	0.0	0.00	0.00	0.00	7.468000e+03
num_file_creations	125972.0	0.012669	4.839370e-01	0.0	0.00	0.00	0.00	4.300000e+01
num_shells	125972.0	0.000413	2.218122e-02	0.0	0.00	0.00	0.00	2.000000e+00
num_access_files	125972.0	0.004096	9.936995e-02	0.0	0.00	0.00	0.00	9.000000e+00
num_outbound_cmds	125972.0	0.000000	0.000000e+00	0.0	0.00	0.00	0.00	0.000000e+00
is_host_login	125972.0	0.000008	2.817494e-03	0.0	0.00	0.00	0.00	1.000000e+00
is_guest_login	125972.0	0.009423	9.661271e-02	0.0	0.00	0.00	0.00	1.000000e+00
Count	125972.0	84.108207	1.145088e+02	0.0	2.00	14.00	143.0	5.110000e+02
srv_count	125972.0	27.738093	7.263609e+01	0.0	2.00	8.00	18.00	5.110000e+02
serror_rate	125972.0	0.284487	4.464567e-01	0.0	0.00	0.00	1.00	1.000000e+00
srv_serror_rate	125972.0	0.282488	4.470236e-01	0.0	0.00	0.00	1.00	1.000000e+00
rerror_rate	125972.0	0.119959	3.204366e-01	0.0	0.00	0.00	0.00	1.000000e+00
srv_rerror_rate	125972.0	0.121184	3.236483e-01	0.0	0.00	0.00	0.00	1.000000e+00
same_srv_rate	125972.0	0.660925	4.396236e-01	0.0	0.09	1.00	1.00	1.000000e+00
diff_srv_rate	125972.0	0.063053	1.803150e-01	0.0	0.00	0.00	0.06	1.000000e+00
srv_diff_host_rate	125972.0	0.097322	2.598314e-01	0.0	0.00	0.00	0.00	1.000000e+00
dst_host_count	125972.0	182.14920	9.920657e+01	0.0	82.00	255.0	255.0	2.550000e+02
dst_host_srv_count	125972.0	115.65372	1.107029e+02	0.0	10.0	63.00	255.0	2.550000e+02
dst_host_same_srv_rate	125972.0	0.521244	4.489501e-01	0.0	0.05	0.51	1.00	1.000000e+00
dst_host_diff_srv_rate	125972.0	0.082952	1.889225e-01	0.0	0.00	0.02	0.07	1.000000e+00
dst_host_same_src_port_rate	125972.0	0.148379	3.089984e-01	0.0	0.00	0.00	0.06	1.000000e+00
dst_host_srv_diff_host_rate	125972.0	0.032543	1.125642e-01	0.0	0.00	0.00	0.02	1.000000e+00
dst_host_serror_rate	125972.0	0.284455	4.447851e-01	0.0	0.00	0.00	1.00	1.000000e+00
dst_host_srv_serror_rate	125972.0	0.278487	4.456702e-01	0.0	0.00	0.00	1.00	1.000000e+00
dst_host_rerror_rate	125972.0	0.118832	3.065586e-01	0.0	0.00	0.00	0.00	1.000000e+00
dst_host_srv_rerror_rate	125972.0	0.120241	3.194605e-01	0.0	0.00	0.00	0.00	1.000000e+00
Level	125972.0	19.504056	2.291512e+00	0.0	18.00	20.00	21.00	2.100000e+01

On the other hand, the Edge-IIoTset dataset was proposed to cover the need for realistic security data in edge-based IoT environments [31]. In the Edge-IIoTset dataset, the data was acquired using a comprehensive testbed spanning seven layers. At the coarse-grained level, the data are broadly divided into normal traffic and attack traffic [31]. Encompassing a total of 2,219,201 samples, this dataset serves as a robust repository for analyzing cyber threats in the context of industrial IoT environments. Of these

samples, a majority, comprising 1,615,643 instances, are classified as normal, representing typical communication patterns within IoT and IIoT networks. Contrastingly, the dataset also includes 603,558 samples associated with 14 distinct attack scenarios, offering valuable insights into the vulnerabilities and potential security breaches prevalent in such systems [32]. Each sample in the Edge-IIoTset dataset is characterized by an extensive set of features extracted from network traffic data. These features are carefully selected to capture diverse aspects of network communication and behavior, facilitating nuanced analysis and robust model development [31]. Table 2 presents a statistical analysis of the Edge-IIoTset dataset, providing insights into its composition and characteristics.

Table 2: Statistical analysis of the Edge-IIoTset dataset

	count	mean	std	min	25%	50%	75%	max
arp. opcode	2219201.0	3.323268e-03	6.843237e-02	0.0	0.0	0.000000e+00	0.000000e+00	2.000000e+00
arp.hw.size	2219201.0	1.582732e-02	3.077555e-01	0.0	0.0	0.000000e+00	0.000000e+00	6.000000e+00
icmp.checksum	2219201.0	1.730285e+03	8.526581e+03	0.0	0.0	0.000000e+00	0.000000e+00	6.553300e+04
icmp.seq_le	2219201.0	1.893064e+03	8.870474e+03	0.0	0.0	0.000000e+00	0.000000e+00	6.553500e+04
icmp.transmit_timestamp	2219201.0	2.877556e+03	4.705188e+05	0.0	0.0	0.000000e+00	0.000000e+00	7.728902e+07
icmp.unused	2219201.0	0.000000e+00	0.000000e+00	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00
http.content_length	2219201.0	4.808231e+00	9.642259e+01	0.0	0.0	0.000000e+00	0.000000e+00	8.365500e+04
http.response	2219201.0	1.469132e-02	1.203142e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
http.tls_port	2219201.0	0.000000e+00	0.000000e+00	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00
tcp.ack	2219201.0	2.278400e+07	1.649033e+08	0.0	1.0	6.000000e+00	5.900000e+01	3.949529e+09
tcp.ack_raw	2219201.0	1.573687e+09	1.337361e+09	0.0	42609615.0	1.426945e+09	2.506984e+09	4.294947e+09
tcp.checksum	2219201.0	2.897927e+04	2.065386e+04	0.0	9951.0	2.843400e+04	4.699400e+04	6.553500e+04
tcp.connection.fin	2219201.0	8.686910e-02	2.816432e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
tcp.connection.rst	2219201.0	9.222779e-02	2.893473e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
tcp.connection.syn	2219201.0	7.060199e-02	2.561589e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
tcp.connection.synack	2219201.0	4.524016e-02	2.078305e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
tcp.dstport	2219201.0	2.626435e+04	2.750352e+04	0.0	1883.0	4.321000e+03	5.632200e+04	6.553500e+04
tcp.flags	2219201.0	1.437637e+01	8.232028e+00	0.0	4.0	1.600000e+01	2.000000e+01	2.500000e+01
tcp.flags.ack	2219201.0	7.385613e-01	4.394185e-01	0.0	0.0	1.000000e+00	1.000000e+00	1.000000e+00
tcp.len	2219201.0	1.214432e+02	4.996815e+02	0.0	0.0	0.000000e+00	4.000000e+00	6.522800e+04
tcp.seq	2219201.0	8.682862e+06	3.476928e+07	0.0	1.0	5.000000e+00	5.900000e+01	4.294967e+09
udp.port	2219201.0	2.867523e+01	1.185792e+03	0.0	0.0	0.000000e+00	0.000000e+00	6.097600e+04
udp.stream	2219201.0	7.198890e+04	3.654767e+05	0.0	0.0	0.000000e+00	0.000000e+00	2.898776e+06
udp.time_delta	2219201.0	3.541572e-01	1.153177e+01	0.0	0.0	0.000000e+00	0.000000e+00	5.290000e+02
dns.qry.name	2219201.0	7.374416e+03	1.191786e+05	0.0	0.0	0.000000e+00	0.000000e+00	2.898522e+06
dns.qry.qu	2219201.0	7.247068e-02	6.182116e+00	0.0	0.0	0.000000e+00	0.000000e+00	1.028000e+03
dns.qry.type	2219201.0	0.000000e+00	0.000000e+00	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00
dns.retransmission	2219201.0	1.179884e-02	5.641567e-01	0.0	0.0	0.000000e+00	0.000000e+00	2.800000e+01
dns.retransmit_request	2219201.0	2.208002e-05	4.698887e-03	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
dns.retransmit_request_in	2219201.0	1.802451e-05	4.245490e-03	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
mqtt.conflag.cleansess	2219201.0	3.741662e-02	1.897805e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00
mqtt.conflags	2219201.0	7.483324e-02	3.795610e-01	0.0	0.0	0.000000e+00	0.000000e+00	2.000000e+00

mqtt.hdrflags	2219201.0	1.197063e+01	4.321738e+01	0.0	0.0	0.000000e+00	0.000000e+00	2.240000e+02
mqtt.len	2219201.0	1.982731e+00	7.648797e+00	0.0	0.0	0.000000e+00	0.000000e+00	3.900000e+01
mqtt.msg_decoded_as	2219201.0	0.000000e+00	0.000000e+00	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00
mqtt.msgtype	2219201.0	7.481643e-01	2.701086e+00	0.0	0.0	0.000000e+00	0.000000e+00	1.400000e+01
mqtt.proto_len	2219201.0	1.496665e-01	7.591219e-01	0.0	0.0	0.000000e+00	0.000000e+00	4.000000e+00
mqtt.topic_len	2219201.0	8.977934e-01	4.554231e+00	0.0	0.0	0.000000e+00	0.000000e+00	2.400000e+01
mqtt.ver	2219201.0	1.496665e-01	7.591219e-01	0.0	0.0	0.000000e+00	0.000000e+00	4.000000e+00
mbtcp.len	2219201.0	1.297764e-03	1.711483e-01	0.0	0.0	0.000000e+00	0.000000e+00	2.700000e+01
mbtcp.trans_id	2219201.0	5.170780e-03	7.226807e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.510000e+02
-mbtcp.unit_id	2219201.0	9.417804e-05	1.377313e-02	0.0	0.0	0.000000e+00	0.000000e+00	6.000000e+00
Attack_label	2219201.0	2.719709e-01	4.449751e-01	0.0	0.0	0.000000e+00	1.000000e+00	1.000000e+00

4.2. Implementation setups

For our implementation setup, we utilized a system operating on Microsoft Windows 10 Pro version 10.0.19045, with specifications incorporating an Intel64 Family 6 Model 60 Stepping 3 Genuine Intel processor clocked at approximately 2800 Mhz and a total physical memory of 8,073 MB. The system, an HP ZBook 15 G2, provided a stable and capable environment for our experimentation. Furthermore, we configured our software environment with Python version 3.9.1 and scikit-learn version 1.4.2. Leveraging the capabilities of Python and scikit-learn, we ensured compatibility with the latest tools and libraries for ML experimentation. This setup allowed us to efficiently develop and evaluate our intrusion detection algorithms while maintaining consistency and reproducibility throughout the experimentation process.

4.3. Evaluations Measures

4.3.1 Accuracy

Accuracy stands as a broadly applied metric in multi-class classification, serving as an indicator of the model's proficiency in correctly assigning classes to individual data units. Its derivation hinges on insights stemming from the confusion matrix, a tabular representation detailing both the accurate and inaccurate classifications made by the model [33]. The accuracy formula accentuates the significance of true positives and true negatives, which signify successfully identified instances, against the backdrop of false positives and false negatives, indicating misclassifications. While accuracy offers a straightforward measure of model performance, its utility is raised in scenarios where the primary concern centers on individual data units rather than class distribution [33].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \qquad (14)$$

Accuracy metrics Spanning a scale from 0 to 1, higher values connote enhanced accuracy. However, its suitability diminishes in the context of imbalanced datasets, where certain classes are disproportionately represented. In such instances, accuracy risks obscuring errors in minority classes, as their contribution to overall accuracy diminishes [33].

4.3.2 Precision

The precision metric is fundamental in assessing the performance of a classifier. It denotes the ratio of accurately predicted positive instances among all instances classified as positive by the model. In simpler terms, precision represents the count of truly positive instances among all instances predicted as positive [33]. The formula for precision, as derived from the confusion matrix, is given by:

$$Precision = \frac{TP}{TP + FP} \qquad (15)$$

Here, TP denotes the count of true positive instances (instances accurately classified as positive), while FP indicates the count of false positive instances (instances erroneously classified as positive) [33].

4.3.3 Recall

The recall metric complements precision by focusing on the model's capacity to accurately recognize positive instances. It quantifies the ratio of correctly classified positive instances to the total number of actual positive instances within the dataset. In other words, recall represents, the number of actual positive instances did the model identify out of all instances [33]. The formula for recall, derived from the confusion matrix, is as follows:

$$Recall = \frac{TP}{TP+FN} \qquad (16)$$

Where TP indicates true positive instances (correctly classified positive instances), and FN indicates false negative instances (positive instances incorrectly classified as negative). Recall measures the model's effectiveness in capturing all positive instances, without missing any [33].

4.3.4 F1-score

In the context of classification model evaluation, accuracy alone may not suffice, prompting a deeper exploration of metrics like the F1-Score. This metric, rooted in the intricacies of the confusion matrix, consolidates Precision and Recall, encapsulating their essence through the notion of harmonic mean [33].

The F1-Score emerges as a pivotal amalgamation of Precision and Recall, offering a nuanced evaluation of model performance. Its formulation as the harmonic mean highlights the delicate equilibrium between Precision and Recall, with a perfect score of 1 signaling flawless performance and a score of 0 indicating suboptimal classification accuracy as presented in equation (17) [33].

$$F1 - Score = \left(\frac{2}{precision^{-1}+recall^{-1}}\right) = 2 . \frac{precision . recall}{precision + recall} \qquad (17)$$

This metric impartially considers Precision and Recall, irrespective of class distribution disparities. Notably, smaller classes wield disproportionate influence, with the harmonic mean favoring models that exhibit uniform Precision and Recall profiles [33].

4.3.5 AUC

The area under the curve (AUC) metric, derived from the receiver operating characteristic (ROC) curve and a decision function (f), provides perspective on the trade-off between false positives and true positives by varying the threshold for $f(x)$. It furnishes a graphical depiction illustrating the quantity of true positives contrasted with false positives. The AUC, representing the area under the ROC curve, quantifies the likelihood that the decision function $f(x)$ assigns a greater value to a randomly chosen positive example than to a negative one [34].

Calculating the AUC involves estimating the true distribution of positive and negative instances using a sample. Equation (18) provides a maximum likelihood estimate of the true AUC based on the number of positive and negative samples. It examines all pairs of positive and negative instances, contributing to the overall AUC performance when $(f(x_+) > f(x_-))$ [34].

$$AUC(f) = \Pr(f(x_+) > f(x_-)) \qquad (18)$$

Maximizing the AUC involves maximizing the number of pairs meeting the condition $(f(x_+) > f(x_-))$. The AUC remains impartial to class distribution and impervious to decision thresholds. In practice, the unthresholded model $f(x) = (w, \emptyset(x))$ is often employed, where $\emptyset : X \to F$ signifies an implicit preprocessing of input data, mapping it into a feature space F. This feature space facilitates learning through kernel functions, streamlining the learning process without the need for explicit feature representations [35].

5. Results & Discussions

In this section, we showcase the outcomes derived from our experimental endeavors with various ML approaches for intrusion detection. Our objective was to establish reproducible baselines that serve as fundamental benchmarks for assessing the performance of several models in detecting intrusions effectively.

Table 3 furnishes a thorough summary of the quantitative results obtained from the evaluation of the collection of ML algorithms on the NSL-KDD dataset. Each algorithm was meticulously trained, validated, and tested using standardized procedures to guarantee the reproducibility and reliability of the findings. The table offers a comparative examination of crucial performance indicators across different algorithms.

Table 3. Numerical results of ML algorithms on NSL-KDD dataset (Binary Classification scenario).

ML algorithm	**Accuracy**	**Precision**	**Recall**	**F1-score**	**AUC**
Logistic Regression	0.89	0.87	0.89	0.88	0.90
Decision Tree Classifier	0.89	0.97	0.80	0.88	0.90
Random Forest Classifier	1.00	1.00	1.00	1.00	1.00
K-nearest neighbors (KNN)	0.99	1.00	0.99	0.99	1.00
SVM Classifier	0.98	0.97	0.98	0.98	0.98
Gradient Boosting Classifier	1.00	1.00	1.00	1.00	1.00
Extreme Gradient Boosting (XGBoost)	1.00	1.00	1.00	1.00	1.00
Light Gradient Boosting Machine (LGBM)	1.00	1.00	1.00	1.00	1.00
CatBoost Classifier	1.00	1.00	1.00	1.00	1.00
Naive Bayes Classifier	0.96	0.94	0.98	0.96	0.97
Linear Discriminant Analysis (LDA)	0.98	0.98	0.99	0.98	1.00
Quadratic Discriminant Analysis (QDA)	0.99	0.97	1.00	0.99	0.99
Passive Aggressive Classifier	0.98	0.98	0.97	0.97	0.97
AdaBoost Classifier	1.00	1.00	1.00	1.00	1.00
Ridge Classifier	0.98	0.98	0.99	0.98	1.00

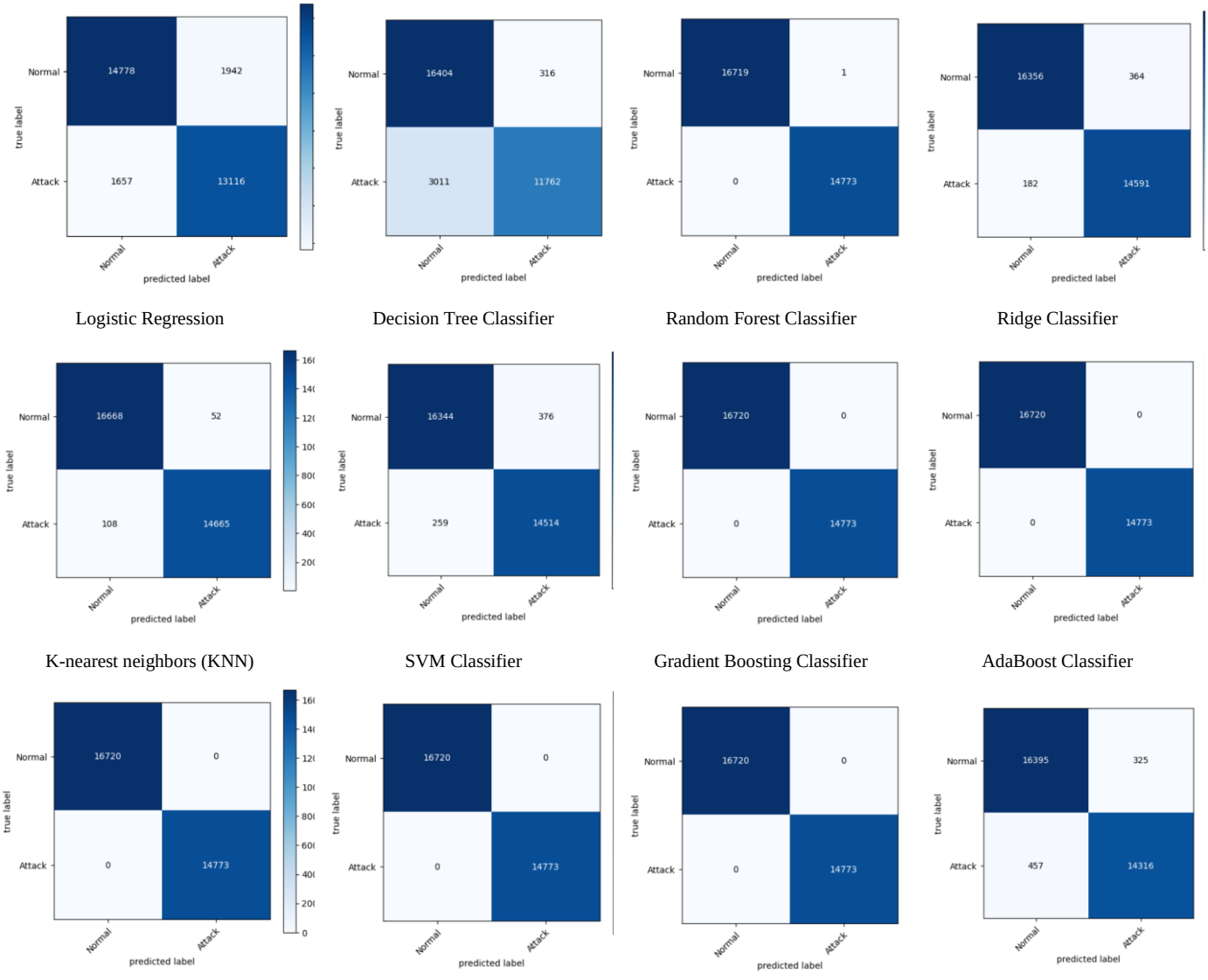
Table 4 unveils the quantitative results stemming from our meticulous evaluation of diverse ML algorithms on the Edge-IIoTset dataset, shedding light on their efficacy in intrusion detection within Industrial Internet of Things (IIoT) environments. Through thorough experimentation and analysis, we meticulously assessed the performance of each algorithm across key benchmarks. The dataset's unique characteristics, including the dynamic nature of IIoT networks and the inherent complexities of industrial environments, pose distinct challenges for intrusion detection systems. Consequently, the algorithms' performance on Edge-IIoTset offers valuable insights into their adaptability and robustness in confronting real-world scenarios characterized by diverse network traffic patterns and security threats. Our comparative analysis in Table 4 serves as a pivotal reference point for stakeholders seeking to deploy intrusion detection mechanisms tailored to IIoT infrastructures, facilitating informed decision-making, and fostering advancements in cybersecurity practices within industrial domains.

Table 4. Numerical results of ML algorithms on Edge-IIoTset dataset (Binary Classification scenario).

ML algorithm	Accuracy	Precision	Recall	F1-score	AUC
Logistic Regression	0.89	0.89	1.00	0.94	0.87
Decision Tree Classifier	0.89	0.88	1.00	0.94	0.82
Random Forest Classifier	1.00	1.00	1.00	1.00	1.00
K-nearest neighbors (KNN)	1.00	1.00	1.00	1.00	1.00
SVM Classifier	0.86	0.88	0.97	0.92	0.82
Gradient Boosting Classifier	1.00	1.00	1.00	1.00	1.00
Extreme Gradient Boosting (XGBoost) Classifier	1.00	1.00	1.00	1.00	1.00
Light Gradient Boosting Machine (LGBM) Classifier	1.00	1.00	1.00	1.00	1.00

CatBoost Classifier	1.00	1.00	1.00	1.00	1.00
Naive Bayes Classifier	0.44	1.0	0.34	0.51	0.73
Linear Discriminant Analysis (LDA)	0.89	0.88	1.00	0.94	0.86
Quadratic Discriminant Analysis (QDA)	0.47	1.00	0.37	0.54	0.93
Passive Aggressive Classifier	0.88	0.96	0.96	0.93	0.90
AdaBoost Classifier	1.00	1.00	1.00	1.00	1.00
Ridge Classifier	0.89	0.88	1.00	0.94	0.86

Figure 1 encapsulates the quantitative results derived from our comprehensive evaluation of ML algorithms in a scenario involving binary classification using the NSL-KDD dataset. This visualization offers a succinct yet informative depiction of each algorithm's performance across critical benchmarks. The figure not only provides a comparative overview of algorithm performance but also elucidates the interplay between different metrics, offering valuable insights into the trade-offs inherent in optimizing detection capabilities while minimizing false positives and negatives. Furthermore, the inclusion of this binary classification scenario enables a focused examination of algorithmic robustness in distinguishing between benign and malicious network behaviors, thereby enhancing our understanding of their suitability for real-world deployment in cybersecurity applications.



Logistic Regression | Decision Tree Classifier | Random Forest Classifier | Ridge Classifier

K-nearest neighbors (KNN) | SVM Classifier | Gradient Boosting Classifier | AdaBoost Classifier

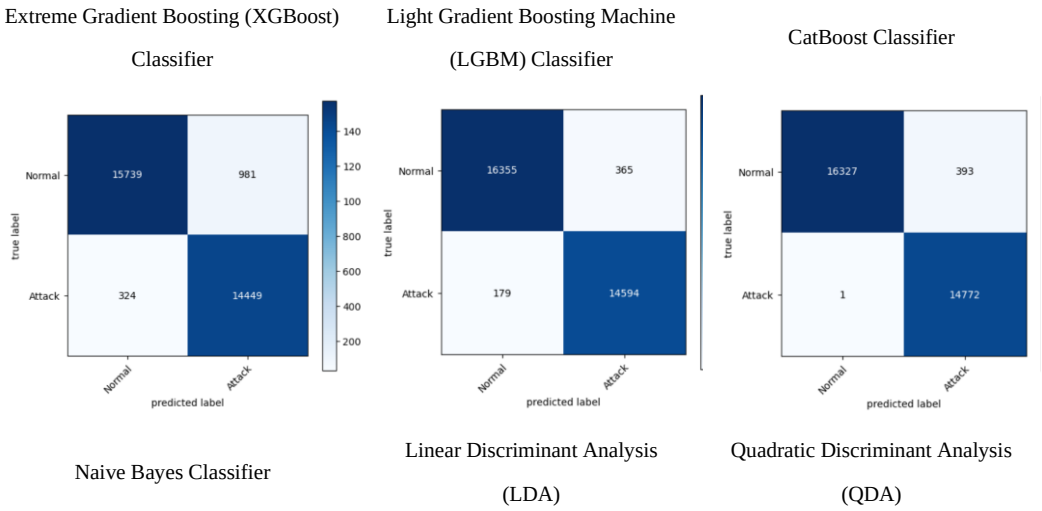
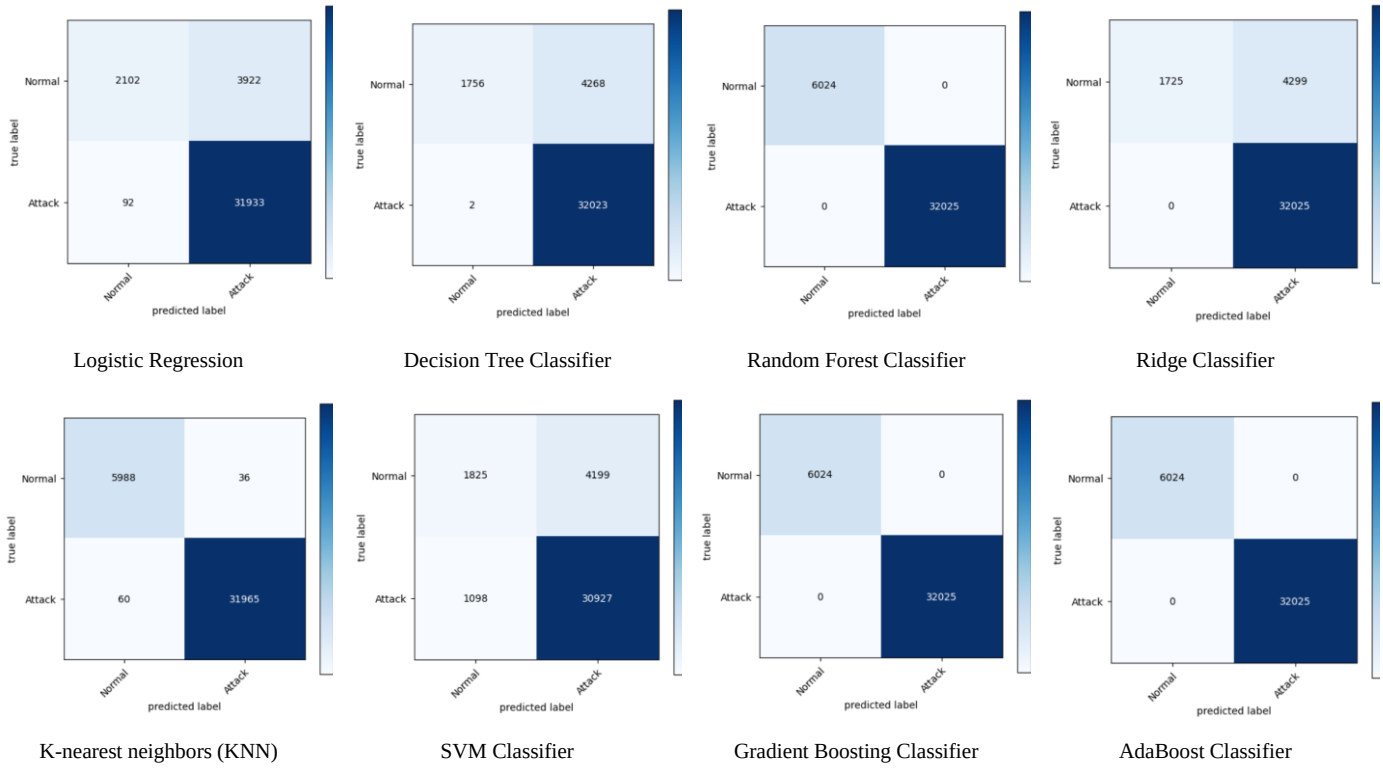
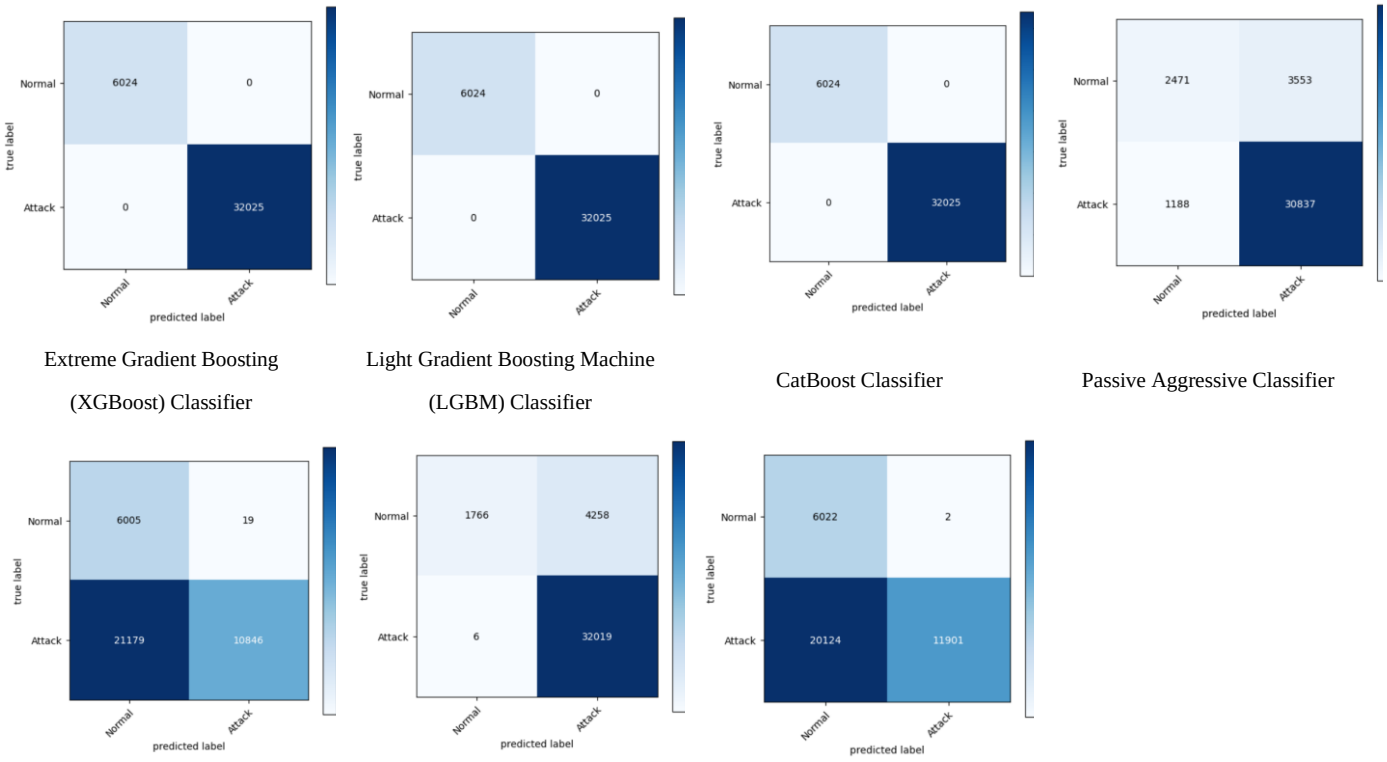


Figure 1. Confusion Matrices of ML algorithms on NSL-KDD dataset (Binary Classification scenario)

Figure 2 provides a detailed comparison of confusion matrices obtained from the evaluation of various ML algorithms on the Edge-IIoTset dataset in a scenario involving binary classification. The confusion matrices offer a granular insight into the performance of each algorithm by visualizing the distribution of true positives, false positives, true negatives, and false negatives. This visual representation not only facilitates a comprehensive understanding of algorithmic behavior but also highlights the intricacies of intrusion detection within IIoT environments. Furthermore, the comparison of confusion matrices allows for the identification of specific areas of strength and weakness across different algorithms, thereby informing targeted strategies for optimization and improvement.





Extreme Gradient Boosting (XGBoost) Classifier

Light Gradient Boosting Machine (LGBM) Classifier

CatBoost Classifier

Passive Aggressive Classifier

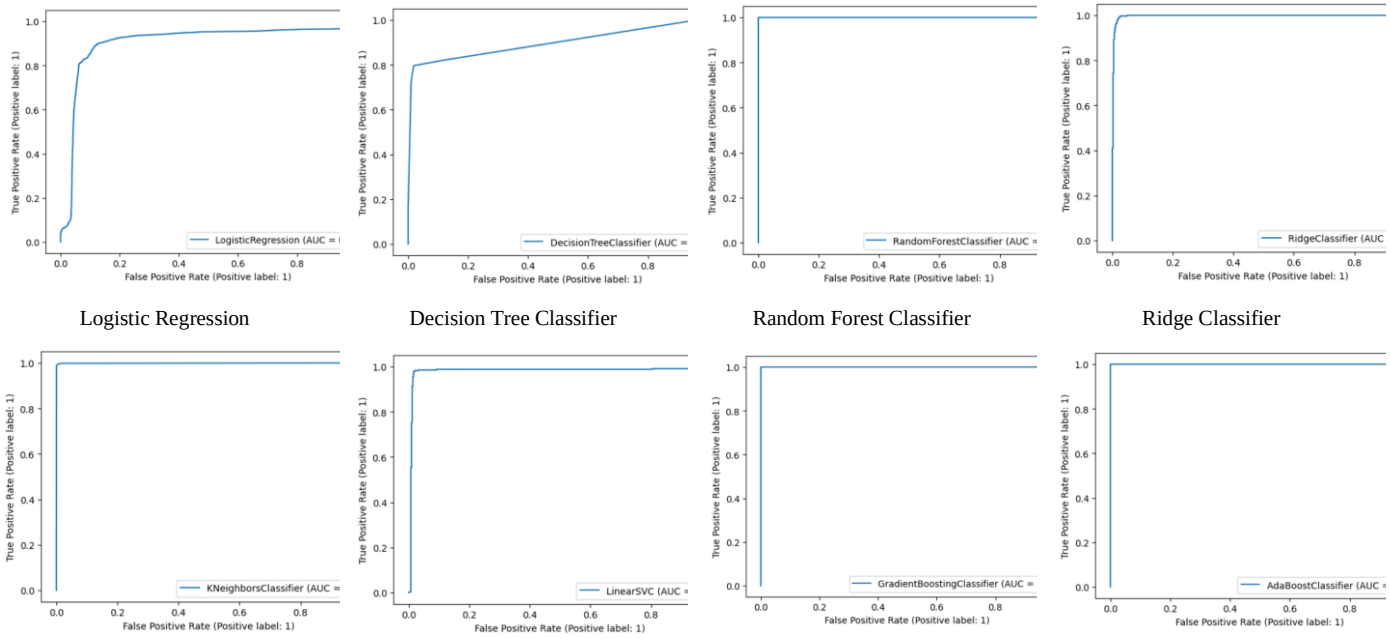
Naive Bayes Classifier

Linear Discriminant Analysis (LDA)

Quadratic Discriminant Analysis (QDA)

Figure 2. Confusion Matrices of ML algorithms on Edge-IIoTset dataset (Binary Classification scenario)

A thorough comparison of ROCAUC curves obtained from testing several ML methods in a binary classification situation using the NSL-KDD dataset is presented in Figure 3. These ROC curves give insights into each algorithm's discriminatory strength across a range of thresholds by graphically representing how well it performs in differentiating between normal and intrusive network activity.



Logistic Regression

Decision Tree Classifier

Random Forest Classifier

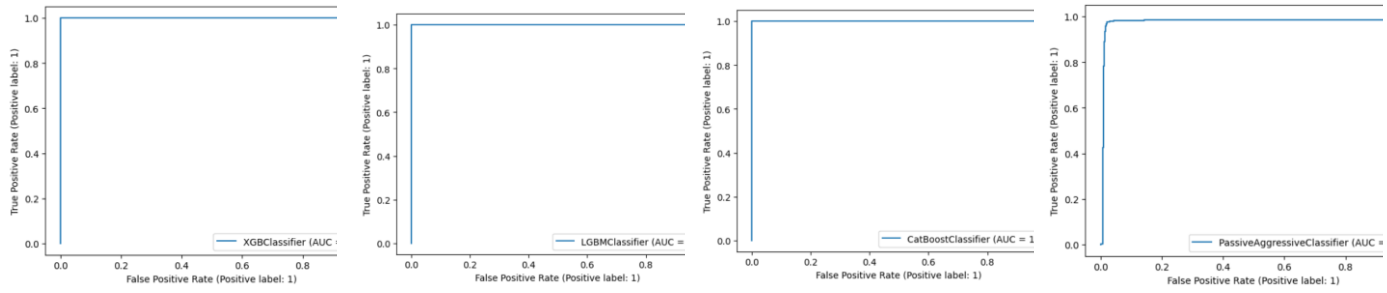
Ridge Classifier

K-nearest neighbors (KNN)

SVM Classifier

Gradient Boosting Classifier

AdaBoost Classifier

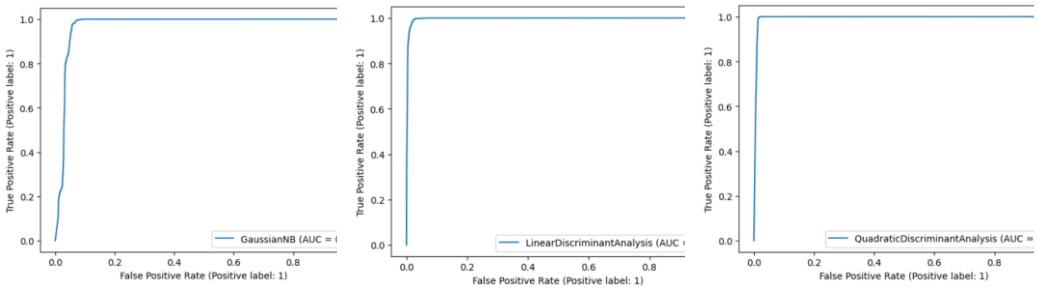


Extreme Gradient Boosting (XGBoost) Classifier

Light Gradient Boosting Machine (LGBM) Classifier

CatBoost Classifier

Passive Aggressive Classifier



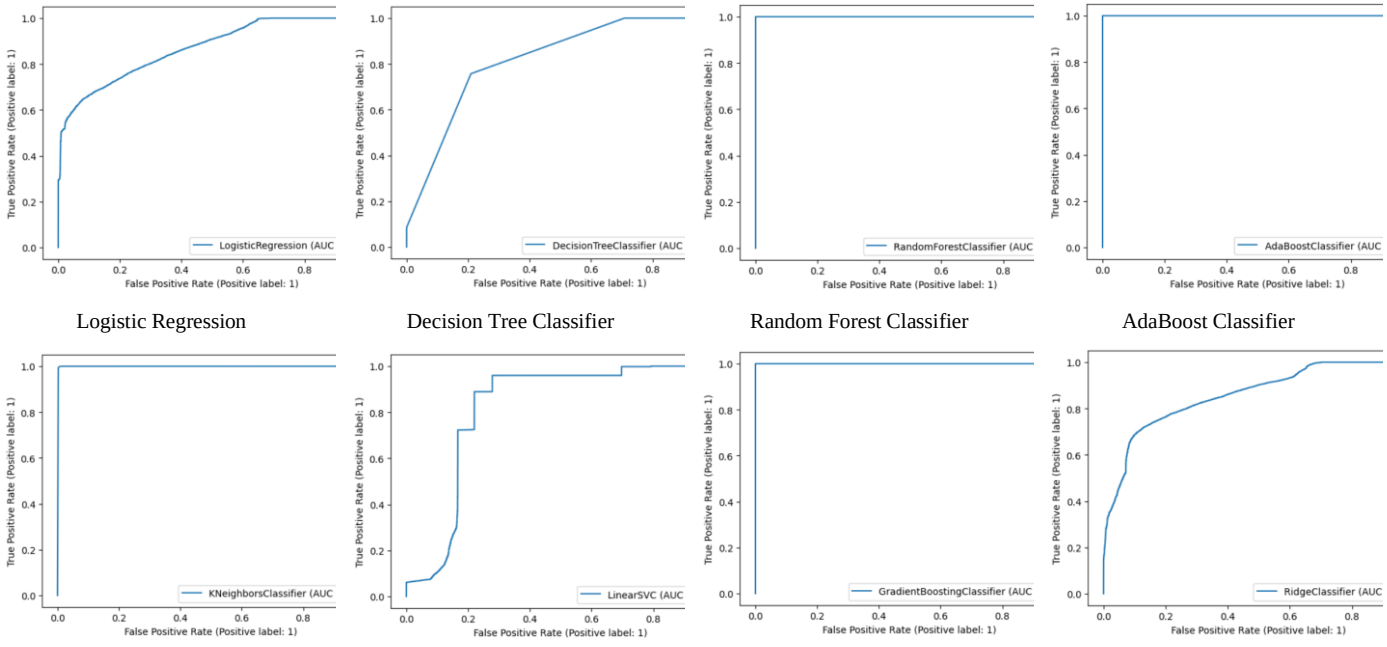
Naive Bayes Classifier

Linear Discriminant Analysis (LDA)

Quadratic Discriminant Analysis (QDA)

Figure 3. visualizations of ROC curves of ML algorithms on NSL-KDD dataset (Binary Classification scenario)

Figure 4 provides a comprehensive comparison of ROC curves derived from the evaluation of diverse ML algorithms on the Edge-IIoTset dataset in a binary classification scenario. These ROC curves serve as graphical representations of each algorithm's performance in distinguishing between normal and intrusive network activities, offering insights into their discriminatory power across varying thresholds. The AUC-ROC serves as a key metric for assessing algorithmic efficacy, with higher AUC-ROC values indicating superior performance in correctly identifying instances of intrusion while minimizing false positives. Furthermore, the comparative analysis of ROC curves facilitates a nuanced understanding of algorithmic strengths and weaknesses, enabling informed decision-making in the selection and deployment of intrusion detection mechanisms within industrial domains [35].



Logistic Regression

Decision Tree Classifier

Random Forest Classifier

AdaBoost Classifier

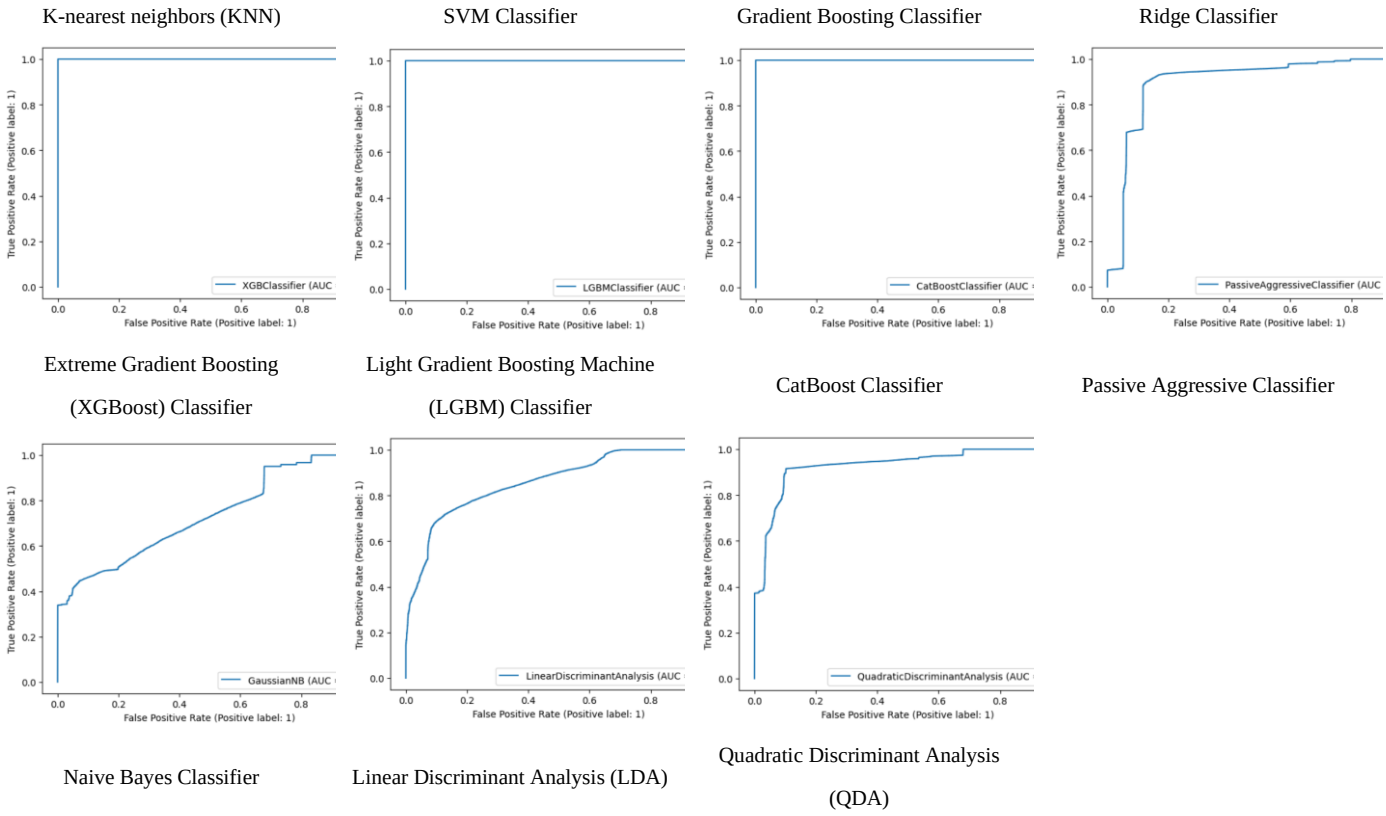


Figure 4. ROC curves of ML algorithms on Edge-IIoTset dataset (Binary Classification scenario)

Table 5 presents a detailed analysis of the numerical results from evaluating various ML algorithms on the NSL-KDD dataset in a multi-class classification scenario. Each algorithm underwent rigorous training and testing techniques to ensure the consistency and reliability of the results. The table facilitates a comparative assessment of key performance metrics, including accuracy, precision, recall, F1-score, and the (AUC-ROC). This thorough examination across multiple algorithms highlights their effectiveness in handling multi-class classification tasks, providing critical insights into their robustness and overall performance.

Table 5. Numerical results of ML algorithms on NSL-KDD dataset (Multi-Class Classification scenario).

ML algorithm	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Micro-Average AUC**
Logistic Regression	0.87	0.87	0.87	0.87	0.93
Decision Tree Classifier	0.83	0.82	0.83	0.80	0.98
Random Forest Classifier	1.00	1.00	1.00	1.00	1.00
K-nearest neighbors (KNN)	0.99	0.99	0.99	0.99	1.00
SVM Classifier	0.91	0.92	0.91	0.87	0.99
Gradient Boosting Classifier	1.00	1.00	1.00	1.00	1.00
Extreme Gradient Boosting (XGBoost)	1.00	1.00	1.00	1.00	1.00
Light Gradient Boosting Machine (LGBM)	0.97	0.97	0.97	0.97	0.98
CatBoost Classifier	1.00	1.00	1.00	1.00	1.00
Naive Bayes Classifier	0.51	0.76	0.51	0.46	0.88
Linear Discriminant Analysis (LDA)	0.97	0.98	0.97	0.98	1.00
Quadratic Discriminant Analysis (QDA)	0.94	0.94	0.94	0.94	0.99
Passive Aggressive Classifier	0.53	0.28	0.53	0.37	0.93
AdaBoost Classifier	1.00	1.00	1.00	1.00	1.00

Ridge Classifier	1.00	1.00	1.00	0.99	1.00

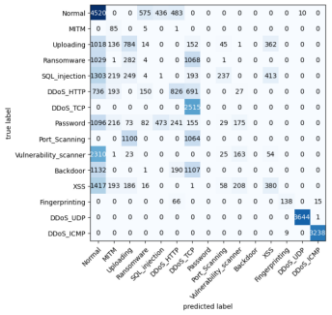
Table 6 provide the assessment of the performance of various ML models on the Edge-IIoTset dataset within a multi-class classification context. The evaluation uses metrics such as accuracy, precision, recall, F1-score, and the Micro-Average AUC. The results show a wide range of effectiveness among the models. Some models exhibit moderate performance, providing consistent but limited classification capabilities. Conversely, other models demonstrate exceptional performance, achieving perfect scores across all metrics, highlighting their robustness and reliability in distinguishing between multiple classes. Some models show a balanced trade-off between performance metrics, excelling in certain areas while maintaining consistent overall performance. This evaluation underscores the varying strengths of different algorithms in handling complex multi-class classification tasks.

Table 6. Numerical results of ML algorithms on Edge-IIoTset dataset (Multi-Class Classification scenario).

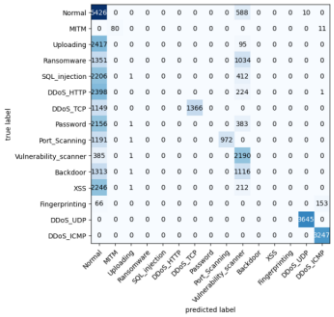
ML algorithm	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Micro-Average AUC**
Logistic Regression	0.43	0.35	0.43	0.36	0.80
Decision Tree Classifier	0.44	0.36	0.44	0.36	0.88
Random Forest Classifier	1.00	1.00	1.00	1.00	1.00
K-nearest neighbors (KNN)	0.93	0.94	0.93	0.93	1.00
SVM Classifier	0.40	0.41	0.40	0.36	0.88
Gradient Boosting Classifier	1.00	1.00	1.00	1.00	1.00
Extreme Gradient Boosting (XGBoost) Classifier	1.00	1.00	1.00	1.00	1.00
Light Gradient Boosting Machine (LGBM) Classifier	1.00	1.00	1.00	1.00	1.00
CatBoost Classifier	1.00	1.00	1.00	1.00	1.00
Naive Bayes Classifier	0.43	0.56	0.43	0.42	0.87
Linear Discriminant Analysis (LDA)	0.41	0.50	0.41	0.43	0.90
Quadratic Discriminant Analysis (QDA)	0.60	0.79	0.60	0.61	0.93
Passive Aggressive Classifier	0.49	0.38	0.49	0.39	0.85
AdaBoost Classifier	0.99	0.99	0.99	0.99	1.00
Ridge Classifier	0.73	0.74	0.73	0.73	0.98

Figure 5. Confusion Matrices of ML algorithms on NSL-KDD dataset (Multi-Class Classification scenario)

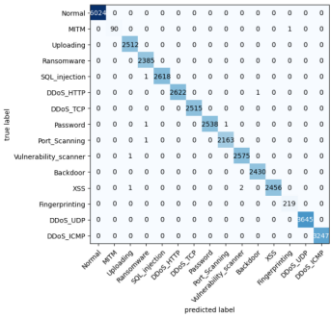
Figure 5 illustrates an exhaustive analysis of confusion matrices resulting from the comprehensive evaluation of diverse ML algorithms applied to the NSL-KDD dataset within a multi-class classification framework. Tasked with discerning between five distinct classes normal, Dos, R2L, Probe, and U2R, these matrices offer a meticulous breakdown of algorithmic performance



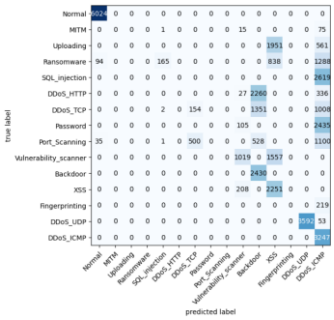
Logistic Regression



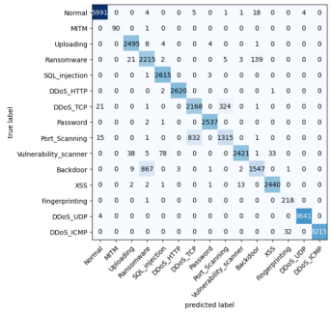
Decision Tree Classifier



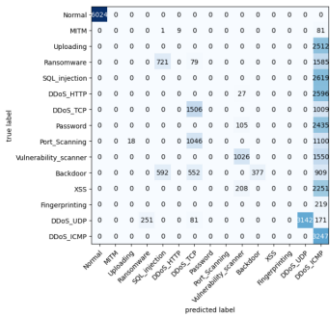
Random Forest Classifier



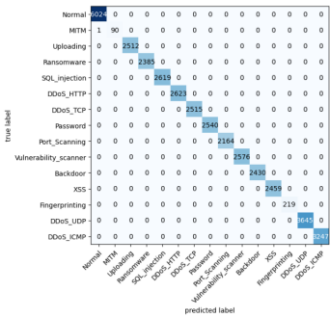
Passive Aggressive Classifier



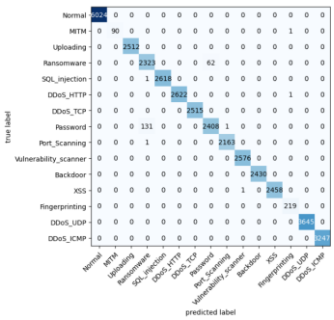
K-nearest neighbors (KNN)



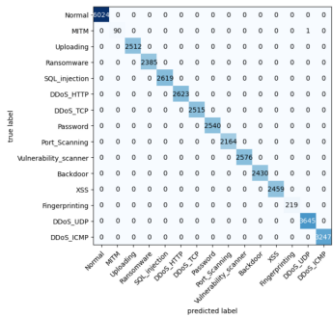
SVM Classifier



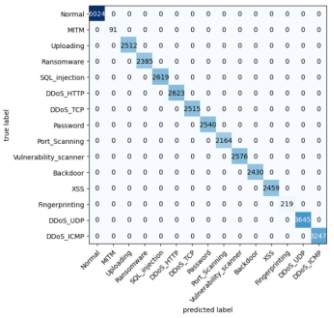
Gradient Boosting Classifier



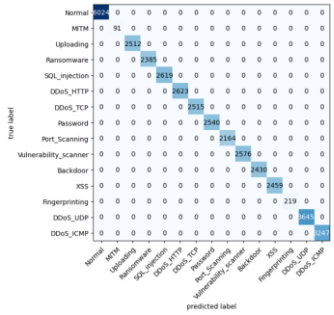
AdaBoost Classifier



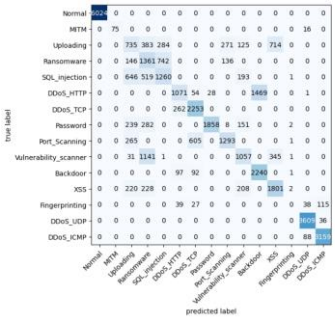
Extreme Gradient Boosting (XGBoost) Classifier



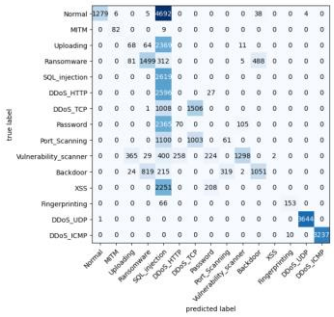
Light Gradient Boosting Machine (LGBM) Classifier



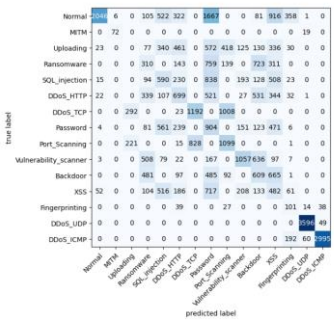
CatBoost Classifier



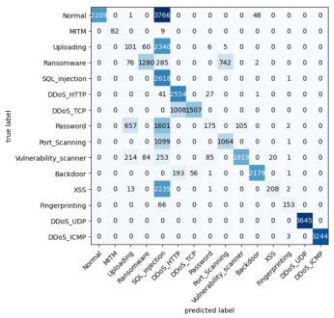
Ridge Classifier



Naive Bayes Classifier



Linear Discriminant Analysis (LDA)



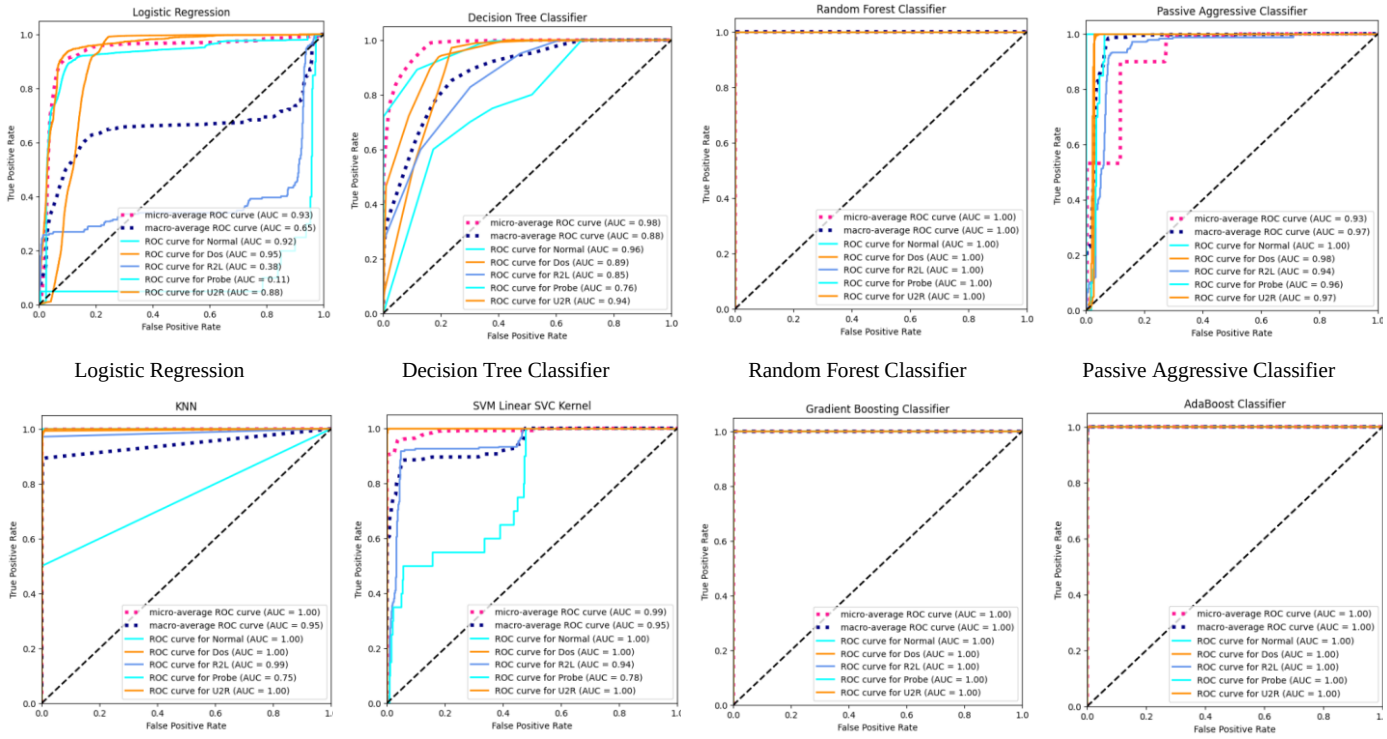
Quadratic Discriminant Analysis (QDA)

Figure 6. Confusion Matrices of ML algorithms on Edge-IIoTset dataset (Multi-Class Classification scenario)

metrics. This comparative assessment facilitates the identification of algorithmic strengths and vulnerabilities, thereby informing targeted optimization strategies.

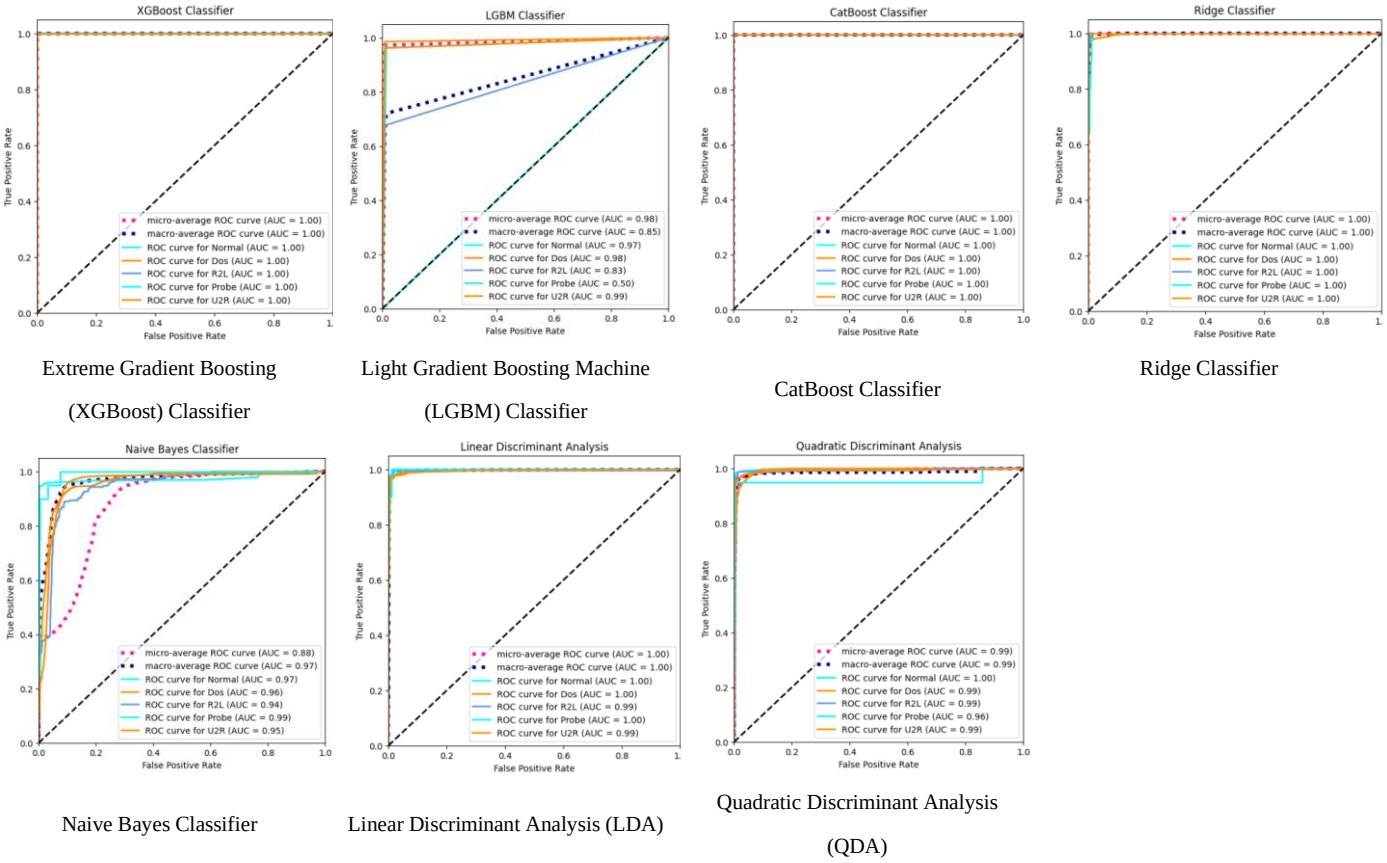
Figure 6 illustrates an examination of confusion matrices resulting from the meticulous evaluation of diverse ML algorithms applied to the Edge-IIoTset dataset in a multi-class classification setting. The dataset presents a formidable challenge of discerning among 14 distinct classes encompassing a spectrum of cyber threat categories including Ransomware, DDoS_HTTP, SQL_injection, MITM, Port_Scanning, XSS, Backdoor, Uploading, Vulnerability_scanner, DDoS_UDP, DDoS_ICMP, Password, DDoS_TCP, Normal, and Fingerprinting. Through these matrices, a detailed dissection of algorithmic performance metrics is provided, elucidating the nuances of true positives, false positives, true negatives, and false negatives. This granular analysis serves as a cornerstone for assessing the efficacy of algorithms in accurately classifying diverse threat scenarios. This comparative exploration lays the groundwork for targeted refinement strategies, pivotal for advancing the frontier of industrial IoT security through enhanced threat detection and mitigation methodologies.

Figure 7 showcases a series of visualizations demonstrating (ROC) curves, which result from the rigorous evaluation of diverse ML algorithms on the NSL-KDD dataset within a multi-class classification scenario. These graphical representations elucidate the prowess of each algorithm in discerning between normal network activities and potentially intrusive behaviors. Employing the One-vs-the-Rest (OvR) multiclass strategy, also known as one-vs-all, these curves delineate the performance of each algorithm as it treats individual classes as positives while the rest are deemed negatives collectively. Micro-averaging is subsequently applied to amalgamate the contributions from all classes, leveraging property for calculating average metrics [36]. The crux of algorithmic efficacy lies in the AUC-ROC, where higher values signify superior performance in accurately identifying instances of intrusion while mitigating false positives [35]. This graphical analysis offers a comprehensive insight into the algorithmic landscape of network intrusion detection, facilitating informed decision-making in cybersecurity endeavors.



Logistic Regression | Decision Tree Classifier | Random Forest Classifier | Passive Aggressive Classifier

K-nearest neighbors (KNN) | SVM Classifier | Gradient Boosting Classifier | AdaBoost Classifier



Extreme Gradient Boosting (XGBoost) Classifier

Light Gradient Boosting Machine (LGBM) Classifier

CatBoost Classifier

Ridge Classifier

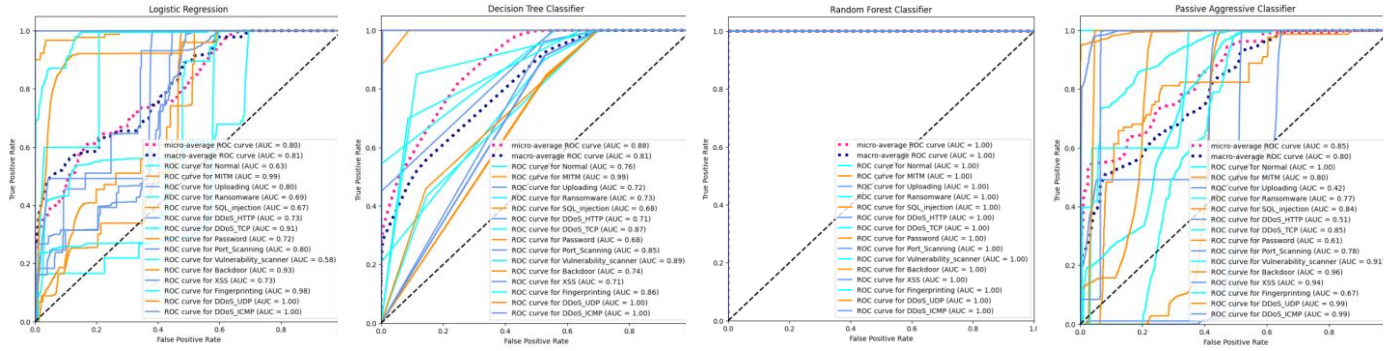
Naive Bayes Classifier

Linear Discriminant Analysis (LDA)

Quadratic Discriminant Analysis (QDA)

Figure 7. visualizations of ROC curves of ML algorithms on NSL-KDD dataset (Multi-Class Classification scenario)

Figure 8 unveils a panoramic display of (ROC) curves resulting from the examination of ML algorithms applied to the Edge-IIoTset dataset in a multi-class classification context. Utilizing (OvR) multiclass approach, each algorithm meticulously constructs ROC curves for individual classes, methodically distinguishing between positive and collective negative classes at each iteration. Micro-averaging synthesizes a spectrum of performance metrics, harmonizing contributions across all classes [36]. The essence of algorithmic prowess is encapsulated within the ROC curve, where elevated AUC-ROC values illuminate the path toward adeptly identifying instances of intrusion while skillfully navigating the landscape of false positives [35].

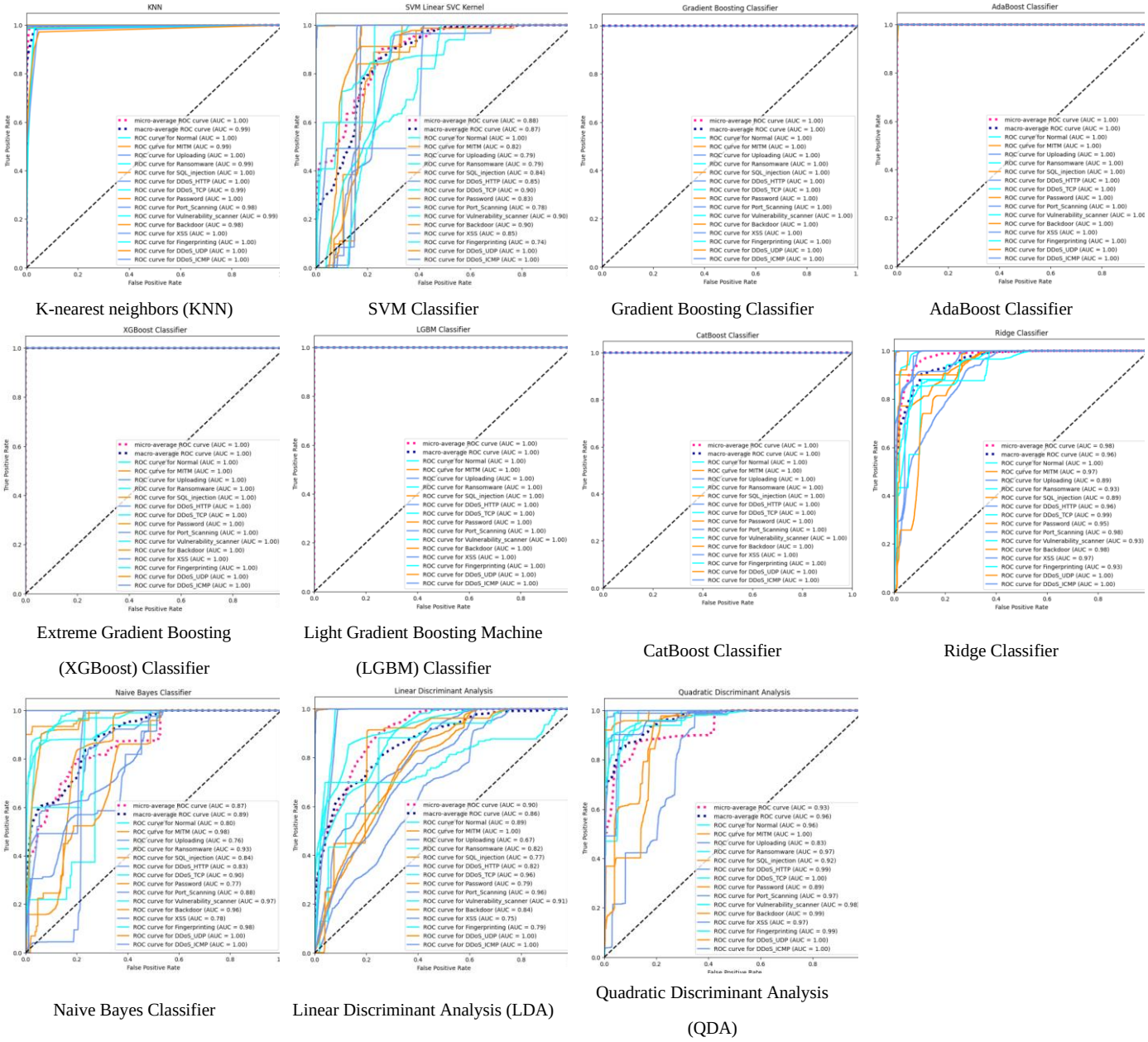


Logistic Regression

Decision Tree Classifier

Random Forest Classifier

Passive Aggressive Classifier



K-nearest neighbors (KNN) | SVM Classifier | Gradient Boosting Classifier | AdaBoost Classifier

Extreme Gradient Boosting (XGBoost) Classifier | Light Gradient Boosting Machine (LGBM) Classifier | CatBoost Classifier | Ridge Classifier

Naive Bayes Classifier | Linear Discriminant Analysis (LDA) | Quadratic Discriminant Analysis (QDA)

Figure 8. ROC curves of ML algorithms on Edge-IIoTset dataset (Multi-Class Classification scenario)

6. Conclusions and Future Directions

In this study, we embarked on a journey to investigate the effectiveness of ML methodologies in the context of intrusion detection, emphasizing the importance of reproducible baselines in benchmarking algorithmic performance. Through rigorous experimentation and analysis, we presented comprehensive results comparing the efficacy of different ML algorithms across two separate datasets: NSL-KDD and Edge-IIoTset. Our findings underscored the significance of nuanced evaluation metrics, ranging from accuracy and precision to ROC curves and confusion matrices, in gauging algorithmic robustness and adaptability.

In our plan for future work, we will explore the promise of advanced ML (algorithms such as graph neural networks [35], Deep Reinforcement Learning [37], Quantum Machine learning [38], etc.) for revolutionizing the role of the design of intrusion detection systems in dynamic and complex IoT settings.

Supplementary Materials

https://github.com/Salma-00/Machine-Learning-for-Intrusion-Detection

Author Contributions

All authors contributed equally to this work.

Funding

This research was conducted without external funding support.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Conflicts of Interest

The authors declare that there is no conflict of interest in the research.

Informed Consent Statement

Not applicable.

References

[1] Pinto, A., Herrera, L.-C., Donoso, Y., & Gutierrez, J. A. (2023). Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure. Sensors, 23(5), 2415. https://doi.org/10.3390/s23052415.

[2] Latif, S., Dola, F. F., Afsar, M. M., & Esha, I. J. (2022). Investigation of Machine Learning Algorithms for Network Intrusion Detection. I.J. Information Engineering and Electronic Business, 2(1), 1-22. Retrieved from http://www.mecs-press.org/ DOI: 10.5815/ijieeb.2022.02.01.

[3] Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., & Yang, A. (2022). Comparative research on network intrusion detection methods based on machine learning. Computers & Security, 121, 102861. https://doi.org/10.1016/j.cose.2022.102861.

[4] S. Shitharth, P. R. Kshirsagar, P. K. Balachandran, K. H. Alyoubi and A. O. Khadidos, "An Innovative Perceptual Pigeon Galvanized Optimization (PPGO) Based Likelihood Naïve Bayes (LNB) Classification Approach for Network Intrusion Detection System," in IEEE Access, vol. 10, pp. 46424-46441, 2022, doi: 10.1109/ACCESS.2022.3171660.

[5] Le, T.-T.-H., Kim, H., Kang, H., & Kim, H. (2022). Classification and explanation for intrusion detection system based on ensemble trees and SHAP method. Sensors, 22(3), 1154. https://doi.org/10.3390/s22031154.

[6] Musa, A. B. (2013). Comparative study on classification performance between support vector machine and logistic regression. International Journal of Machine Learning & Cybernetics, 4(1), 13–24. DOI: 10.1007/s13042-012-0068-x.

[7] Khanna, D., Sahu, R., Baths, V., & Deshpande, B. (2015). Comparative Study of Classification Techniques (SVM, Logistic Regression and Neural Networks) to Predict the Prevalence of Heart Disease. International Journal of Machine Learning and Computing, 5(5), October 2015.

[8] Shler Farhad Khorshid, & Adnan Mohsin Abdulazeez. (2021). BREAST CANCER DIAGNOSIS BASED ON K-NEAREST NEIGHBORS: A REVIEW. PalArch's Journal of Archaeology of Egypt / Egyptology, 18(4), 1927-1951. Retrieved from https://archives.palarch.nl/index.php/jae/article/view/6601

[9] Kramer, O. (2013). K-Nearest Neighbors. In: Dimensionality Reduction with Unsupervised Nearest Neighbors. Intelligent Systems Reference Library, vol 51. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-38652-7_2

[10] B. Charbuty and A. Abdulazeez, "Classification Based on Decision Tree Algorithm for Machine Learning", JASTT, vol. 2, no. 01, pp. 20 - 28, Mar. 2021.

[11] Kulkarni, V. Y., & Sinha, P. K. (2013). Random Forest Classifiers: A Survey and Future Research Directions. International Journal of Advanced Computing, 36(Issue), 27702358.

[12] Parmar, A., Katariya, R., Patel, V. (2019). A Review on Random Forest: An Ensemble Classifier. In: Hemanth, J., Fernando, X., Lafata, P., Baig, Z. (eds) International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018. ICICI 2018. Lecture Notes on Data Engineering and Communications Technologies, vol 26. Springer, Cham. https://doi.org/10.1007/978-3-030-03146-6_86

[13] Bentéjac, C., Csörgő, A. & Martínez-Muñoz, G. A comparative analysis of gradient boosting algorithms. Artif Intell Rev 54, 1937–1967 (2021). https://doi.org/10.1007/s10462-020-09896-5

[14] D. Upadhyay, J. Manero, M. Zaman and S. Sampalli, "Gradient Boosting Feature Selection With Machine Learning Classifiers for Intrusion Detection on Power Grids," in IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 1104-1116, March 2021, doi: 10.1109/TNSM.2020.3032618

[15] Arif Ali, Z., H. Abduljabbar, Z., A. Taher, H., Bibo Sallow, A., & Almufti, S. M. (2023). Exploring the Power of eXtreme Gradient Boosting Algorithm in Machine Learning: a Review. Academic Journal of Nawroz University, 12(2), 320–334. https://doi.org/10.25007/ajnu.v12n2a1612

[16] M. Osman, J. He, F. M. M. Mokbal, N. Zhu and S. Qureshi, "ML-LGBM: A Machine Learning Model Based on Light Gradient Boosting Machine for the Detection of Version Number Attacks in RPL-Based Networks," in IEEE Access, vol. 9, pp. 83654-83665, 2021, doi: 10.1109/ACCESS.2021.3087175.
[17] Ibrahim, A.A., Ridwan, R.L., Muhammed, M.M., Abdulaziz, R.O., & Saheed, G.A. (2020). Comparison of the CatBoost Classifier with other Machine Learning Methods. International Journal of Advanced Computer Science and Applications, 11.
[18] M. Sheykhmousa, M. Mahdianpari, H. Ghanbari, F. Mohammadimanesh, P. Ghamisi and S. Homayouni, "Support Vector Machine Versus Random Forest for Remote Sensing Image Classification: A Meta-Analysis and Systematic Review," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 13, pp. 6308-6325, 2020, doi: 10.1109/JSTARS.2020.3026724.
[19] Bafjaish, S. S. (2020). Comparative Analysis of Naive Bayesian Techniques in Health-Related For Classification Task. Journal of Soft Computing and Data Mining, 1(2), 1-10. https://penerbit.uthm.edu.my/ojs/index.php/jscdm/article/view/7144
[20] Maswadi, K., Ghani, N.A., Hamid, S. et al. Human activity classification using Decision Tree and Naïve Bayes classifiers. Multimed Tools Appl 80, 21709–21726 (2021). https://doi.org/10.1007/s11042-020-10447-x.
[21] Isnanto, R. R., Rashad, I., & Widodo, C. E. (2023). Classification of Heart Disease Using Linear Discriminant Analysis Algorithm. 3S Web of Conferences ICENIS 2023, 448, 020. https://doi.org/10.1051/e3sconf/202344802053
[22] Ha, D.H., Nguyen, P.T., Costache, R. et al. Quadratic Discriminant Analysis Based Ensemble Machine Learning Models for Groundwater Potential Modeling and Mapping. Water Resour Manage 35, 4415–4433 (2021). https://doi.org/10.1007/s11269-021-02957-6
[23] Araveeporn, A. (2022). Comparing the Linear and Quadratic Discriminant Analysis of Diabetes Disease Classification Based on Data Multicollinearity. *Indawi International Journal of Mathematics and Mathematical Sciences*, Volume 2022, Article ID 7829795, 11 pages. https://doi.org/10.1155/2022/7829795
[24] Gupta, S., Meel, P. (2021). Fake News Detection Using Passive-Aggressive Classifier. In: Ranganathan, G., Chen, J., Rocha, Á. (eds) Inventive Communication and Computational Technologies. Lecture Notes in Networks and Systems, vol 145. Springer, Singapore. https://doi.org/10.1007/978-981-15-7345-3_13
[25] Kiranmayee, B.V., Suresh, C., SreeRakshak, S. (2022). Classification of the Suicide-Related Text Data Using Passive Aggressive Classifier. In: Shakya, S., Balas, V.E., Kamolphiwong, S., Du, KL. (eds) Sentimental Analysis and Deep Learning. Advances in Intelligent Systems and Computing, vol 1408. Springer, Singapore. https://doi.org/10.1007/978-981-16-5157-1_34
[26] Ruihu Wang, AdaBoost for Feature Selection, Classification and Its Relation with SVM, A Review, Physics Procedia, Volume 25,2012, Pages 800-807, ISSN 1875-3892, https://doi.org/10.1016/j.phpro.2012.03.160.
[27] Chen, Y., Dou, P., Yang, X. Improving Land Use/Cover Classification with a Multiple Classifier System Using AdaBoost Integration Technique. Remote Sens. 2017, 9, 1055. https://doi.org/10.3390/rs9101055
[28] Deepa, N., Prabadevi, B., Maddikunta, P.K. et al. An AI-based intelligent system for healthcare analysis using Ridge-Adaline Stochastic Gradient Descent Classifier. J Supercomput 77, 1998–2017 (2021). https://doi.org/10.1007/s11227-020-03347-2
[29] Ambika B J, Nirmala S Guptha, & Syeda Ayesha Siddiqha. (2023). Anaemia Estimation for Patients Using Lasso And Ridge Regression Algorithms. Milestone Transactions on Medical Technometrics, 1(2), 53–63. https://doi.org/10.5281/zenodo.10255349
[30] Magdy, M. E., Matter, A. M., Hussin, S., Hassan, D., & Elsaid, S. A. (2023). A Comparative Study of Intrusion Detection Systems Applied to NSL-KDD Dataset. The Egyptian International Journal of Engineering Sciences and Technology, 43, 88–98. DOI: 10.21608/EIJEST.2022.137441.1156.
[31] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras and H. Janicke, "Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning," in IEEE Access, vol. 10, pp. 40281-40306, 2022, doi: 10.1109/ACCESS.2022.3165809.
[32] M. S. Alshehri, O. Saidani, F. S. Alrayes, S. F. Abbasi and J. Ahmad, "A Self-Attention-Based Deep Convolutional Neural Networks for IIoT Networks Intrusion Detection," in IEEE Access, vol. 12, pp. 45762-45772, 2024, doi: 10.1109/ACCESS.2024.3380816.
[33] Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for Multi-Class Classification: An Overview. CRIF S.p.A. arXiv:2008.05756v1 [stat.ML].
[34] Brefeld, U., & Scheffer, T. (2005). AUC Maximizing Support Vector Learning. Humboldt-Universität zu Berlin, Department of Computer Science, Unter den Linden 6, 10099 Berlin, Germany.
[35] Abdel-Basset, M., Moustafa, N., Hawash, H., & Tari, Z. (2023). Responsible Graph Neural Networks (1st ed.). New York: Chapman and Hall/CRC. https://doi.org/10.1201/9781003329701
[36] scikit-learn. (n.d.). Plotting Receiver Operating Characteristic (ROC) Curves. Retrieved from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
[37] Abdel-Basset, M., Moustafa, N., Hawash, H., Ding, W., Abdel-Basset, M., Moustafa, N., Hawash, H. and Ding, W., 2022. Deep reinforcement learning for secure Internet of Things. Deep learning techniques for IoT security and privacy, pp.203-213. https://doi.org/10.1007/978-3-030-89025-4_8
[38] Huang, Hsin-Yuan, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R. McClean. "Power of data in quantum machine learning." *Nature communications* 12, no. 1 (2021): 2631. https://doi.org/10.1038/s41467-021-22539-9