applied sciences

MDPI

Review

Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity

Pierpaolo Dini [1,*], Abdussalam Elhanashi [1], Andrea Begni [1], Sergio Saponara [1], Qinghe Zheng [2] and Kaouther Gasmi [3]

[1] Department of Information Engineering, University of Pisa, 56126 Pisa, Italy; a.elhanashi@studenti.unipi.it (A.E.); a.begni88@gmail.com (A.B.); sergio.saponara@unipi.it (S.S.)
[2] School of Intelligence Engineering, Shandong Management University, Jinan 250100, China; 15005414319@163.com
[3] Department of the Computer Science, University of Tunis, Tunis 1007, Tunisia; kaouther.gasmi@enit.rnu.tn
* Correspondence: pierpaolo.dini@ing.unipi.it

Abstract: The Intrusion Detection System (IDS) is an effective tool utilized in cybersecurity systems to detect and identify intrusion attacks. With the increasing volume of data generation, the possibility of various forms of intrusion attacks also increases. Feature selection is crucial and often necessary to enhance performance. The structure of the dataset can impact the efficiency of the machine learning model. Furthermore, data imbalance can pose a problem, but sampling approaches can help mitigate it. This research aims to explore machine learning (ML) approaches for IDS, specifically focusing on datasets, machine algorithms, and metrics. Three datasets were utilized in this study: KDD 99, UNSW-NB15, and CSE-CIC-IDS 2018. Various machine learning algorithms were chosen and examined to assess IDS performance. The primary objective was to provide a taxonomy for interconnected intrusion detection systems and supervised machine learning algorithms. The selection of datasets is crucial to ensure the suitability of the model construction for IDS usage. The evaluation was conducted for both binary and multi-class classification to ensure the consistency of the selected ML algorithms for the given dataset. The experimental results demonstrated accuracy rates of 100% for binary classification and 99.4In conclusion, it can be stated that supervised machine learning algorithms exhibit high and promising classification performance based on the study of three popular datasets.

Keywords: intrusion detection systems; machine learning; feature selection; data management; KDD 99; UNSW-NB15; CSE-CIC-IDS 2018

Citation: Dini, P.; Elhanashi, A.; Begni, A.; Saponara, S.; Zheng, Q.; Gasmi, K. Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity. *Appl. Sci.* **2023**, *13*, 7507. https://doi.org/10.3390/app13137507

Academic Editor: Luis Javier Garcia Villalba

Received: 30 May 2023
Revised: 20 June 2023
Accepted: 21 June 2023
Published: 25 June 2023

1. Introduction

1.1. Overview

With the rapid growth of networking technologies and the increasing number of cyber threats, ensuring effective cybersecurity has become a paramount concern. One crucial aspect of cybersecurity is the detection and prevention of unauthorized access and malicious activities within computer networks. Intrusion Detection Systems (IDS) play a vital role in monitoring network traffic and identifying potential security breaches. Traditional IDS methods rely heavily on signature-based approaches, which are limited in their ability to detect novel and sophisticated attacks. To overcome these limitations, researchers and practitioners have started to explore the integration of machine learning techniques into IDS design. Machine learning (ML) has demonstrated remarkable success in various domains, including natural language processing, computer vision, and pattern recognition. Leveraging ML algorithms in the realm of networking cybersecurity offers promising opportunities to enhance the accuracy and efficiency of intrusion detection systems. ML-based IDS models can learn from large volumes of network data, detect anomalous patterns, and adapt to evolving attack strategies [1–3]. This approach holds the potential to improve the overall security posture by reducing false positives and detecting

previously unknown attacks. The design of IDS exploiting machine learning for networking cybersecurity involves several key components. Firstly, a robust and comprehensive dataset is required for training and evaluating the ML models. The proposed datasets encompass a wide range of network traffic patterns, including both normal and malicious activities, to enable effective learning. Secondly, suitable feature selection and extraction techniques are crucial to capture relevant information from the network data [4]. These features serve as input to the ML algorithms, enabling them to discern normal traffic from potential intrusions. Furthermore, the choice of ML algorithms plays a vital role in IDS design. Various algorithms, such as support vector machines, random forests, and deep learning architectures, have been investigated for intrusion detection. Each algorithm has its strengths and weaknesses, and the selection depends on factors such as the complexity of the problem, the availability of data, and the desired trade-offs between detection accuracy and computational efficiency. In this paper, we explore the design and implementation of intrusion detection systems that exploit machine learning techniques for networking cybersecurity. We examine various machine learning algorithms and methodologies that have been applied to IDS for both binary and multi-classification approaches. We analyze their strengths and weaknesses and discuss their suitability for different types of network environments and attack scenarios.

1.2. Motivations

Intrusion detection systems (IDS) play a critical role in safeguarding computer networks by identifying and responding to security threats. The use of machine learning models has gained popularity in IDS due to their ability to handle large volumes of data and detect patterns in real-time. By leveraging machine learning models, IDS can learn from historical data and detect new patterns that may indicate potential intrusions. This helps to reduce false positives and enhance the accuracy of IDS. The application of machine learning models in IDS is particularly important when dealing with diverse datasets. Different datasets may exhibit distinct characteristics, including various types of intrusions, network configurations, and user behavior. Machine learning models can be trained on different datasets and adapt to the unique characteristics of each dataset. This ensures that IDS remains effective in detecting intrusions across a variety of environments. Networked computer systems and the continuous availability of Internet services are crucial for modern society, which heavily relies on them for almost every activity. However, with the increasing use of Internet-based technologies, attackers can target computer systems without physically interfering with them, leading to malfunctions and compromising security in terms of confidentiality, availability, and integrity. Network traffic comprises packets characterized by properties such as duration, protocol type, and the amount of data transferred between source and destination. Since attackers can compromise packets by modifying their content during creation or transit, it is vital to identify attacks and ensure service integrity, considering the impossibility of creating a completely attack-free networked computer system. The number of anomalies, such as misconfigurations of network devices, port scans as preparations for future attacks, resource-consuming and self-spreading viruses and worms, or denial of service (DoS) attacks that render network services unavailable, increases proportionally with the growth of network traffic. Effective detection and diagnosis of such anomalies are crucial to guarantee proper and reliable functioning [5]. In this context, network security and reliability become even more crucial in safety-critical systems (SCS). An SCS refers to a system whose failure or malfunction can result in equipment/property loss or severe damage, environmental harm, or serious injury or death to people. SCS encompasses various applications such as robots for industrial automation, logistics and human assistance, vehicles, medical systems, and defense. The continuous evolution towards software-defined autonomous and connected systems further escalates the risk of cyber attacks and their consequences. Therefore, the availability of intrusion and anomaly detection capabilities is of utmost importance for SCS. Certain literature works employ the IDS concept to manage communication flow anomalies in

the context of control systems in mechatronic and industrial applications [6–12]. Unlike attack monitoring algorithms in communication network contexts, which tend to associate a dynamic model with overall behavior in a complex manner, a hybrid approach combining model-based and data-driven techniques is employed. This involves utilizing state observers and estimates through stochastic analysis of the residuals between the model and direct process measurements. While this work often proposes specific anomaly detection solutions with known integration into the overall system, the IDS problem in the context of this article, focusing on the security of communication networks, is more complex. The attack is not treated merely as an additive nuisance that perturbs the model relative to nominal behavior. Hence, this article places significant emphasis on the security aspects of communication networks, considering three of the most widely used datasets in these applications. There have been significant advancements in the design of machine learning-based Intrusion Detection Systems (IDS) for network cybersecurity. However, there are still several research gaps and limitations that need to be addressed. One major research gap is the lack of labeled datasets for training and evaluating IDS models. Existing publicly available datasets suffer from issues like insufficient size, lack of diversity in attack scenarios, or outdated data, which makes it difficult to develop robust and generalized IDS models capable of detecting novel and sophisticated attacks. Another limitation is the lack of transparency and interpretability of machine learning-based IDS models, particularly deep learning algorithms, which operate as black boxes, making it challenging to understand the decision-making process. This lack of transparency hampers trust and adoption, especially in critical network security applications where explanations for detected threats are crucial. Additionally, existing work often focuses on a binary classification, distinguishing between normal and malicious network traffic, without exploring more detailed classification schemes like identifying specific attack types or characterizing the severity of an intrusion.

1.3. Machine Learning in Network Security

Within Control Systems (CS) networks, a wide array of components engage in communication with one another. An assailant possessing proficient knowledge of networks, operating systems, and software can exploit these components to gain unauthorized access and carry out malicious activities within the control system. Among the numerous types of attacks that a network may encounter, three notable threats deserve attention: Denial of Service (DOS), Spoofing, and Eavesdropping. During a DOS attack, the intruder inundates the network with a barrage of either valid or invalid messages, thereby impairing the availability of network resources. Spoofing, on the other hand, involves the attacker assuming the identity of a legitimate user to gain unauthorized access to privileged data and network resources. Eavesdropping refers to attacks on the confidentiality of data transmitted across the network. In wireless networks, eavesdropping by third parties poses a significant threat since attackers can intercept transmissions over the air from a considerable distance, beyond the premises of the company. The ongoing cat-and-mouse game between attackers and Intrusion Detection Systems (IDS) has spurred significant advancements in security measures. However, it has also given rise to increasingly subtle and difficult-to-identify attack methods. Here are some key points of benefit introduce by ML in the context of networking cybersecurity:

- **Threat detection**: Machine learning can be used to develop predictive models capable of identifying and detecting suspicious behavior or cyber attacks in communication networks. These models can analyze large amounts of data in real time from various sources, such as network logs, packet traffic and user behavior, in order to identify anomalies and patterns associated with malicious activity.
- **Automation of attack responses**: Automation is a key aspect in the security of communication networks. Using machine learning algorithms can help automate an attack response, allowing you to react quickly and effectively to threats. For example, a machine learning system can be trained to recognize certain types of attacks and

automatically trigger appropriate countermeasures, such as isolating compromised devices or changing security rules.

- **Detect new types of attacks**: As cyberthreats evolve, new types of attacks are constantly emerging. The traditional signature-based approach may not be enough to detect these new threats. The use of machine learning algorithms can help recognize anomalous patterns or behavior that could indicate the emergence of new types of attacks, even in the absence of specific signatures.
- **Reduce False Positives**: Traditional security systems often generate large numbers of false positives, that is, they falsely report normal activity as an attack. This can lead to wasted time and valuable resources in dealing with non-relevant reports. Using machine learning models can help reduce false positives, increasing the efficiency of security operations and enabling more accurate identification of real threats.
- **Adaptation and continuous learning**: Machine learning models can be adapted and updated in real time to address new threats and changing conditions of communication networks. With continuous learning, models can improve over time, gaining greater knowledge of threats and their variants.

Ultimately, using machine learning in communication network security offers a number of benefits, including the ability to spot threats in real time, automate responses, detect new types of attacks, and reduce false positives. These benefits help improve overall network security and protect underlying data and assets.

As schematically shown in Figure 1, the machine learning paradigm is composed of the following main steps:

- **Data Collection**: The initial phase involves the collection of training data. This data consists of labeled examples, i.e., pairs of matching inputs and outputs. For example, if we are trying to build a model to recognize images of cats, the data will contain images of cats labeled "cat" and different images labeled "not cat".
- **Data preparation**: This phase involves cleaning, normalizing, and transforming the training data to make it suitable for processing by the machine learning model. This can include eliminating missing data, handling categorical characteristics, and normalizing numeric values.
- **Model selection and training**: In this phase, you select the appropriate machine learning model for the problem at hand. The model is then trained on the training data, which consists of making the model learn the patterns and relationships present in the data. During training, the model is iteratively adjusted to minimize the error between its predictions and the corresponding output labels in the training data.
- **Model Evaluation**: After training, the model is evaluated using separate test data, which was not used during the training. This allows you to evaluate the effectiveness of the model in generalizing patterns to new data. Several metrics, such as accuracy, precision, and area under the *ROC* curve, are used to evaluate model performance.
- **Model Usage**: Once the model has been trained and evaluated, it can be used to make predictions on new input data. The model applies the relationships learned during training to make predictions about new input instances.

In the following sections the main types of ML models and the preliminary data manipulation techniques for the optimal training of the selected algorithm will be described in great detail, as well as a presentation of the results of applying ML algorithms on the most important datasets for validation of ML-based systems in the IDS context.

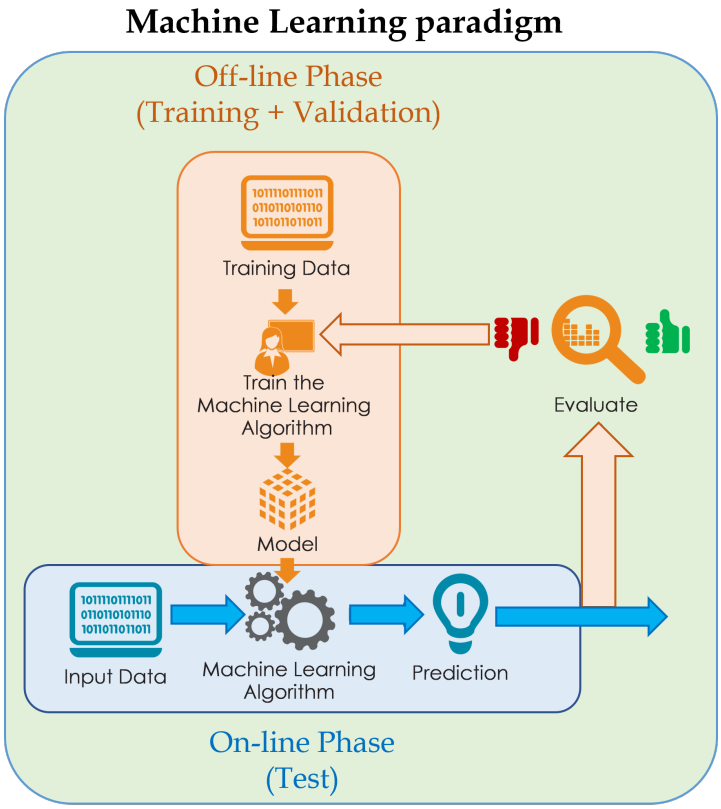

Figure 1. Schematic representation of the machine learning workflow.

1.4. Contribution

This paper delves into the exploration of different machine learning models, with the objective of evaluating their performance in addressing issues related to data traffic security. While existing state-of-the-art studies tend to focus on a limited selection of machine learning models, offering minimal comprehensive comparisons in terms of the effectiveness of available solutions, this research stands out by conducting extensive testing of numerous machine learning models. This approach allows for a critical evaluation of the strengths and weaknesses inherent in each model. Many existing papers in the literature propose models tailored exclusively for either binary classification or multi-class problems. However, this article takes a broader approach by testing the selected models for both binary and multi-class classification scenarios, significantly enriching the depth of the technical analysis presented. The contributions made by this study can be summarized as follows:

- A thorough analysis of multiple machine learning algorithms has been conducted to effectively handle the processing of large volumes of network traffic data. These algorithms can be scaled to accommodate the expanding sizes of networks, resulting in the development of a robust and efficient intrusion detection system.
- The impact of dataset selection on the performance of intrusion detection systems has been explored in this study. By comparing the performance of intrusion detection systems across three distinct datasets (KDD 99, UNSW-NB15, and CSE-CIC-IDS 2018), a better understanding of the influence of dataset choice on system performance has been achieved.
- This research has made significant contributions to the advancement of effective feature engineering techniques, which are essential for constructing successful machine learning models. These techniques encompass feature selection, feature scaling, and feature normalization, enhancing the overall effectiveness of the models.

- Additionally, this study proposes an analysis of the computational time required by different models, a factor often overlooked in existing literature. This inclusion enriches the performance analysis of the models across the selected datasets.

2. Related Works

Machine learning (ML) is a field of study within artificial intelligence that focuses on computer algorithms capable of improving automatically through experience and data utilization. ML algorithms construct models by analyzing sample data, referred to as "training data", to make predictions or decisions without explicit programming. ML tasks can be categorized into four paradigms: Supervised Learning, Unsupervised Learning, Semi-supervised Learning, and Reinforcement Learning. For the purpose of this paper, we will solely analyze the category of Supervised Learning. Supervised Learning involves the learning of a function that can map inputs to outputs based on a set of input-output examples. Each example consists of an input object, typically represented as a vector, and its corresponding desired output value. By examining the training data, algorithms generate an inferred function that can be employed to map new examples. The primary objective of the algorithm is to accurately determine class labels for unseen observations, making the ability to generalize from the training data crucial. Among the various supervised ML algorithms discussed in the literature, we have selected the most widely used approaches in the field of classification. Our review is very detailed in analyzing the various ML models, to which in fact a detailed subsection is dedicated in which in addition to an overview, as is done in most review papers the state of the art is presented and compared the results obtained by applying each model to 3 different datasets. Moreover, note that each of the three represents a kind of standard for testing ML/AI classifiers for verifying the performance of IDS systems. It should also be noted that each model and the results obtained are the result of new tests carried out to obtain an analysis of the models with the same calculation system used. This type of analysis is substantially absent in the reviews of the state of the art found in the literature in which only the results deriving from scientific research articles are taken and compared with those obtained by other authors. In our opinion, this is a serious flaw in the overview articles of ML methodologies as they are strongly dependent on the SW modules, and on the type of HW used, which must be the same to make an exhaustive comparison. The rest of the section is divided into subsections dedicated to the ML models taken into consideration in which not only the technical aspects are analyzed but also the pros and cons in the context of the design of IDS systems.

2.1. Support Vector Machine

Support Vector Machine (SVM) is a widely utilized supervised learning approach for regression problems [13]. It is commonly employed in machine learning for regression and classification tasks. The primary objective of SVM is to determine the optimal line and decision boundary that effectively separates classes in an n-dimensional space, enabling accurate categorization of new data points [14]. SVM constructs a hyperplane or a set of hyperplanes in a high or infinite-dimensional space, which can be applied to classification, regression, and outlier detection tasks [14]. The quality of separation achieved by SVM relies on the hyperplane that maximizes the distance to the nearest training data point from any class, also known as the functional margin. A larger margin leads to a lower generalization error, thereby improving the classifier's performance [14]. In the field of intrusion detection systems, SVM is widely employed. Studies have tested SVM on the KDD 99 dataset by varying the number of features [15] or by exploring different SVM kernel types to evaluate accuracy [16]. SVM has also been tested on the UNSW-NB15 dataset, comparing its performance with other binary and multi-class classification methods [17,18]. Furthermore, SVM has been utilized with the CSE-CIC-IDS2018 dataset in work by Kanimozhi et al. [19] and Liu et al. [20]. Support Vector Machine (SVM) is a widely employed machine learning algorithm in intrusion detection systems. One of its key advantages is its ability to accurately classify complex and high-dimensional

datasets, which is crucial for detecting unknown and evolving cyber threats. SVM is capable of handling both linear and non-linear data and can efficiently process large datasets. However, one of the drawbacks of SVM is the potentially time-consuming training process, particularly with large datasets. Additionally, the performance of SVM heavily relies on selecting appropriate parameters, which can pose a challenge. Moreover, the presence of irrelevant and noisy data can negatively impact the accuracy of intrusion detection using SVM [15,20]. In summary, SVM is a popular machine learning algorithm used in intrusion detection systems. It demonstrates strong performance in accurately classifying complex datasets, but considerations should be given to training time, parameter selection, and the presence of irrelevant data to ensure effective intrusion detection.

Support Vector Machines (SVM) is a supervised learning algorithm used for both classification and regression tasks. It aims to find the optimal decision boundary, known as the hyper-plane, that separates data points belonging to different classes in a high-dimensional feature space. Given a training dataset with labeled examples, the SVM learning process involves the following steps:

Feature Mapping: If the data is not linearly separable in the input space, SVM uses a kernel function to map the input data into a higher-dimensional feature space. This allows for finding a linear decision boundary that can separate the classes in the transformed space. Margin Maximization: SVM aims to find the hyperplane that maximizes the margin between the support vectors, which are the data points closest to the decision boundary. The margin is the distance between the hyperplane and the support vectors. The larger the margin, the better the generalization performance of the SVM.

Soft Margin (C-parameter): In cases where the data is not perfectly separable, SVM allows for some misclassifications by introducing a regularization parameter called the C-parameter. This parameter controls the trade-off between maximizing the margin and minimizing the misclassification errors. A smaller C-value allows for a larger margin but permits more misclassifications, while a larger C-value reduces the margin to achieve better classification accuracy.

Optimization: The SVM learning algorithm solves a constrained optimization problem to find the optimal hyperplane. The objective is to minimize the classification error and maximize the margin. This is typically done using quadratic programming techniques. The decision function of an SVM can be represented as:

$$f(x) = sign(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b) \quad (1)$$

where $f(x)$ is the predicted class label for a new data point x, α_i and y_i are the Lagrange multipliers and class labels for the support vectors, $K(x_i, x)$ is the kernel function that measures the similarity between the support vectors and the new data point, and b is the bias term. During the learning process, the SVM determines the optimal values of α_i, y_i, and b by solving the optimization problem, typically using optimization algorithms like Sequential Minimal Optimization (SMO). Once the SVM model is trained, it can be used to make predictions on new, unseen data points by evaluating the sign of the decision function. In summary, the learning process of Support Vector Machines involves mapping the data to a higher-dimensional feature space, maximizing the margin between support vectors, optimizing the decision function parameters, and making predictions based on the learned model. Please note that the above explanation provides a general understanding of SVM learning. SVM have various extensions and variations, such as kernel selection, handling imbalanced data, and handling multi-class problems, which can be explored in more detail for a comprehensive understanding of SVM.

2.2. Decision Tree

The decision tree algorithm is a popular supervised learning approach widely utilized in data mining and statistics [21,22]. It serves as a predictive model for analyzing a set

of observations and visually representing decision-making processes. Its simplicity and integration make it one of the most popular machine learning algorithms. The decision tree algorithm utilizes a data structure called a "tree" to predict outcomes for specific problems. In the supervised approach, the algorithm is trained on a collection of pre-processed data. The main concept is to partition the data space into dense and sparse regions using the decision tree. The splitting of the binary tree can be either binary or multi-way, and the algorithm continues splitting until the data becomes sufficiently homogeneous. At the end of the training process, a decision tree is generated, which can be used to make accurate predictions. Decision trees are particularly useful for handling non-linear data. Their simplicity and ease of understanding and implementation contribute to their wide usage in various contexts [21,22]. In the field of intrusion detection systems, decision tree methods have been applied in work by Lee et al. [23], Amor et al. [24], and others [25,26]. The advantages of using decision trees for intrusion detection include their ability to handle large volumes of data, handle both continuous and categorical data, and provide clear and interpretable results. However, decision tree algorithms are susceptible to over-fitting, which can lead to biased results when the training data is imbalanced. Additionally, decision tree algorithms can be computationally intensive, and their performance may degrade with increasing tree size.

Decision Trees are supervised learning algorithms used for both classification and regression tasks. They create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The learning process of a Decision Tree involves the following steps: (i) **Attribute Selection** : The algorithm starts by selecting the best attribute to split the data. This attribute should have the highest predictive power and effectively partition the data into subsets that are more homogeneous with respect to the target variable. The selection is typically based on measures like information gain, Gini index, or entropy; (ii) **Splitting**: Once the attribute is selected, the data is partitioned into subsets based on the attribute values. Each subset represents a different branch or path in the tree, corresponding to a specific attribute value; (iii) **Recursive Construction**: The above steps are repeated recursively for each subset or branch until a stopping criterion is met. This criterion could be reaching a maximum depth, having a minimum number of samples in a leaf node, or achieving a specific level of purity in the subsets; (iv) **Leaf Node Assignment**: At each leaf node, the majority class or the mean value of the target variable is assigned as the predicted value. For classification tasks, the class with the highest frequency is selected, while for regression tasks, the mean or median value is used. The decision-making process in a Decision Tree can be represented using conditional statements. Each internal node represents a decision based on an attribute, and each leaf node represents a prediction. During the learning process, the algorithm constructs the tree by recursively selecting the best attribute, creating branches, and assigning values to leaf nodes. The selection of attributes and splitting points aims to maximize the predictive power and minimize the impurity within each subset. Once the Decision Tree is constructed, it can be used to make predictions on new, unseen data points by traversing the tree based on the attribute values of the data point until reaching a leaf node, where the prediction is obtained. In summary, the learning process of Decision Trees involves selecting the best attribute to split the data, recursively partitioning the data based on attribute values, assigning predictions to leaf nodes, and constructing a tree that represents a set of decision rules for making predictions. Please note that the above explanation provides a general understanding of Decision Tree learning. Decision Trees have various extensions and variations, such as handling missing values, pruning to avoid over-fitting, and ensemble methods like Random Forest and Gradient Boosting, which can be explored in more detail for a comprehensive understanding of Decision Trees.

2.3. Random Forest

Random Forest is an ensemble learning algorithm primarily used for classification tasks [27]. It extends the bagging approach by incorporating feature randomness to con-

struct a forest of uncorrelated decision trees. The Random Forest algorithm combines the outputs of multiple decision trees to generate a final result. A Random Forest consists of a collection of decision trees, with each tree in the ensemble created by sampling data from a training set using a bootstrap sample. Before training, three key hyper-parameters need to be set: the node size, the number of features to be sampled, and the number of trees in the forest. The popularity of Random Forest stems from its ease of use and flexibility in handling both classification and regression problems. Unlike individual decision trees that consider all possible feature splits, Random Forest only selects a subset of features. The prediction process varies depending on the problem type. For regression tasks, the individual decision trees are averaged, while for classification tasks, the predicted class is determined by a majority vote based on the most frequent categorical variable [27]. In the field of intrusion detection systems, Random Forest has been widely utilized and tested with datasets such as KDD 99 [28,29], where it has been compared to SVM. It is also commonly employed with UNSW-NB15 [30,31] and CSE-CIC-IDS2018 [19]. Random Forest exhibits several advantages for intrusion detection, including its ability to handle high-dimensional and complex data, making it suitable for detecting anomalies in large datasets. It also performs well in the presence of noisy data, which is common in intrusion detection systems due to false alarms and other anomalies. However, Random Forest can be slow when processing large datasets, and it requires a significant amount of training data, posing a challenge when working with limited data in intrusion detection systems.

Random Forest is an ensemble learning algorithm that combines the outputs of multiple decision trees to make predictions. It is widely used for both classification and regression tasks. The learning process of Random Forest involves the following steps:

1 Bootstrapped Sampling: Random Forest starts by creating multiple subsets of the original training data through bootstrapped sampling. This sampling technique involves randomly selecting data points from the original dataset with replacement. Each subset is called a bootstrap sample and is used to train a separate decision tree;
2 Random Feature Selection: For each decision tree in the Random Forest, a random subset of features is selected. This process introduces randomness and reduces the correlation between trees. The number of features in the subset is typically determined by a hyper-parameter called "max_features".
3 Decision Tree Construction: Using the bootstrapped sample and the randomly selected feature subset, a decision tree is constructed for each subset. The construction follows the same steps as in the standalone decision tree learning process, including attribute selection, splitting, and recursive construction;
4 Voting or Averaging: Once all the decision trees are constructed, predictions are made by either voting (for classification tasks) or averaging (for regression tasks) the predictions of individual trees. In classification tasks, the class with the highest number of votes is chosen as the final prediction. In regression tasks, the mean or median value of the predictions is taken as the final prediction.

Lets define a Random Forest ensemble with N decision trees. Each decision tree, denoted as T_i, is constructed using a bootstrapped sample D_i from the original training dataset D. Additionally, a random subset of features, denoted as F_i, is selected for each tree. For classification tasks, the Random Forest prediction function can be defined as:

$$y_{pred} = argmax(\frac{sum(T_i(X)}{N}), i = 1toN), \quad (2)$$

where y_{pred} is the predicted class, $T_i(X)$ represents the prediction of tree T_i for input features X, and $argmax(*)$ is the function that returns the class with the highest vote. For regression tasks, the Random Forest prediction function can be defined as:

$$y_{pred} = \frac{sum(T_i(X))}{N}, i = 1toN, \quad (3)$$

where y_{pred} is the predicted value, and $T_i(X)$ represents the prediction of tree T_i for input features X.

The randomness introduced through bootstrapped sampling and random feature selection helps to reduce over-fitting and improve the generalization ability of the Random Forest model. By combining the predictions of multiple trees, Random Forest leverages the wisdom of the crowd and provides more robust and accurate predictions compared to a single decision tree. It is important to note that there are additional considerations in Random Forest, such as hyper-parameter tuning, out-of-bag estimation, and feature importance analysis, which can further enhance the performance and interpretability of the model.

2.4. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a multivariate technique used to classify individual observation vectors into predefined groups based on measurements [32]. Unlike cluster analysis, LDA is a supervised method that relies on a training dataset with pre-assigned groups. LDA, also known as Fisher's linear discriminant, aims to find a linear combination of features that effectively distinguishes between classes of objects. It belongs to the analysis of variance family, where one independent variable is expressed as a linear combination of other features. LDA assumes that the features are independent variables for each observation. The goal of LDA is to create synthetic features, which are linear combinations of the original variables, that best capture the differences between known groups while minimizing the variances within the groups. By using statistical parameters, LDA constructs a boundary between the known classes of training objects. This boundary is defined by a discriminant function, which assigns a score or value to a test object, indicating the group to which the new object should be assigned [33]. LDA has been widely employed in intrusion detection research, particularly in studies using the KDD 99 [34,35], UNSW-NB15 [36], and CSE-CIC-IDS2018 [37] datasets. These studies often compare the performance of LDA with other intrusion detection methods. The main advantage of using LDA for anomaly detection is its effectiveness in handling large datasets and achieving high accuracy. LDA can identify patterns in the data and make predictions based on these patterns, making it a powerful tool for anomaly detection. However, there are also some limitations to consider when using LDA for anomaly detection. One major drawback is that it requires a substantial number of features to be effective. This is because LDA is designed to work with multidimensional data, necessitating a sufficient number of features to capture all relevant patterns and relationships. Additionally, LDA can be sensitive to noisy data, which may lead to false positive anomalies.

Linear Discriminant Analysis is a statistical method used for supervised classification problems. It aims to find a linear combination of features that maximizes the separation between predefined classes. The learning process of LDA involves the following steps:

- Data Preprocessing: LDA requires a training dataset with predefined classes. The training dataset consists of feature vectors X and their corresponding class labels y;
- Class-wise Summary Statistics: For each class in the training dataset, LDA computes class-specific summary statistics. These statistics include the mean vector μ and the covariance matrix Σ of the features within each class;
- Between-class Scatter Matrix: LDA computes the between-class scatter matrix (SB), which measures the separation between different classes. It is defined as the sum of the outer products of the difference between class means and the overall mean, weighted by the number of samples in each class.

$$SB = \sum(Ni(\mu_i - \mu)(\mu_i - \mu)^T) \quad (4)$$

where Ni is the number of samples in class i, μ_i is the mean vector of class i, mu is the overall mean vector.

- Within-class Scatter Matrix: LDA computes the within-class scatter matrix (SW), which measures the spread of the samples within each class. It is defined as the sum of the covariance matrices of each class, weighted by the number of samples in each class.

$$SW = \sum(Ni\Sigma_i), \tag{5}$$

where Σ_i is the covariance matrix of class i.

- Fisher's Criterion: LDA aims to find a projection that maximizes the separation between classes while minimizing the scatter within each class. This is achieved by computing Fisher's criterion, which is the ratio of the determinant of SB to the determinant of SW.

$$J(w) = (w^T SBw)/(w^T SWw), \tag{6}$$

where w is the projection vector.

- Projection Vector: To find the optimal projection vector w, LDA solves the generalized eigenvalue problem:

$$SBw = \lambda SWw, \tag{7}$$

where λ is the eigenvalue corresponding to w.

- Dimensional Reduction: LDA selects the top k eigenvectors corresponding to the largest eigenvalues as the projection vectors. These projection vectors form a lower-dimensional subspace that preserves the most discriminating information between classes.
- Classification: To classify a new observation, LDA projects it onto the subspace spanned by the projection vectors and assigns it to the class with the closest mean in the projected space.

LDA assumes that the features follow a multivariate normal distribution and that the covariance matrices are equal across classes. It seeks to find a linear decision boundary that maximizes the separation between classes while minimizing the overlap within each class. By capturing the underlying structure of the data and finding an optimal projection, LDA provides a powerful tool for classification problems, especially when the classes are well-separated. It is important to note that there are variations of LDA, such as Regularized Discriminant Analysis (RDA) and Quadratic Discriminant Analysis (QDA), which relax certain assumptions and extend the applicability of the method to different types of data distributions.

2.5. K-Nearest Neighbors

The K-nearest neighbors algorithm (KNN) is a non-parametric supervised learning method. It is commonly used for classification and regression problems. KNN classifies unlabeled observations by assigning them to the class of the most similar labeled examples. The similarity is determined based on distance calculations between observations. The algorithm stores the data and identifies the nearest points based on their similarity to classify them into classes. The neighbors are obtained from a set of classes where the object properties are known. The KNN algorithm uses a positive integer, typically small, called K to determine the number of nearest neighbors considered for classification. For example, when K = 1, only the closest neighbor is considered for determining class membership. In regression tasks, the output value is the average of the K neighboring values. To ensure accurate results, it is important to normalize or standardize the dataset since KNN relies on distance calculations. KNN is widely utilized in developing intrusion detection systems (IDS) in computer networks, as seen in previous studies [38–42]. One of the key advantages of KNN is its simplicity and ease of understanding, making it suitable for small-scale or entry-level systems. It is also fast, efficient, and accurate, making it suitable for real-time intrusion detection. Additionally, KNN does not require extensive training or high computational resources, making it a cost-effective solution for intrusion detection. However, the performance of the KNN algorithm heavily relies on the quality of

the training data. Outdated or irrelevant data can impact its effectiveness and may lead to incorrect classifications.

KNN is a supervised learning algorithm used for both classification and regression tasks. It is a non-parametric method that makes predictions based on the similarity between input data points. The learning process of KNN involves the following steps:

- Data Preparation: The training dataset consists of labeled examples, where each example is represented by a feature vector X and its corresponding class label y.
- Choosing the Value of k: The value of k, representing the number of nearest neighbors to consider, needs to be determined. Typically, k is chosen based on domain knowledge or through cross-validation.
- Computing Distance: KNN utilizes a distance metric, such as Euclidean distance or Manhattan distance, to measure the similarity between feature vectors. Lets denote the distance metric as $dist(x_1, x_2)$, where x_1 and x_2 are two feature vectors.
- Finding K Nearest Neighbors: Given an input data point x, the distances between x and all data points in the training set are computed. The k nearest neighbors of x are selected based on the smallest computed distances. Lets denote the set of k nearest neighbors as $N(x)$.
- Voting or Averaging: For classification tasks, KNN employs majority voting to determine the class label of the input data point. It assigns the class label that is most frequent among the class labels of the k nearest neighbors in $N(x)$. Mathematically, the predicted class label for x, denoted as y_{pred}, is given by:

$$y_{pred} = argmax(\sum(y_i \in N(x))I(y_i)) \quad (8)$$

  where y_i represents the class label of the ith neighbor in $N(x)$, $I(*)$ is the indicator function, and $argmax(*)$ returns the class label with the highest count. For regression tasks, KNN takes the average of the target values of the k nearest neighbors to make a prediction. Mathematically, the predicted value for x, denoted as y_{pred}, is given by:

$$y_{pred} = (1/k)\sum(y_i \in N(x))y_i \quad (9)$$

  where y_i represents the target value of the ith neighbor in $N(x)$.
- Handling Ties: In cases where there is a tie in the number of votes for different classes during majority voting, various strategies can be employed. For example, KNN may select the class label of the nearest neighbor among the ties or use weighted voting based on the distances.
- Predicting with the Trained Model: Once the KNN model is trained on the training dataset, it can be used to make predictions on new, unseen data points by following the steps mentioned above.

It is important to note that the KNN algorithm does not involve an explicit learning phase where a model is built. Instead, it stores the training data points and their corresponding labels, and the actual learning occurs during the prediction phase by comparing the input data point to the stored training data. KNN is a versatile algorithm that can handle both numerical and categorical data. However, it is important to normalize or scale the feature vectors to ensure that features with larger scales do not dominate the distance calculations. Additionally, the choice of distance metric and the value of k can significantly impact the algorithm's performance.

2.6. Artificial Neural Network

Artificial Neural Network (ANN) is extensively discussed in the literature, particularly in the context of classification. It is a mathematical model comprising interconnected artificial neurons, designed for information processing and solving artificial intelligence problems. Neural networks possess remarkable capabilities such as self-organization, adaptive learning, and fault-tolerance. These qualities make them popular for pattern

recognition and classification tasks [43]. There are various types of Intrusion Detection Systems (IDS) based on ANN, implemented with datasets like KDD 99 [41,44,45], UNSW-NB15 [46,47], and CSE-CIC-IDS2018 [48,49]. One of the strengths of ANN is its ability to handle non-linear relationships in data, which can pose challenges for traditional statistical methods. However, ANN can be computationally expensive, especially when dealing with real-time or high-volume data processing. Furthermore, the interpretability and understanding of ANN models can be challenging, which may hinder their implementation in certain applications.

Feed-forward neural networks are a type of artificial neural network in which information propagates only in one direction, forward, without cycles or feedback. The learning process of feed-forward neural networks occurs through a process known as "back-propagation" or error back-propagation. The learning process starts with the initialization of the weights of the connections between the neurons in the network. The initial weights can be randomly assigned or using a specific initialization method. The goal of learning is to optimize these weights so that the network can produce accurate predictions for a given problem. During the training phase, input data is fed into the network and propagates through the various layers of neurons until it reaches the output layer. Each neuron in the output layer produces a prediction based on the connection weights and the neuron's activation. The produced output is then compared to the desired output (class label) to calculate the prediction error. Subsequently, the error is backward propagated through the network using the error back-propagation mechanism. During this phase, the error is divided and assigned proportionally to the neuron connections to determine the effect of each connection on the overall error of the network. This is done using the weight update rule, which allows adjusting the weights based on the incurred error. The weight update is performed using an optimization algorithm such as gradient descent. The objective is to minimize a cost function that represents the overall error of the network. During the weight update, the gradients of the cost function with respect to the weights are computed and used to update the weights so that the error decreases. This process of error propagation and weight update is repeated for different training examples until the network achieves good generalization ability, meaning it can produce accurate predictions for unseen data. In summary, the learning process of feed-forward neural networks involves the forward propagation of data through the network, calculation of prediction error, error back-propagation to update the connection weights, and optimization of the cost function to minimize the overall error of the network. This process enables the network to learn and adapt to the training data, improving its ability to make accurate predictions. Formally, lets denote the input to the network as x, the weights of the connections as W, the activation function of a neuron as f, and the desired output as y. The output of a neuron j in layer l can be calculated as:

$$z_j^l = f(a_j^l) = f(\sum_{i=1}^{n_{l-1}} W_{ji}^l a_i^{l-1} + b_j^l) \tag{10}$$

where z_j^l is the weighted sum of inputs to neuron j in layer l, a_j^l is the activation of neuron j in layer l, W_{ji}^l is the weight of the connection between neuron i in layer $l-1$ and neuron j in layer l, a_i^{l-1} is the activation of neuron i in layer $l-1$, and b_j^l is the bias term for neuron j in layer l. The prediction error for a single training example can be measured using a suitable loss function, such as mean squared error:

$$E = \frac{1}{2} \sum_{k=1}^{n_{\text{output}}} (y_k - a_k^L)^2 \tag{11}$$

where n_{output} is the number of neurons in the output layer, y_k is the desired output for neuron k in the output layer, and a_k^L is the actual output of neuron k in the output layer. The weight update rule for the connection weights can be expressed as:

$$\Delta W_{ji}^l = -\eta \frac{\partial E}{\partial W_{ji}^l} \tag{12}$$

where η is the learning rate, and $\frac{\partial E}{\partial W_{ji}^l}$ represents the derivative of the cost function with respect to the weight W_{ji}^l. The back-propagation algorithm computes the partial derivatives of the cost function with respect to the weights in a layer-by-layer fashion, starting from the output layer and moving backward to the input layer. This process is achieved by applying the chain rule to calculate the gradients. The weight update rule can be further refined using variations of gradient descent algorithms, such as stochastic gradient descent (SGD) or mini-batch gradient descent, which update the weights based on subsets of training examples rather than the entire dataset. These variations aim to improve the convergence speed and overcome issues related to large datasets. The learning process continues iteratively, with multiple passes through the training data, until the network converges to a state where the prediction error is minimized. At this point, the network has learned the underlying patterns and relationships in the training data and can make accurate predictions for new, unseen data. In summary, the learning process of feed-forward neural networks involves forward propagation of data, calculation of prediction error, back-propagation of the error to update the weights using the gradient descent algorithm, and repetition of this process until convergence. Through this iterative process, the network learns to make accurate predictions and generalize to unseen data. Please note that the above explanation provides a high-level overview of feedforward neural network learning and the associated mathematical concepts. There are various additional aspects, techniques, and variations that can be explored in greater detail for a comprehensive understanding of neural network learning.

3. Selected Datasets

To effectively train and develop machine learning algorithms, a significant amount of data is essential. The quantity and quality of data play a crucial role in determining the performance of any machine learning problem, as the results heavily rely on the available data. In the field of Intrusion Detection Systems (IDS), three primary datasets have gained wide recognition in the literature. These datasets serve as benchmarks for evaluating machine learning-based IDS systems, ensuring consistency and comparability across different approaches. It is important to note that the workflow described in this context is specifically tailored for supervised learning. This implies that the observations used to train the presented machine learning models are assumed to have correct labels assigned beforehand. This allows for the evaluation of performance metrics post-training. However, it is worth mentioning that this workflow can be adapted to accommodate unsupervised learning approaches as well. In such cases, clustering algorithms are commonly employed to obtain initial groupings and handle missing labels. The datasets discussed below, which serve as the basis for training the models used in subsequent performance comparisons, are widely recognized and extensively used for evaluating anomaly and intrusion detection algorithms. Consequently, these datasets have become the standard benchmarks in this domain, adding credibility to the presented work and enhancing its validity.

3.1. KDD 99

The KDD 99 dataset comprises a comprehensive collection of 25,192 TCP/IP connections (observations) derived from a simulated LAN environment resembling a typical US Air Force setup. This network was specifically designed to mimic real-world conditions and was subjected to various attacks to create a diverse dataset. In this context, a connection refers to a sequence of TCP packets transmitted between a source IP address and a target

IP address under a well-defined protocol. These connections are characterized by their start and end duration, along with the data exchanged. Each connection is labeled as either "normal" or "anomalous" based on its behavior. Each recorded connection contains approximately 100 bytes of data. A total of 41 features are extracted from each TCP/IP connection, consisting of 38 quantitative features and 3 qualitative features. The class variable, which determines if a connection is considered an intrusion or not, has two categories: "Normal" and "Anomalous". To provide a comprehensive overview of the dataset features, Tables 1 and 2 present a list of these features along with brief descriptions. The features are categorized as "Basic", "Content-based", "Time-based", and "Connection-based". Additionally, the tables indicate whether each feature is "Continuous" (C) or "Discrete" (D) in nature.

Table 1. Features Description KDD 99 part 1.

Feature Name	Type	Description
Basic Features		
Duration	C	Length of the connection
Protocol-type	D	Type of protocol
Service	D	Network service at the dest.
Flag	D	Normal or error status of the connection
Src-bytes	C	N. of data bytes from source to destination
Dst-bytes	C	N. of data bytes from destination to source
Land	D	1 if connection is from/to the same host/port; 0 otherwise
Wrong fragment	C	N. of "wrong" fragments
Urgen	C	N. of urgent packets
Content-based Features		
Hot	C	N. of "hot" indicators
Num-failed-logins	C	N. of failed login attempts
Logged-in	D	1 if successfully logged-in; 0 otherwise
Num-compromised	C	N. of compromised conditions
Root-shell	D	1 if root-shell is obtained; 0 otherwise
Su-attempted	D	1 if "su root" command
Num-root	C	N.of "root" accesses
Num-file-creations	C	N. of file creation operations
Num-shells	C	N. of shell prompt
Num-access-files	C	N. of operations on access control files
Num-outbound-cmds	C	N. of outbound commands in an FTP session
Is-host-login	D	1 if login belongs to the "hot" list; 0 otherwise
Is-guest-login	D	1 if the login is a "guest" login; 0 otherwise

Table 2. Features Description KDD 99 part 2.

Feature Name	Type	Description
Time-based Features		
Count	C	N. of connect. to the same host as the current connect. in the past 2 s
Srv-count	C	N. of connect. to the same service as the current connection. in past 2 s.
Serror-rate	C	% of connect. that have SYN errors (same-host connect.)
Srv-error-rate	C	% of connect. that have SYN errors (same-service connect.)
Rerror-rate	C	% of connect. that have REJ errors (same-host connect.)
Srv-error-rate	C	% of connect. that have REJ errors (same-service connect.)
Same-srv-rate	C	% of connect. to the same service (same-host connect.)
Diff-srv-rate	C	% of connect. to different services (same-host connect.)
Srv-diff-host-rate	C	% of connect. to different hosts (same-service connect.)

Table 2. *Cont.*

Feature Name	Type	Description
Connection-based features		
Dst-host-count	C	Count of dest. hosts
Dst-host-srv-count	C	Srv-count for dest. host
Dst-host-same-srv-rate	C	Same-srv-rate for dest. host
Dst-host-diff-srv-rate	C	Diff-srv-rate for dest. host
Dst-host-same-src-port-rate	C	Same-src-port-rate for dest. host
Dst-host-srv-diff-host-rate	C	Diff-host-rate for dest. host
Dst-host-error-rate	C	Serror-rate for dest. host
Dst-host-srv-error-rate	C	Srv-error-rate for dest. host
Dst-host-rerror-rate	C	Rerror-rate for dest. host
Dst-host-srv-rerror-rate	C	Srv-rerror-rate for dest. host

3.2. UNSW-NB15

The UNSW-NB15 dataset is a widely recognized and freely available benchmark for evaluating intrusion detection systems. It was created using the IXIA Perfect Storm tool in the Cyber Range Lab at Canberra University. This dataset combines real-world normal activities with synthetic contemporary attack behaviors to provide a comprehensive testing environment. To generate the dataset, the Tcpdump tool was employed to capture 100 GB of raw traffic data. The dataset includes nine different types of attacks, namely Analysis, Backdoors, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms.

The Argus and Bro-IDS tools were utilized, resulting in the generation of 47 features through the implementation of twelve algorithms. Tables 3 and 4 present a detailed list of the dataset features, categorized into Flow features, Basic features, Content-based features, Time-based features, General purpose features, and Connection-based features. Each feature is characterized by its type, such as Nominal (N), Integer (I), Float (F), Timestamp (T), or Binary (B). The dataset contains a total of 2,540,047 records, which are stored in four .csv files. Additionally, raw traffic records in .pcap format are also available in the repository. The dataset size corresponds to 2,540,047 observations with 47 features. More information about the dataset can be found in the reference [50].

Table 3. Features Description UNSW-NB15 part 1.

Feature Name	Type	Description
Flow Features		
srcip	N	Source IP address
sport	I	Source port number
dstip	N	Dest. IP address
dsport	I	Dest. port number
proto	N	Transaction protocol
Basic Features		
state	N	State and its dependent protocol
dur	F	Record total duration
sbyte	I	Source to dest. bytes
dbytes	I	Dest. to source bytes
state	I	Source to dest. time to live
dttl	I	Dest. to source time to live
sloss	I	Source packets retransmitted or dropped
dloss	I	Dest. packets retransmitted or dropped
service	N	http, ftp, ssh, dns ..., else (-)
load	F	Source bits per second
load	F	Dest. bits per second
spkts	I	Source to dest. packet count
dpkts	I	Dest. to source packet count

Table 3. *Cont.*

Feature Name	Type	Description
Content-based Features		
swin	I	Source TCP window advert.
dwin	I	Dest. TCP window advert.
step	I	1 Source TCP sequence num.
dtcpb	I	Dest. TCP sequence num.
means	I	Mean of the flow packet size transmitted by the src
means	I	Mean of the flow packet size transmitted by the dst
trans_depth	I	Depth into the connection of http request/response transaction
res_bdy_len	I	Size of the data transferred from the server's http service
Time-based Features		
sjit	F	Source jitter
djit	F	Dest. jitter
stime	T	Record start time
ltime	T	Record last time
sintpkt	F	Source inter-packet arrival time
dintpkt	F	Dest. inter-packet arrival time
tcprtt	F	The sum of "synack" and "ackdat" of the TCP
synack	F	Time between the SYN and the SYN_ACK packets of the TCP
ackdat	F	Time between the SYN_ACK and the ACK packets of the TCP
General purpose Features		
is_sm_ips_ports	B	If source equals to dest. IP addresses and port n. are equal, it is 1
ct_state_ttl	I	N. for each state according to specific range of values
ct_flw_http_mthd	I	N. of flows that has methods such as Get and Post in HTTP service
is_ftp_login	B	If the ftp session is accessed by user and password then 1 else 0
ct_ftp_cmd	I	N. of flows that has a command in ftp session

Table 4. Features Description UNSW-NB15 part 2.

Feature Name	Type	Description
Connection-based Features		
ct_srv_src	I	N. of connect. that contain the same service and dest. in address 100 connect. according to the last time
ct_srv_dst	I	N. of connect. of the same dest. address in 100 connect. according to the last time
ct_dst_ltm	I	N. of connect. of the same source address in 100 connect. according to the last time
ct_src_dport_ltm	I	N. of connect. of the same source address and the dest. port in 100 connect. according to the last time
ct_dst_sport_ltm	I	N. of connect. of the same dest. address and the source port in 100 connect according to the last time
ct_dst_src_ltm	I	N. of connect. of the same source and the dest. address in 100 connect.

3.3. CSE-CIC-IDS 2018

This dataset encompasses a comprehensive collection of observations, encompassing both normal network traffic and 14 distinct types of attacks: SHH Brute Force, SQL Injection, Infiltration, FTP Brute Force, DoS attacks Slowloris, DoS attacks Slow HTTP Test, DoS attacks Hulk, DoS attacks GoldenEye, DDoS attacks LOIC UDP, DDoS attacks LOIC HTTP, DDoS attacks HOIC, Brute Force XSS, Brute Force Web, and Bot. The attacking infrastructure involved in this dataset consists of 50 machines, while the victim organization comprises 5 departments with 420 machines and 30 servers. The dataset includes network traffic and system logs from each machine, with 75 features extracted from the captured traffic using CICFlowMeter-V3. CICFlowMeter-V3, a Java-based network traffic flow generator, offers flexibility in feature selection, addition, and control over the flow timeout duration. Within the CSE-CIC-IDS 2018 dataset, profiles are employed to systematically generate datasets. These datasets provide intricate intrusion descriptions and abstract distribution models for applications, protocols, or lower-level network entities. The finalized dataset is presented in CSV format, with each flow represented by four columns: Destination Port, Protocol, Timestamp, and over 75 network traffic features mentioned previously. A total of

78 features are extracted, as listed in Tables 5 and 6. The dataset encompasses a voluminous 16,233,002 observations, each associated with 78 features. Notably, the last column signifies the categories assigned to each observation [51].

Table 5. Features Description CSE-CIC-IDS2018 part 2.

Feature Name	Description
fw_blk_rate_avg	Average number of bulk rates in the forward direction
bw_byt_blk_avg	Average number of bytes bulk rate in the backward direction
bw_pkt_blk_avg	Average number of packets bulk rate in the backward direction
bw_blk_rate_avg	Average number of bulk rate in the backward direction
subfl_fw_pk	The average number of packets in a sub-flow in the forward direction
subfl_fw_byt	The average number of bytes in a sub-flow in the forward direction
subfl_bw_pkt	The average number of packets in a sub-flow in the backward direction
subfl_bw_byt	The average number of bytes in a sub-flow in the backward direction
fw_win_byt	Number of bytes sent in initial window in the forward direction
bw_win_byt	Number of bytes sent in initial window in the backward direction
Fw_act_pkt	Number of packets with at least 1 byte of TCP data payload in the forward direction
fw_seg_min	Minimum segment size observed in the forward direction
atv_avg	Meantime, a flow was active before becoming idle
atv_std	Standard deviation time a flow was active before becoming idle
atv_max	Maximum time a flow was active before becoming idle
atv_min	Minimum time a flow was active before becoming idle
idl_avg	Meantime, a flow was idle before becoming active
idl_std	Standard deviation time a flow was idle before becoming active
idl_max	Maximum time a flow was idle before becoming active
idl_min	Minimum time a flow was idle before becoming active
dstPort	Dest. address
Protocol	Type of protocol
Timestmap	Date, and time of the event

Table 6. Features Description CSE-CIC-IDS2018 part 1.

Feature Name	Description
dst_port	Destination Port proto N. of protocol
time_stmp	Timestamp of the event
fl_dur	Flow duration
fl_pkt_s	Flow packets rate that is the number of packets transferred per second
fl_iat_avg	Average time between two flows
fl_iat_std	Standard deviation time two flows
fl_iat_max	Maximum time between two flows
fl_iat_min	Minimum time between two flows
fw_iat_tot	Total time between two packets sent in the forward direction
fw_iat_avg	Mean time between two packets sent in the forward direction
fw_iat_std	Standard deviation time between two packets sent in the forward direction
fw_iat_max	Maximum time between two packets sent in the forward direction
fw_iat_min	Minimum time between two packets sent in the forward direction
bw_iat_tot	Total time between two packets sent in the backward direction
bw_iat_avg	Mean time between two packets sent in the backward direction
bw_iat_std	Standard deviation time between two packets sent in the backward direction
bw_iat_max	Maximum time between two packets sent in the backward direction
bw_iat_min	Minimum time between two packets sent in the backward direction
fw_psh_flag	Number of times the PSH flag was set in packets traveling in the forward direction (0 for UDP)
bw_psh_flag	Number of times the PSH flag was set in packets traveling in the backward direction (0 for UDP)
fw_urg_flag	Number of times the URG flag was set in packets traveling in the forward direction (0 for UDP)
bw_urg_flag	Number of times the URG flag was set in packets traveling in the backward direction (0 for UDP)
fw_hdr_len	Total bytes used for headers in the forward direction
bw_hdr_len	Total bytes used for headers in the forward direction
fw_pkt_s	Number of forwarding packets per second
bw_pkt_s	Number of backward packets per second
pkt_len_min	Minimum length of a flow
pkt_len_max	Maximum length of a flow
pkt_len_avg	Mean length of a flow
pkt_len_std	Standard deviation length of a flow

Table 6. *Cont.*

Feature Name	Description
pkt_len_va	Minimum inter-arrival time of packet
fin_cnt	Number of packets with FIN
syn_cnt	Number of packets with SYN
rst_cnt	Number of packets with RST
pst_cnt	Number of packets with PUSH
ack_cnt	Number of packets with ACK
urg_cnt	Number of packets with URG
cwe_cnt	Number of packets with CWE
ece_cnt	Number of packets with ECE
down_up_ratio	Download and upload ratio
pkt_size_avg	Average size of packet
fw_seg_avg	Average size observed in the forward direction
bw_seg_avg	Average size observed in the backward direction
fw_byt_blk_avg	Average number of bytes bulk rate in the forward direction
fw_pkt_blk_avg	Average number of packets bulk rate in the forward direction

4. ML Performance Evaluation

In the field of machine learning, the general focus is to predict an outcome using the available set of data. For classification, the most common setting involves only two classes, although there may be more than two. In this last case, the issue changes his name and is called "multi-class classification" [52]. Once the ML models have been trained, it is necessary to test them and use metrics to compare the performance of each algorithm. The analysis in the case of IDS can be carried out in two ways: binary performance and multi-class performance. The case of binary classification is applicable to all three datasets in this work since only the binary labeling of each observation traffic is required. In the case of multi-class classification, however, it is necessary that in each dataset, in addition to labeling each observation of normal traffic, each observation of attack must also be labeled with its specific type. In KDD 99, there is no labeling of attack observations. Multi-class classification cannot be applied.

4.1. Binary Classification Metric

The models are only trained to distinguish normal data traffic from abnormal data traffic using the binary labeling. In binary classification problem for network IDS, if *Norm* (Normal traffic observation) and *Att* (Attack traffic observation) are the possible labels for each observation:

- *TP* (true positive): if the model predicts *Norm*, it is also the correct answer.
- *FP* (false positive): if the model predicts *Norm*, the correct answer is *Att*.
- *TN* (true negative): if the model predicts *Att*, and it is also the correct answer
- *FN* (false negative): if the model predicts *Att*, the correct answer is *Norm*.

From this definition of the prediction result by the classification model, some performance indices are defined to evaluate the model itself in terms of quality in classifying ability. In this article we refer to ROC-like indexes [41,53], explained in the following. Sensitivity (SE) or *true-positive rate* is the ratio of *T-Norm* to total traffic normal observations in the data:

$$SE = \frac{TP}{TP + FN} \tag{13}$$

Specificity (SP) or *true-negative rate* is the ratio of *TAtt* to total traffic normal observations in the data:

$$SP = \frac{TN}{TN + FP} \tag{14}$$

Precision (PR) or *positive-predictive value* is the proportion of normal traffic predicted observations to the total predicted

$$Pr = \frac{TP}{TN + FP} \tag{15}$$

Accuracy (AC) is the fraction of correctly identified results (attack and normal traffic).

$$AC = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$

4.2. Multi-Class Classification Metric

In this scenario, the models are trained to classify all specific types of attacks present in the given dataset. The performance evaluation follows the same principles as in the binary case. However, certain clarifications are necessary when defining performance metrics in the multi-class case. When extending a binary metric to multi-class problems, the data is treated as a collection of binary problems using the One-Vs-All (OVA) approach. In OVA classification, a binary classifier model is created for each class in the dataset.

Based on these new classification labels, the same performance metrics used in the binary case are computed for each class in the dataset [52,54]. After obtaining the binary results for each class, there are two different approaches that can be taken:

- Macro-average: This method simply calculates the average of the binary metrics, assigning equal weight to each class. It can be particularly useful when infrequent classes hold significance, as it highlights their performance. With the *Macro-average*, the effect of the most frequent classes is considered equally important as that of the least frequent ones.
- Micro-average: In this approach, each sample-class pair contributes equally to the overall metric. Instead of summing the metrics by class, it sums the numerators and denominators that constitute the metrics by class to calculate an overall quotient. The *Micro-average* is often preferred in multi-label or multi-class classification settings, where the majority class must not dominate the evaluation.

The difference between these methods lies in how they weigh each class or sample. *Macro-average* assigns equal weight to each class, while *Micro-average* assigns equal weight to each sample. If the dataset is perfectly balanced, both *Macro* and *Micro* averages yield the same result. In this work, the performance of all attack classes in the two multi-class datasets under analysis will be evaluated. Hence, the contribution of each class's performance value is equally important compared to the others. In the upcoming sections, balanced datasets will be used, eliminating any distinction between *Macro* and *Micro* indices. The following metrics will be employed as indicators of multi-class performance: *M-Sensitivity* (M-SE), *M-Specificity* (M-SP), *M-Precision* (M-PR), and *M-Accuracy* (M-AC). The "M" prefix indicates the term *Mean*, encompassing both *Micro* and *Macro* without distinction (see Figure 2).

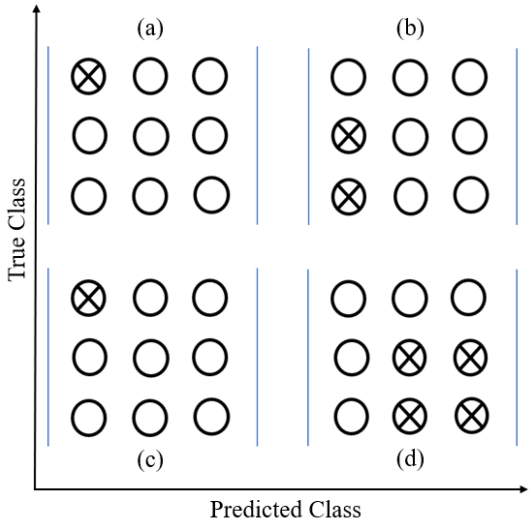

Figure 2. Graphical interpretation of the definition of True-Positive (**a**), False-Positive (**b**), False-Negative (**c**), and True-Negative (**d**), referring to class 1 out of 3.

5. Dataset Manipulation

In machine learning, it is essential to preprocess the data to enhance the performance of classification methods. Raw data in datasets often cannot be directly used by all machine learning techniques. Therefore, it becomes necessary to manipulate the datasets through various operations to ensure proper interpretation by the machine learning algorithms. In the context of this work, focusing on the three datasets used, three fundamental steps are crucial to ensure the correct execution of the learning phase [53].

5.1. Variable Encoding

For many machine learning algorithms, it is necessary to convert the data into a numerical form, specifically quantitative values. In the literature, various methods have been proposed to achieve this, but two widely used methodologies stand out. One of these methodologies is "one-hot encoding", which involves creating dummy variables to represent categorical variables. A categorical variable with K levels (i.e., having a set of K possible values) is transformed into an array of K binary variables, where only one variable is active (set to 1) at a time. While there are more efficient encoding techniques available, one-hot encoding has the advantage of preserving all the characteristics of the dataset. However, a drawback of one-hot encoding is that it can significantly increase the dataset size, depending on the number of values the variable can take. This can impact computational resources and memory requirements [55–57]. Figure 3 provides an example illustrating the one-hot encoding method.

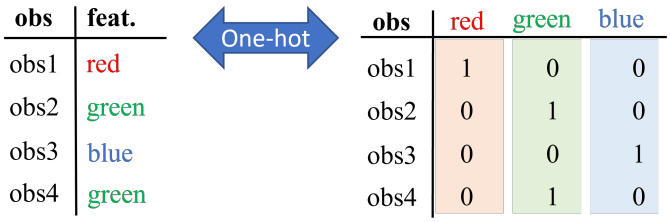

Figure 3. Example of One-hot Encoding.

Another widely used method is "label encoding". In label encoding, each unique value in a categorical variable, denoted as set G, is assigned a numerical value ranging from 1 to K, where K represents the total number of different values in set G. The primary advantage of label encoding is that it does not increase the dataset size, as it simply replaces the categorical values with corresponding numerical labels. However, one potential drawback of this method is that it may inadvertently introduce relationships or order among the values that were not originally present in the data [58,59]. Figure 4 provides an example illustrating the label encoding technique.

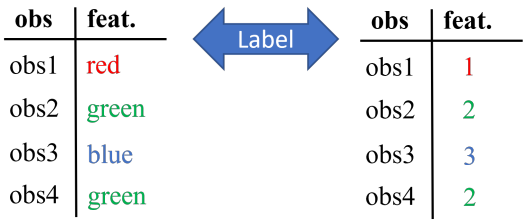

Figure 4. Example of Labeled Encoding.

In this study, we have employed the "one-hot encoding" technique for the KDD 99 and UNSW-NB15 datasets, as they contain qualitative features that require encoding. This decision was made with the aim of preserving the relationships between the features in the original datasets, despite the increase in the overall number of features. Specifically, the number of features expanded from 41 to 118 for the KDD 99 dataset and from 47 to

254 for the UNSW-NB15 dataset. It is important to note that the CSE-CIC-IDS 2018 dataset does not have any qualitative features, so no encoding technique was applied to its data.

5.2. Data Scaling

In the field of machine learning, it is common practice to pre-process the data by scaling the features to eliminate redundancy, improve stability, and aid convergence of algorithms. In this study, all the features used underwent scaling before being utilized. This step is essential to ensure that the features are on the same relevance level and can be compared effectively. In the literature, various techniques exist for feature scaling, but two main methods are commonly employed in machine learning. One of these methods is *normalization*, also known as Min-Max scaling. It involves shifting and re-scaling the values of the features so that they fall within the range of 0 to 1. This scaling technique helps to ensure that all features have equal importance and are comparable across the same scale.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{17}$$

In the Equation (17), the symbol X represents the current value of the feature, X_{max} and X_{min} denote the maximum and minimum values of the feature column in the training set, respectively, and X' represents the final value of the scaled feature. Normalization is applied when the data distribution is not assumed to follow a Gaussian distribution. Another commonly used scaling technique is *standardization*, which involves centering the values around the mean and dividing them by the standard deviation. This technique reshapes each value within its column using a common scale without losing any information or altering the differences in the ranges. Standardization is particularly useful when dealing with features that exhibit a Gaussian distribution.

$$X' = \frac{X - \mu}{\sigma} \tag{18}$$

In the Equation (18), the symbol X denotes the current value of the feature, μ and σ represent the mean and standard deviation of the feature column in the training set, respectively, and X' represents the final value of the scaled feature. Standardization does not impose any specific range on the values, unlike normalization. It is particularly useful when dealing with data that follows a Gaussian distribution. The decision to use either normalization or standardization depends on the specific problem and dataset at hand. There is no definitive rule for choosing between the two techniques. It is generally recommended to try both approaches and compare their performance on the given task. In this work, both normalization and standardization were applied to evaluate their suitability for the case under examination. Ultimately, normalization was chosen because some features in the datasets had NaN (Not a Number) values after standardization. By opting for normalization, it was possible to retain as much information as possible from the original datasets and avoid discarding any features during the analysis.

5.3. Dataset Balancing

An imbalanced training dataset refers to a scenario where the distribution of examples across different classes is uneven or skewed. This can range from a slight bias to a severe imbalance, where the majority class has a significantly larger number of examples compared to the minority class(es). Such an imbalance poses a challenge for classification tasks because many machine learning algorithms assume an equal number of examples for each class, leading to poor predictive performance, particularly for the minority class. The issue is compounded by the fact that the minority class is often more important, and misclassification errors for this class have a higher impact on the problem at hand than errors for the majority class [60]. In supervised machine learning methods, having a balanced dataset during the training phase is crucial to achieve good classification

performance. Considering the specific datasets analyzed in this work, Table 7 presents the percentage distribution of attacks for binary classification. It is evident that only the KDD 99 dataset is balanced in terms of attack-normal observations. On the other hand, the UNSW-NB15 and CSE-CIC-IDS 2018 datasets exhibit a significant imbalance, with a strong bias towards normal traffic observations.

Table 7. Traffic percent distribution in Binary classification.

KDD 99		UNSW-NB15		CSE-CIC-IDS 2018	
Normal	*Attack*	*Normal*	*Attack*	*Normal*	*Attack*
54.4%	46.6%	87.4%	12.6%	83.1%	16.9%

To address the imbalance issue in the datasets (see Figure 5), a solution was implemented by considering the total number of attacks as the upper limit for selecting normal observations randomly. In the UNSW-NB15 dataset, the total number of attacks is 321,283. Thus, the final dataset is constructed to have an equal number of attacks (321,283) and normal observations. Similarly, in the CSE-CIC-IDS 2018 dataset, which has a total of 2,748,294 attacks, an equal number of attacks and normal observations is chosen. As a result, two reduced datasets are obtained, where the number of normal traffic observations is lower than the original dataset, but the datasets are perfectly balanced with a 50% representation of normal observations and 50 % attacks for the training phase. The same imbalance challenge arises in the multi-class case. However, it should be noted that the KDD 99 dataset is excluded from the multi-class analysis as it only supports binary classification. Tables 8 and 9 provide the percentages of each individual attack type in the UNSW-NB15 and CSE-CIC-IDS 2018 datasets, respectively, compared to the total number of attacks.

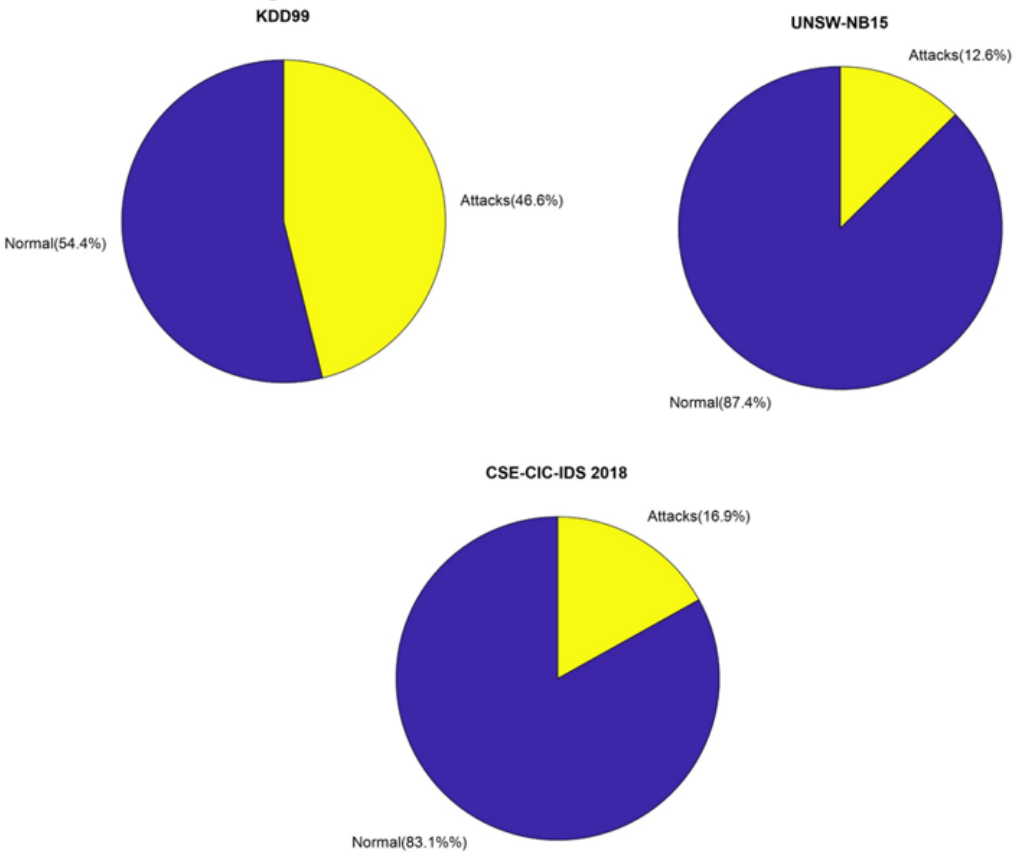

Figure 5. Proportion between Normal and Attack observations.

These tables demonstrate the existing imbalance in the datasets, which necessitates balancing the classes for effective analysis and classification.

Table 8. Traffic percent distribution in Multi-class classification attacks in UNSW-NB15.

UNSW-NB15	
Attack Type	**Percent**
Analysis	0.83%
Backdoors	0.17%
DoS	5.10%
Exploits	13.86%
Fuzzers	7.55%
Generic	67.07%
Reconnaissance	4.35%
Shellcode	0.47%
Worms	0.05%
Total	**100%**

Table 9. Traffic percent distribution in Multi-class classification attacks in CSE-CIC-IDS 2018.

CSE-CIC-IDS 2018	
Attack Type	**Percent**
Bot	10.413%
Brute Force Web	0.022%
Brute Force XSS	0.008%
DDoS attack HOIC	24.961%
DDoS attack LOIC HTTP	20.965%
DDoS attack LOIC UDP	0.063%
DoS attack GoldenEye	1.510%
DoS attack Hulk	16.807%
DoS attack SlowHTTP Test	5.090%
DoS attack Slowloris	0.400%
FTP Brute Force	7.036%
Infiltration	5.891%
SQL Injection	0.003%
SSH Brute Force	6.825%
Total	**100%**

As observed in the UNSW-NB15 and CSE-CIC-IDS 2018 datasets, the distribution of attacks across different categories is highly unbalanced. Some attack categories have a very small number of observations, which is statistically insignificant for the analysis conducted in this work. For instance, the "Worms" attack category in UNSW-NB15 constitutes only 0.05% of the dataset, equivalent to 174 observations. Similarly, the "SQL Injection" category in CSE-CIC-IDS 2018 comprises a mere 0.003% of the dataset, corresponding to 87 observations. Having such low absolute numbers poses a challenge for machine learning algorithms to effectively learn the distinct characteristics of these classes during the supervised training phase. Insufficient samples hinder the algorithm's ability to accurately classify these attack categories. To address this issue, a solution was proposed to set a lower limit threshold of 10,000 observations, ensuring an adequate number of samples for machine learning training. Attack categories with observation counts below this threshold were removed. For the remaining attack categories, the number of observations was set equal to the category with the lowest count above the threshold. In UNSW-NB15,

the "Reconnaissance" category has the lowest count above the threshold, which is 16,353. Therefore, each category in the resulting training dataset contained 16,353 observations. In CSE-CIC-IDS 2018, the "DoS attack Slowloris" category has the lowest count above the threshold, which is 10,990. Similarly, each category in the resulting training dataset consisted of 10,990 observations. Tables 10–14 provide a list of the attack categories remaining after the balancing procedure for UNSW-NB15 and CSE-CIC-IDS 2018, respectively.

Table 10. Attack categories remained after the UNSW-NB15 dataset balancing.

UNSW-NB15	
Attack Type	**Percent**
DoS	20%
Exploits	20%
Fuzzer	20%
Generic	20%
Reconnaissance	20%
Total	**100%**

Table 11. Binary Classification Results.

	KDD 99				**UNSW-NB15**				**CSE-CIC-IDS 2018**			
	SE	*SP*	*PR*	*AC*	*SE*	*SP*	*PR*	*AC*	*SE*	*SP*	*PR*	*AC*
SVM	98.69%	95.60%	98.45%	97.25%	99.50%	99.22%	86.80%	99.49%	99.71%	96.72%	95.51%	97.95%
DT	99.65%	**99.60%**	99.60%	99.63%	99.93%	98.30%	98.0%	99.88%	**99.99%**	99.99%	**99.99%**	**99.99%**
RF	**99.90%**	99.57%	**99.89%**	**99.75%**	**99.97%**	99.25%	**99.08%**	**99.95%**	99.90%	99.57%	99.89%	99.75%
LDA	98.44%	95.04%	98.15%	96.85%	99.45%	99.46%	85.43%	99.45%	99.45%	99.46%	85.43%	99.45%
KNN	99.36%	98.97%	99.26%	99.18%	93.71%	99.85%	95.23%	99.65%	99.85%	93.71%	95.23%	99.65%
ANN SL	99.07%	98.72%	98.90%	98.92%	99.46%	**100%**	85.97%	99.48%	99.86%	99.99%	99.81%	99.92%
ANN ML	99.46%	99.30%	99.36%	99.39%	99.94%	99.46%	86.01%	99.45%	99.93%	**100%**	99.91%	99.96%

Table 12. Multi-class Classification Results.

	UNSW-NB15				**CSE-CIC-IDS 2018**			
	M-SE	*M-SP*	*M-PR*	*M-AC*	*M-SE*	*M-SP*	*M-PR*	*M-AC*
SVM	75.52%	93.88%	77.44%	75.52%	86.67%	98.52%	90.17%	86.68%
DT	79.74%	94.93%	82.10%	79.74%	**94.99%**	**99.44%**	**96.20%**	**94.99%**
RF	**80.20%**	**95.05%**	**83.08%**	**80.20**%	94.17%	99.35%	95.08%	94.20%
LDA	71.34%	92.83%	73.07%	71.34%	85.36%	98.37%	86.80%	85.34%
KNN	69.03%	92.26%	69.80%	69.04%	90.21%	98.92%	90.43%	90.20%
ANN ML	79.85%	94.97%	82.20%	79.90%	89.62%	98.84%	91.31%	89.60%

Table 13. Computation Time in ms for 1000 observation test.

	KDD 99	**UNSW-NB15**		**CSE-CIC-IDS 2018**	
	Binary	*Binary*	*Multiclass*	*Binary*	*Multiclass*
SVM	24.08 ms	26.70 ms	29.72 ms	44.40 ms	383.51 ms
DT	**07.54 ms**	**10.59 ms**	**17.31 ms**	**04.44 ms**	**07.78 ms**
RF	77.82 ms	85.67 ms	105.82 ms	80.30 ms	136.84 ms
LDA	10.48 ms	20.62 ms	33.23 ms	07.38 ms	11.30 ms
KNN	453.12 ms	2806 ms	30,902 ms	878 ms	1354 ms
ANN SL	20.74 ms	31.67 ms	\	17.25 ms	\
ANN ML	40.56 ms	68.78 ms	75.39 ms	35.20 ms	54.73 ms

Table 14. Attack categories remained after CSE-CIC-IDS 2018 dataset balancing.

CSE-CIC-IDS 2018	
Attack Type	**Percent**
Bot	10%
DDoS attack HOIC	10%
DDoS attack LOIC HTTP	10%
DoS attack GoldenEye	10%
DoS attack Hulk	10%
DoS attack SlowHTTP Test	10%
DoS attack Slowloris	10%
FTP Brute Force	10%
Infiltration	10%
SSH Brute Force	10%
Total	**100%**

After applying the balancing procedure, the UNSW-NB15 dataset has undergone a reduction from nine to four attack categories. The categories *Analysis*, *Backdoors*, *Shellcode*, and *Worm* have been removed from the dataset. Similarly, in the CSE-CIC-IDS 2018 dataset, the number of attack categories has reduced from fourteen to ten. The categories *Brute Force Web*, *Brute Force XSS*, *DDoS attack LOIC UDP*, and *SQL Injection* have been removed. As a result, the remaining training datasets are perfectly balanced, ensuring an equal number of observations for each attack category.

6. Experimental Design and Results

In this section, we present the results of our analysis, where various configurations were tested for each of the proposed methods. The selected hyper-parameters were chosen based on their optimal performance in terms of classification accuracy and computational efficiency. For SVM, we employed the Linear Kernel function to separate the data. This choice was motivated by its simplicity, efficiency, and effectiveness in finding an optimal solution for data separation. By projecting the data into a higher-dimensional space, the linear kernel function makes it easier for the SVM to identify and separate data points, thus enhancing classification performance. In the case of Decision Trees (DT) and Random Forests (RF), we utilized Gini's diversity index as the criterion for generating the trees. Gini's index measures the diversity of samples used in creating a decision tree. It calculates the probability of misclassifying a randomly chosen sample when using the decision tree. By minimizing the Gini index, we increase the diversity of the samples and reduce the probability of incorrect classification. This makes Gini's diversity index a valuable tool for evaluating the performance of DT and RF algorithms in intrusion detection systems. KNN (K-Nearest Neighbors) was configured with K-neighbors set to 5, and we employed the Cosine Distance as the distance metric. The use of the cosine distance in KNN models for intrusion detection systems offers several advantages. The cosine distance measures the similarity between two vectors, making it a suitable metric for comparing the similarity between intrusion patterns. In the case of Linear Discriminant Analysis (LDA), we utilized the Pseudo-linear Discriminant type. This approach involves the use of the pseudo-inverse of the covariance matrix rather than the linear covariance matrix. By employing kernel functions to project the data into a higher-dimensional space, the separability of the data is enhanced. Additionally, the pseudo-linear approach is computationally efficient and capable of handling large datasets, making it ideal for intrusion detection systems that require real-time processing of substantial amounts of data.

Additionally, our approach allowed for the incorporation of non-linear relationships in the data, enhancing the accuracy and robustness of intrusion detection. Regarding the Artificial Neural Network (ANN) models, we employed different architectures for binary

and multi-class classification. In binary classification, we utilized two types of networks: ANN Single Layer (SL) and ANN Multi-layer (ML). The number of neurons in each layer was adjusted based on the dataset number of features. For the KDD 99 dataset, we used ANN SL with a single hidden layer consisting of 140 neurons. The ANN ML model comprised five hidden layers with 140, 280, 140, 60, and 30 neurons, respectively. Both ANNs employed the ReLU activation function, and the output layer contained 2 neurons with the Softmax activation function. In the UNSW-NB15 dataset, we implemented two approaches for binary classification. The first approach utilized ANN SL with a single hidden layer of 260 neurons and ReLU activation functions. The second approach involved a five-layer ANN with 260, 520, 260, 100, and 50 neurons in each respective layer. All hidden layers used ReLU activation functions, while the output layer consisted of 2 neurons with Softmax activation. For the CSE-CIC-ID 2018 dataset in binary classification, we employed two ANN models. The first model, ANN SL, had a single hidden layer with 90 neurons. The second model, ANN ML, comprised five hidden layers with 90, 180, 90, 45, and 20 neurons, respectively. ReLU activation functions were used in the hidden layers, and the Softmax activation function was applied to the output layer, which contained 2 neurons. In multi-class classification, we only utilized the ANN ML model. However, we observed that this model lacked the capability to accurately classify the data, so the performance results for multi-class classification are not reported. Tables 11 and 12 present a comparison of the binary and multi-class classification performance, respectively, highlighting the best results. The majority of the methods achieved high performance, with ANN, DT, and RF demonstrating the best overall performance. SVM, LDA, and KNN exhibited the worst results in both binary and multi-class cases. ANN (SL and ML) showed particularly good specificity, indicating their effectiveness in recognizing attacks. Computation times are reported in Table 13, calculated for 1000 observations. DT exhibited the shortest computation time, likely due to its lower complexity compared to other methods. However, all the methods showed relatively high computation times, underscoring the need for efficient algorithms in intrusion detection systems. The obtained results validate the applicability of the proposed machine learning algorithms in intrusion detection systems. It is important to note that binary classification generally outperformed multi-class classification, as binary classification achieved precise results with low false negatives and false positives. On the other hand, multi-class classification is more computationally demanding and complex, leading to less effective outcomes, as observed in this study. Figures 6 and 7 provide graphical representations of the results obtained from the classifiers in the binary and multi-class cases, respectively. The tests were conducted on an Acer AN515-54 laptop featuring an Intel(R) Core™ i7-9150H processor, running Windows 10 and MATLAB version R2021b.

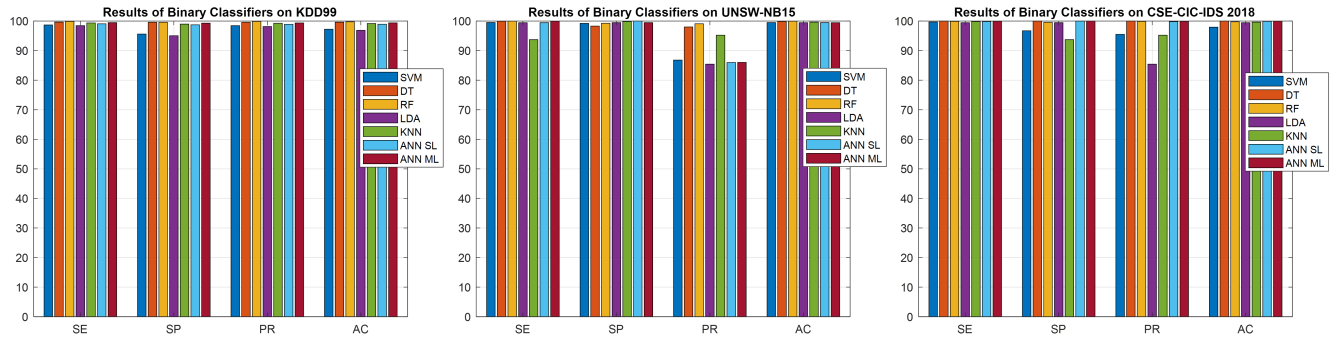

Figure 6. Graphical representation of binary classification results.

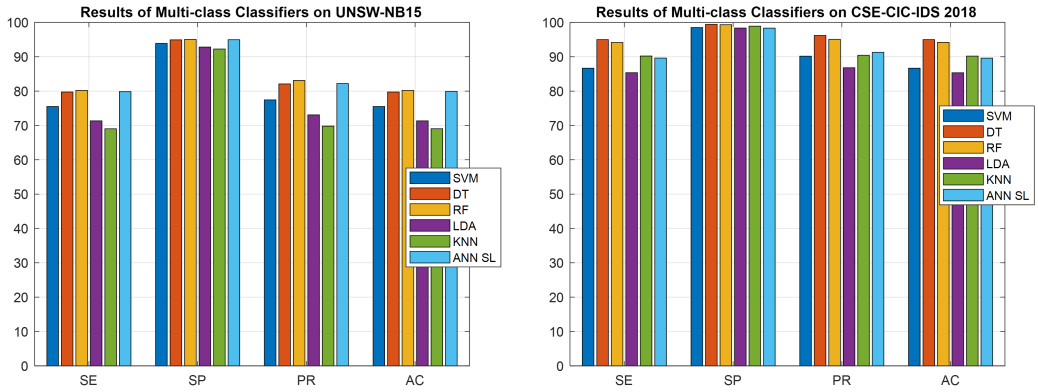

Figure 7. Graphical representation of the results obtained with multi-class classifiers, in the UNSW-NB15 and CSE-CIC-IDS 2018 datest cases.

7. Final Discussion

The introduction of machine learning models and feature selection techniques has revolutionized data analysis and opened up new perspectives for solving complex classification and regression problems. In this discussion, we will look at some of the most popular machine learning models, such as Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN) and Artificial Neural Network (AN). We will also explore some dataset manipulation techniques, such as variable encoding, data scaling, and dataset balancing. Lets start by analyzing machine learning models. The SVM is a classification model that tries to find an optimal hyperplane to separate data into different classes. It has demonstrated good performance in many applications, but can be computationally expensive and require careful parameter selection. Decision Trees (DT) are intuitive models that represent a set of decision rules organized in a tree structure. They are easy to interpret and can handle both numerical and categorical data. However, DTs can suffer from over-fitting if their complexity is not controlled. Random Forests (RF) are a combination of many decision trees, working together to make a prediction. They are robust against over-fitting and can handle a large number of features without requiring pre-selection. However, they can require more computational resources than other models. Linear Discriminant Analysis (LDA) is a classification technique that seeks to maximize the separation between classes through a linear transformation of the data. LDA works best when the data is linearly separable, but can struggle with complex or nonlinear datasets. K-Nearest Neighbors (KNN) is a model that ranks a spot based on the labels of its closest neighbors. KNN is simple to implement and suitable for datasets with a non-linear structure. However, its performance can be affected by the choice of number of neighbors and the distance used. The Artificial Neural Network (AN) is a neural network model with a single unit of output that is trained to approximate a linear function. It can be used for classification and regression problems. AN requires the appropriate selection and fitting of activation functions and can be susceptible to convergence problems. Moving on to dataset manipulation techniques, variable encoding is a process that converts categorical variables into a numeric representation. This allows machine learning models to use those variables in their analyses. However, inappropriate encoding could introduce an illusion of order or relationship between categories. Data scaling is a common practice to bring all features to the same scale. This can improve the convergence and performance of some machine learning models. However, you may need to be careful if your data contains outliers, as they may affect the final result. Dataset balancing is often necessary when target classes are unbalanced. This involves adjusting the number of instances of each class or assigning different weights to the classes in the training process. The balance may improve the model's ability to learn from minority classes, but could also introduce a risk of over-fitting. The use of machine learning models and feature selection techniques offers a wide range

of options for data analysis. Each model and technique has its strengths and weaknesses, which need to be carefully considered based on the specific needs of the problem and dataset. Choosing your model and techniques carefully can lead to meaningful results and a better understanding of the underlying data. Table 15 reports a fast summary on ML models and their performance in the context of IDS, highlighting the most relevant article related to the better model for each different dataset analyzed in this work.

Table 15. Summary of most relevant works.

Dataset	"Best" ML Models	Dataset Management	Performance	Most Relevant References
KDD99 (only binary)	RF, DT and KNN	DT and KNN perform better with preliminary feature transformation, reduction and scaling. RF require less preliminary dataset manipulation.	Comparable performance with very high accuracy (>99%) and very low FPR (<0.5%)	[23,38,41]
UNSW-NB15 (both binary and multiclass)	RF and ANN	RF require feature reduction. ANN perform also without preliminary manipulation.	Comparable performance with very high accuracy and very low FPR in binary classification. RF perform better for multiclass IDS with medium accuracy (<85%) and low FPR (<1%)	[30,31]
CSE-CIC-IDS 2018 (both binary and multiclass)	DT, RF and KNN	Required complete dataset manipulation Workflow.	Comparable performance with very high accuracy and very low FPR in binary classification. RF perform better for multiclass IDS with medium-high accuracy (>90%) and low FPR (<1%)	[19,41]

8. Conclusions and Future Work

This paper demonstrates the excellent performance achieved by machine learning (ML) models in detecting attacks, specifically in the domains of Anomaly Detection (binary classification) and Anomaly Classification (multi-class problems). To provide a comprehensive and relevant analysis, the study compares the performance of these models using three widely used datasets in the literature. The paper presents the entire operational flow of dataset management and manipulation, highlighting the techniques employed to optimize model performance. It discusses the advantages and disadvantages of different approaches to dataset manipulation, feature selection, and reduction. The evaluation of classification metrics commonly used in such problems has been performed, aiming to provide a comprehensive comparison of the models' performance. The results indicate that Decision Trees (DT) and Random Forests (RF) consistently outperform other models across all selected datasets, for both binary and multi-class classification. Moreover, the paper includes an analysis of the computational complexity, specifically computation time, of the models. It demonstrates that DT exhibits the lowest computational burden among the models evaluated. The scope of this research extends to the evaluation of ML models for embedded data traffic safety applications in mechatronic systems, such as fully electrified vehicles, complex industrial automation systems, and industrial robotic systems. Additionally, the study explores the potential of deep learning models, highlighting their effectiveness in various tasks. Integrating deep learning models into intrusion detection systems has the potential to improve accuracy and handle complex data, thereby reducing false alarms, enhancing detection rates, and enabling real-time monitoring for anomaly

detection—particularly crucial for high-traffic networks. The main objective of this review article is to examine the state of the art of machine learning (ML) in the area of computer network security, but it is important to emphasize that future developments in this area will increasingly focus on applying deep learning (DL). DL offers enormous potential to address complex and unknown challenges in detecting cyber threats and improving network security. Here are some reasons why DL will be a key pillar for future development in computer network security. One of the main challenges in the field of network security is the processing of large amounts of data from different sources. The DL, thanks to its deep learning capabilities, can automatically extract high-level and complex features from the raw data. This helps identify patterns and hidden correlations that could be indicative of malicious behavior or suspicious activity within computer networks. Cyberattacks are becoming more sophisticated, using advanced techniques such as signature evasion and camouflage of malicious activity. DL can address this challenge by allowing models to automatically learn new attack patterns without the need for explicit rules. DL models can recognize patterns of suspicious behavior even when attacks take never-before-seen forms, making network defenses more robust and adaptable. Network security requires analyzing massive amounts of data from various traffic streams, system logs, and other sources. DL excels at processing large datasets, enabling rapid and simultaneous analysis of multiple data sources. This can lead to better detection of threats and faster processing of information, allowing you to respond quickly to cyber attacks. DL can play a crucial role in detecting anomalous activity in computer networks. For example, DL models can learn the normal behaviors of network devices and individual users, allowing you to detect deviant behavior that could indicate malicious activity. This ability to spot anomalies and unusual behavior helps identify intrusions before they cause significant damage. DL also offers opportunities to improve the performance of existing machine learning models. Deep neural networks can be used to further refine existing ML models, enabling more accurate predictions and reducing the number of false positives and false negatives in threat detections. While this article focuses on machine learning, it is clear that future developments in computer network security will increasingly require the use of deep learning. Extracting complex features, detecting sophisticated attacks, analyzing large amounts of data, detecting anomalous activity and improving model performance are just some of the potential benefits offered by the DL to improve the security of computer networks. The integration of DL into the network security space promises to provide a more robust and resilient defense against increasingly complex and sophisticated cyberthreats [61–68]. Future developments of this research focus on the application of ML/AI models in industrial mechatronics. Examples include ensuring the safety of on-board control systems in assisted and autonomous driving vehicles, and developing advanced anomaly detection systems for information traffic in industrial automation systems, such as drive control systems for manipulator robots or trajectory planning systems for robots [69–82].

Author Contributions: Authors has equally contributed at this article. All authors have read and agreed to the published version of the manuscript.

Funding: This research is partially funded by the Horizon Europe program under grant agreement 101092850 (AERO project) and by the European High-Performance Computing Joint Undertaking (JU) program under grant agreement 101033975 (EUPEX).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Musa, U.S.; Chhabra, M.; Ali, A.; Kaur, M. Intrusion Detection System using Machine Learning Techniques: A Review. In Proceedings of the 2020 International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 10–12 September 2020; pp. 149–155.
2. Aljabri, M.; Altamimi, H.S.; Albelali, S.A.; Maimunah, A.H.; Alhuraib, H.T.; Alotaibi, N.K.; Alahmadi, A.A.; Alhaidari, F.; Mohammad, R.M.A.; Salah, K. Detecting malicious URLs using machine learning techniques: Review and research directions. *IEEE Access* **2022**, *10*, 121395–121417. [CrossRef]
3. Okey, O.D.; Maidin, S.S.; Adasme, P.; Lopes Rosa, R.; Saadi, M.; Carrillo Melgarejo, D.; Zegarra Rodríguez, D. BoostedEnML: Efficient technique for detecting cyberattacks in IoT systems using boosted ensemble machine learning. *Sensors* **2022**, *22*, 7409. [CrossRef] [PubMed]
4. Htun, H.H.; Biehl, M.; Petkov, N. Survey of feature selection and extraction techniques for stock market prediction. *Financ. Innov.* **2023**, *9*, 26. [CrossRef] [PubMed]
5. Bhuyan, M.H.; Bhattacharyya, D.K.; Kalita, J.K. *Network Traffic Anomaly Detection and Prevention: Concepts, Techniques, and Tools*; Springer: Berlin/Heidelberg, Germany, 2017.
6. Liu, J.; Dong, Y.; Zha, L.; Tian, E.; Xie, X. Event-based security tracking control for networked control systems against stochastic cyber-attacks. *Inf. Sci.* **2022**, *612*, 306–321. [CrossRef]
7. Zha, L.; Liao, R.; Liu, J.; Xie, X.; Tian, E.; Cao, J. Dynamic event-triggered output feedback control for networked systems subject to multiple cyber attacks. *IEEE Trans. Cybern.* **2021**, *52*, 13800–13808. [CrossRef] [PubMed]
8. Qu, F.; Tian, E.; Zhao, X. Chance-Constrained H-infinity State Estimation for Recursive Neural Networks Under Deception Attacks and Energy Constraints: The Finite-Horizon Case. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**. [CrossRef]
9. Chen, H.; Jiang, B.; Ding, S.X.; Huang, B. Data-driven fault diagnosis for traction systems in high-speed trains: A survey, challenges, and perspectives. *IEEE Trans. Intell. Transp. Syst.* **2020**, *23*, 1700–1716. [CrossRef]
10. Elhanashi, A.; Lowe Sr, D.; Saponara, S.; Moshfeghi, Y. Deep learning techniques to identify and classify COVID-19 abnormalities on chest X-ray images. In *Proceedings of the Real-Time Image Processing and Deep Learning 2022*; SPIE: Bellingham, WA, USA, 2022; Volume 12102, pp. 15–24.
11. Zheng, Q.; Zhao, P.; Wang, H.; Elhanashi, A.; Saponara, S. Fine-grained modulation classification using multi-scale radio transformer with dual-channel representation. *IEEE Commun. Lett.* **2022**, *26*, 1298–1302. [CrossRef]
12. Elhanashi, A.; Gasmi, K.; Begni, A.; Dini, P.; Zheng, Q.; Saponara, S. Machine Learning Techniques for Anomaly-Based Detection System on CSE-CIC-IDS2018 Dataset. In *Applications in Electronics Pervading Industry, Environment and Society: APPLEPIES 2022*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 131–140.
13. Pisner, D.A.; Schnyer, D.M. Support vector machine. In *Machine Learning*; Elsevier: Amsterdam, The Netherlands, 2020; pp. 101–121.
14. Widodo, A.; Yang, B.S. Support vector machine in machine condition monitoring and fault diagnosis. *Mech. Syst. Signal Process.* **2007**, *21*, 2560–2574. [CrossRef]
15. Pervez, M.S.; Farid, D.M. Feature selection and intrusion classification in NSL-KDD cup 99 dataset employing SVMs. In Proceedings of the 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014), Dhaka, Bangladesh, 18–20 December 2014; pp. 1–6.
16. Al Mehedi Hasan, M.; Nasser, M.; Pal, B. On the KDD'99 dataset: Support vector machine based intrusion detection system (ids) with different kernels. *Int. J. Electron. Commun. Comput. Eng* **2013**, *4*, 1164–1170.
17. Jing, D.; Chen, H.B. SVM based network intrusion detection for the UNSW-NB15 dataset. In Proceedings of the 2019 IEEE 13th international conference on ASIC (ASICON), Chongqing, China, 29 October–1 November 2019; pp. 1–4.
18. Kasongo, S.M.; Sun, Y. Performance analysis of intrusion detection systems using a feature selection method on the UNSW-NB15 dataset. *J. Big Data* **2020**, *7*, 1–20. [CrossRef]
19. Kanimozhi, V.; Jacob, T.P. Calibration of various optimized machine learning classifiers in network intrusion detection system on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing. *Int. J. Eng. Appl. Sci. Technol.* **2019**, *4*, 209–213. [CrossRef]
20. Liu, L.; Wang, P.; Lin, J.; Liu, L. Intrusion detection of imbalanced network traffic based on machine learning and deep learning. *IEEE Access* **2020**, *9*, 7550–7563. [CrossRef]
21. Raj, A. An Exhaustive Guide to Decision Tree Classification in Python 3.x. 2021. Available online: https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f (accessed on 30 January 2023).
22. Patel Brijain, R.; Rana, K.K. A Survey on Decision Tree Algorithm for Classification. *Int. J. Eng. Dev. Res.* **2014**, *2*, 1–5.
23. Lee, J.H.; Lee, J.H.; Sohn, S.G.; Ryu, J.H.; Chung, T.M. Effective value of decision tree with KDD 99 intrusion detection datasets for intrusion detection system. In Proceedings of the 2008 10th International Conference on Advanced Communication Technology, Gangwon, Republic of Korea, 17–20 February 2008; Volume 2, pp. 1170–1175.
24. Amor, N.B.; Benferhat, S.; Elouedi, Z. Naive bayes vs decision trees in intrusion detection systems. In Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 14–17 March 2004; pp. 420–424.
25. Bagui, S.; Kalaimannan, E.; Bagui, S.; Nandi, D.; Pinto, A. Using machine learning techniques to identify rare cyber-attacks on the UNSW-NB15 dataset. *Secur. Priv.* **2019**, *2*, e91.
26. Zuech, R.; Hancock, J.; Khoshgoftaar, T.M. Detecting web attacks using random undersampling and ensemble learners. *J. Big Data* **2021**, *8*, 75. [CrossRef]

27. Education, I.C. Random Forest. 2020. Available online: https://www.ibm.com/cloud/learn/random-forest (accessed on 30 January 2023).
28. Hasan, M.A.M.; Nasser, M.; Ahmad, S.; Molla, K.I. Feature Selection for Intrusion Detection Using Random Forest. *J. Inf. Secur.* **2016**, *7*, 129–140. [CrossRef]
29. Pal, M.M.B.; Ahmad, S. Support Vector Machine and Random Forest Modeling for Intrusion Detection System (IDS). *J. Intell. Learn. Syst. Appl.* **2014**, *6*, 42869.
30. Hassine, K.; Erbad, A.; Hamila, R. Important complexity reduction of random forest in multi-classification problem. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 226–231.
31. Primartha, R.; Tama, B.A. Anomaly detection using random forest: A performance revisited. In Proceedings of the 2017 International Conference on Data and Software Engineering (ICoDSE), Palembang, Indonesia, 1–2 November 2017; pp. 1–6.
32. Mishra, S.; Datta-Gupta, A. Chapter 5—Multivariate Data Analysis. In *Applied Statistical Modeling and Data Analytics*; Mishra, S., Datta-Gupta, A., Eds.; Elsevier: Amsterdam, The Netherlands, 2018; pp. 97–118. [CrossRef]
33. Adams, M. CHEMOMETRICS AND STATISTICS | Multivariate Classification Techniques. In *Encyclopedia of Analytical Science*, 2nd ed.; Worsfold, P., Townshend, A., Poole, C., Eds.; Elsevier: Oxford, UK, 2005; pp. 21–27. [CrossRef]
34. Sathya, S.S.; Ramani, R.G.; Sivaselvi, K. Discriminant analysis based feature selection in kdd intrusion dataset. *Int. J. Comput. Appl.* **2011**, *31*, 1–7.
35. Katos, V. Network intrusion detection: Evaluating cluster, discriminant, and logit analysis. *Inf. Sci.* **2007**, *177*, 3060–3073. [CrossRef]
36. Solani, S.; Jadav, N.K. A Novel Approach to Reduce False-Negative Alarm Rate in Network-Based Intrusion Detection System Using Linear Discriminant Analysis. In *Inventive Communication and Computational Technologies*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 911–921.
37. Karatas, G.; Demir, O.; Sahingoz, O.K. Increasing the performance of machine learning-based IDSs on an imbalanced and up-to-date dataset. *IEEE Access* **2020**, *8*, 32150–32162. [CrossRef]
38. Benaddi, H.; Ibrahimi, K.; Benslimane, A. Improving the Intrusion Detection System for NSL-KDD Dataset based on PCA-Fuzzy Clustering-KNN. In Proceedings of the 2018 6th International Conference on Wireless Networks and Mobile Communications (WINCOM), Marrakesh, Morocco, 16–19 October 2018; pp. 1–6. [CrossRef]
39. Kuang, L.; Zulkernine, M. An anomaly intrusion detection method using the CSI-KNN algorithm. In Proceedings of the 2008 ACM Symposium on Applied Computing, Ceara, Brazil, 16–20 March 2008; pp. 921–926.
40. Kocher, G.; Kumar, G. Performance Analysis of Machine Learning Classifiers for Intrusion Detection Using Unsw-Nb15 Dataset. *Comput. Sci. Inf. Technol. (CSIT)* **2020**, *10*, 31–40.
41. Dini, P.; Saponara, S. Analysis, design, and comparison of machine-learning techniques for networking intrusion detection. *Designs* **2021**, *5*, 9. [CrossRef]
42. Leevy, J.L.; Khoshgoftaar, T.M. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. *J. Big Data* **2020**, *7*, 1–19. [CrossRef]
43. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
44. Al-Janabi, S.T.F.; Saeed, H.A. A Neural Network Based Anomaly Intrusion Detection System. In Proceedings of the 2011 Developments in E-Systems Engineering, Dubai, United Arab Emirates, 6–8 December 2011; pp. 221–226. [CrossRef]
45. Jia, Y.; Wang, M.; Wang, Y. Network intrusion detection algorithm based on deep neural network. *IET Inf. Secur.* **2019**, *13*, 48–53. [CrossRef]
46. Hanif, S.; Ilyas, T.; Zeeshan, M. Intrusion Detection In IoT Using Artificial Neural Networks On UNSW-15 Dataset. In Proceedings of the 2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT and AI (HONET-ICT), Charlotte, NC, USA, 6–9 October 2019; pp. 152–156. [CrossRef]
47. Rajagopal, S.; Hareesha, K.S.; Kundapur, P.P. Feature relevance analysis and feature reduction of UNSW NB-15 using neural networks on MAMLS. In *Advanced Computing and Intelligent Engineering*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 321–332.
48. Kim, J.; Shin, Y.; Choi, E. An intrusion detection model based on a convolutional neural network. *J. Multimed. Inf. Syst.* **2019**, *6*, 165–172. [CrossRef]
49. Kanimozhi, V.; Jacob, T.P. Artificial Intelligence based Network Intrusion Detection with Hyper-Parameter Optimization Tuning on the Realistic Cyber Dataset CSE-CIC-IDS2018 using Cloud Computing. In Proceedings of the 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 4–6 April 2019; pp. 33–36. [CrossRef]
50. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 10–12 November 2015; pp. 1–6.
51. University of New Brunswick, Canadian Institute for Cybersecurity. CSE-CIC-IDS2018 on AWS. 2021. Available online: https://www.unb.ca/cic/datasets/ids-2018.html (accessed on 30 January 2023).
52. Grandini, M.; Bagli, E.; Visani, G. Metrics for multi-class classification: An overview. *arXiv* **2020**, arXiv:2008.05756.
53. Dini, P.; Begni, A.; Ciavarella, S.; De Paoli, E.; Fiorelli, G.; Silvestro, C.; Saponara, S. Design and Testing Novel One-Class Classifier Based on Polynomial Interpolation With Application to Networking Security. *IEEE Access* **2022**, *10*, 67910–67924. [CrossRef]

54. Scikit-Learn Developers. Metrics and Scoring: Quantifying the Quality of Predictions. 2022. Available online: https://scikit-learn.org/stable/modules/model_evaluation.html#metrics-and-scoring-quantifying-the-quality-of-predictions (accessed on 30 January 2023).
55. Devarakonda, A.; Sharma, N.; Saha, P.; Ramya, S. Network intrusion detection: A comparative study of four classifiers using the NSL-KDD and KDD'99 datasets. In *Journal of Physics: Conference Series*; IOP Publishing: Bristol, UK, 2022; Volume 2161, p. 012043.
56. Jie, L.; Jiahao, C.; Xueqin, Z.; Yue, Z.; Jiajun, L. One-hot encoding and convolutional neural network based anomaly detection. *J. Tsinghua Univ. Sci. Technol.* **2019**, *59*, 523–529.
57. Moualla, S.; Khorzom, K.; Jafar, A. Improving the performance of machine learning-based network intrusion detection systems on the UNSW-NB15 dataset. *Comput. Intell. Neurosci.* **2021**, *2021*, 5557577. [CrossRef]
58. Roy, A.; Singh, K.J. Multi-classification of UNSW-NB15 dataset for network anomaly detection system. In *Proceedings of the International Conference on Communication and Computational Technologies*; Springer: Singapore, 2021; pp. 429–451.
59. Kannari, P.R.; Shariff, N.C.; Biradar, R.L. Network intrusion detection using sparse autoencoder with swish-PReLU activation model. *J. Ambient. Intell. Humaniz. Comput.* **2021**, 1–13. [CrossRef]
60. Brownlee, J. A gentle introduction to imbalanced classification. *Mach. Learn. Mastery* **2019**, *22* . Available online: https://machinelearningmastery.com/what-is-imbalanced-classification/ (accessed on 30 January 2023).
61. Lopez-Martin, M.; Sanchez-Esguevillas, A.; Arribas, J.I.; Carro, B. Contrastive Learning Over Random Fourier Features for IoT Network Intrusion Detection. *IEEE Internet Things J.* **2023**, *10*, 8505–8513. [CrossRef]
62. Lopez-Martin, M.; Sanchez-Esguevillas, A.; Arribas, J.I.; Carro, B. Network Intrusion Detection Based on Extended RBF Neural Network With Offline Reinforcement Learning. *IEEE Access* **2021**, *9*, 153153–153170. [CrossRef]
63. Lopez-Martin, M.; Sanchez-Esguevillas, A.; Arribas, J.I.; Carro, B. Supervised contrastive learning over prototype-label embeddings for network intrusion detection. *Inf. Fusion* **2022**, *79*, 200–228. [CrossRef]
64. Lopez-Martin, M.; Carro, B.; Arribas, J.I.; Sanchez-Esguevillas, A. Network intrusion detection with a novel hierarchy of distances between embeddings of hash IP addresses. *Knowl.-Based Syst.* **2021**, *219*, 106887. [CrossRef]
65. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A. Application of deep reinforcement learning to intrusion detection for supervised problems. *Expert Syst. Appl.* **2020**, *141*, 112963. [CrossRef]
66. Caminero, G.; Lopez-Martin, M.; Carro, B. Adversarial environment reinforcement learning algorithm for intrusion detection. *Comput. Netw.* **2019**, *159*, 96–109. [CrossRef]
67. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A. Variational data generative model for intrusion detection. *Knowl. Inf. Syst.* **2019**, *60*, 569–590. [CrossRef]
68. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A.; Lloret, J. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot. *Sensors* **2017**, *17*, 1967. [CrossRef]
69. Benedetti, D.; Agnelli, J.; Gagliardi, A.; Dini, P.; Saponara, S. Design of a digital dashboard on low-cost embedded platform in a fully electric vehicle. In Proceedings of the 2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe), Madrid, Spain, 9–12 June 2020; pp. 1–5.
70. Dini, P.; Saponara, S. Processor-in-the-loop validation of a gradient descent-based model predictive control for assisted driving and obstacles avoidance applications. *IEEE Access* **2022**, *10*, 67958–67975. [CrossRef]
71. Dini, P.; Saponara, S. Model-Based Design of an Improved Electric Drive Controller for High-Precision Applications Based on Feedback Linearization Technique. *Electronics* **2021**, *10*, 2954. [CrossRef]
72. Cosimi, F.; Dini, P.; Giannetti, S.; Petrelli, M.; Saponara, S. Analysis and design of a non-linear MPC algorithm for vehicle trajectory tracking and obstacle avoidance. In *Proceedings of the Applications in Electronics Pervading Industry, Environment and Society: APPLEPIES 2020 8*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 229–234.
73. Bernardeschi, C.; Dini, P.; Domenici, A.; Mouhagir, A.; Palmieri, M.; Saponara, S.; Sassolas, T.; Zaourar, L. Co-simulation of a Model Predictive Control System for Automotive Applications. In *Software Engineering and Formal Methods, Proceedings of the SEFM 2021 Collocated Workshops: CIFMA, CoSim-CPS, OpenCERT, ASYDE, Virtual Event, 6–10 December 2021*; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2022; pp. 204–220.
74. Begni, A.; Dini, P.; Saponara, S. Design and Test of an LSTM-Based Algorithm for Li-Ion Batteries Remaining Useful Life Estimation. In *Applications in Electronics Pervading Industry, Environment and Society: APPLEPIES 2022*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 373–379.
75. Bernardeschi, C.; Dini, P.; Domenici, A.; Palmieri, M.; Saponara, S. Do-it-Yourself FMU Generation. In *Software Engineering and Formal Methods, Proceedings of the SEFM 2022 Collocated Workshops: AI4EA, F-IDE, CoSim-CPS, CIFMA, Berlin, Germany, 26–30 September 2022*; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2023; pp. 210–227.
76. Dini, P.; Saponara, S. Cogging torque reduction in brushless motors by a nonlinear control technique. *Energies* **2019**, *12*, 2224. [CrossRef]
77. Dini, P.; Saponara, S. Electro-thermal model-based design of bidirectional on-board chargers in hybrid and full electric vehicles. *Electronics* **2022**, *11*, 112. [CrossRef]
78. Dini, P.; Saponara, S. Design of adaptive controller exploiting learning concepts applied to a BLDC-based drive system. *Energies* **2020**, *13*, 2512. [CrossRef]
79. Dini, P.; Saponara, S. Design of an observer-based architecture and non-linear control algorithm for cogging torque reduction in synchronous motors. *Energies* **2020**, *13*, 2077. [CrossRef]

80. Benedetti, D.; Agnelli, J.; Gagliardi, A.; Dini, P.; Saponara, S. Design of an Off-Grid Photovoltaic Carport for a Full Electric Vehicle Recharging. In Proceedings of the 2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe), Madrid, Spain, 9–12 June 2020; pp. 1–6. [CrossRef]
81. Bernardeschi, C.; Dini, P.; Domenici, A.; Palmieri, M.; Saponara, S. Formal verification and co-simulation in the design of a synchronous motor control algorithm. *Energies* **2020**, *13*, 4057. [CrossRef]
82. Dini, P.; Ariaudo, G.; Botto, G.; Greca, F.L.; Saponara, S. Real-time electro-thermal modelling & predictive control design of resonant power converter in full electric vehicle applications. *IET Power Electron.* **2023**. [CrossRef]