American University of Sharjah

College of Engineering

Department of Computer Science & Engineering

CMP 466 – Machine Learning and Data Mining

Spring 2024

Team Project – Final Report

Due: May 11th, 2024

Instructor: Dr. Salam Dhou

Team Members:

Ahmad Mansour - b00088651

Amr Abu Alhaj - b00088683

Baraa Abed - b00088000

Abstract

The rapid expansion of Internet of Things (IoT) systems in both home and industrial applications has significantly improved technological capabilities and operational efficiency. However, this widespread adoption also presents security challenges due to the continuous network connectivity required by IoT devices. Using machine learning (ML) algorithms for the analysis of IoT traffic data emerges as a powerful strategy to enhance the security of IoT systems. By identifying and classifying potential attacks through the analysis of captured network traffic, ML models can effectively operate as intrusion detection systems (IDS) for the network. This project focuses on experimenting with various ML techniques, including decision trees (DT), k-nearest neighbors (KNN), Naïve Bayes (NB), and support vector machines (SVM) on the RT-IoT2022 IoT network dataset. We detail the dataset's composition, our data preprocessing methodologies, and the theoretical concepts behind each ML model utilized. The paper further delves into feature selection, modeling method used, and model selection processes. Evaluation metrics such as accuracy, precision, recall, and F1 score are used to assess model performance, and to compare our results with that found in the literature. The best result achieved was for balanced KNN fitted with 75% of features. The model achieved 0.9887 across its accuracy, precision, recall and F1-score as well as achieving AUC of 0.99370.

Introduction

Internet of things (IoT) systems are becoming increasingly popular due to the rapid development of technology, with their use continuously expanding both in home and industrial applications. However, the ever-increasing usage of IoT also introduces significant security risks, which more and more attackers have come to exploit. One main reason for the increased targeting is that IoT devices require continuous network connectivity, opening the door to cybercriminals who can exploit IoT devices and even other devices within the same network. Moreover, due to the widespread use of IoT (especially in the industrial setting), cyber-attacks on IoT systems can have devastating and costly consequences. Additionally, classical rule-based systems are not able to stay up to date with the latest threats. To combat this, machine learning (ML) algorithms can be used to analyze captured IoT traffic data to identify and classify different possible attacks. ML models achieve this by leveraging their capability to learn patterns and detect anomalies within the data. This approach not only enhances the overall security of IoT systems but also reduces the risk of large-scale vulnerabilities being exploited, thereby ensuring safer and more reliable operations.

In this paper, we will experiment with various ML models on the RT-IoT2022 dataset [1], which is a compilation of network traffic information from different IoT devices across different traffic scenarios. In the rest of the paper, we begin with a review of relevant papers in the literature. We then provide an in-depth description of the RT-IoT2022 dataset. Next, definitions of preprocessing methods, ML models, modelling methods, and evaluation metrics that we would be using in our experiments are provided. Finally, results of the experiments will be presented and discussed. As of the date of writing this paper, there was only one paper running ML models on the RT-IoT2022 dataset. We hope our study will broaden the understanding of the RT-IoT2022 dataset's potential for ML applications while also contributing to enhanced security and efficiency in IoT network security.

Literature Review

In the field of intrusion detection, many researchers attempted to create datasets for intrusion detection systems (IDS), particularly in order to deploy Artificial Intelligence (AI) based IDS. One such study is the one conducted by B. S. Sharmila and R. Nagapadma [2]. They examine significant benchmark datasets in the IDS framework evaluation, mentioning several key datasets like DARPA 1999 [3], CICIDS2017 [4], and UNSW-NB15 [5], BoT-IoT [6], and their specific features and limitations. It explores IDS solutions, including existing deep learning approaches such as Convolutional Neural Networks (CNN) and autoencoders, highlighting their computational demands. A novel solution, the quantized autoencoder (QAE), is proposed for anomaly detection within resource-constrained environments. This paper also examines feature extraction capabilities of autoencoders as a benchmark for comparing QAE's performance. The findings indicate that QAE offers significant improvements in efficiency regarding memory size and CPU utilization, a fact underscored by its performance on a Raspberry Pi. Furthermore, the research suggests that the dataset generated could address issues present in existing benchmark IDS datasets and mitigate the scarcity of IDS datasets currently in use.

T. Saba et al. [7] also used CNNs to develop an IoT IDS. They use two IoT network intrusion datasets, namely the aforementioned BoT-IoT dataset [6] and the Network intrusion detection (NID) dataset [8]. The model is evaluated using confusion matrices and the training and testing accuracy for each dataset. They achieved an accuracy of 95.55% for the BoT-IoT dataset and 99.51% for the NID dataset. They conclude that although their results prove the effectiveness of their model, the model could be improved by further hyperparameter tuning. Unfortunately, their tests on the Bot-IoT dataset suffer from data imbalance, reducing the authenticity of their accuracy for this dataset.

On the other hand, [9] proposes an IDS that uses autoencoders to detect anomalies caused by Distributed Denial of Service attacks in Industrial Control Systems. The approach this paper proposed aims to have a high detection rate with low false alarms. Their proposed architecture involves autoencoders that use network flow data as its training data. The proposed architecture was then evaluated based on its performance in detecting anomalies in a real industry plant. Findings suggest that the deep autoencoder showed the ability to detect anomalies effectively, outperforming state of the art as well as baseline models in an unsupervised learning

setting having achieved an AUC of 0.962 for ping flooding, 0.989 for TCP SYN flooding, and 0.990 for MODBUS query flooding. Moreover, it was also able to detect abnormal behavior in legitimate devices even after the attack took place, showcasing its potential to be utilized in more sensitive areas or fields. However, the paper could benefit from investigating techniques in order to better select features when training the model in order to achieve a better model performance.

B. S. Sharmila and R. Nagapadma were not the only ones to investigate the generation of their own datasets. In [10], the focus shifts to the detection of MQTT-based attacks with IoT frameworks. This paper emphasizes the gap in specialized IDS datasets for IoT, especially for the MQTT protocol. The methodology of this paper involves the assessment of six ML techniques across three features: packet-based, unidirectional flow, and bidirectional flow. This was possible by using a simulated MQTT dataset that contains both benign and attack scenarios. This dataset, generated through simulation and includes normal operations along with four attack scenarios such as aggressive scans and MQTT brute-force attacks. The evaluation of ML models is conducted through performance metrics like overall accuracy, and F1-score when distinguishing between attack and benign instances. The findings state the superior performance of flow-based features over packet-based ones in identifying MQTT-based attacks, therefore emphasizing the importance of feature level selection in IoT IDS for protocols like MQTT, where attack patterns closely resemble benign traffic. Despite the significant contributions, including the introduction of a novel MQTT dataset and the exploration of various ML techniques, the study's reliance on simulated data, which might not fully capture the intricacies of real-world IoT environments.

Other key datasets are explored in the literature. In their paper, M. Bhavsar et al. [11] explores various machine learning models for anomaly-based intrusion detection system in connected autonomous vehicle's communication. The authors highlight the importance of such a system as more and more companies invest in developing autonomous cars. In their testing, they used the NSL-KDD dataset [12], an improved version of the KDD99 dataset [13], with 42 attributes and a total of 148,517 records. The KDD99 dataset is considered the standard dataset for intrusion detection and includes data that describes various traffic types. The authors used the Pearson correlation method for feature engineering, and ran the logistic regression (LR), support vector machine (SVM), K-nearest neighbor (KNN), classification and regression tree (CART), and linear discriminant analysis (LDA) models to classify the data both in a binary and multiclass fashion. They achieved 85%+ accuracy for test and train data in both binary and multiclass classification, with CART and KNN-based models performing best with 94% overall accuracy. Their precision and recall were around 0.9 while their false alarm rate was around 0.1. Despite the training and testing of multiple different models, it would have been better if the precision metrics were calculated for all the models used. Additionally, there was no clear indication of how the dataset and models used are purposed for connected autonomous vehicles.

Extending on their previous work, M. Bhavsar et al. [14] argues that traditional signature based IDS is no longer effective for combating the latest attacks and security threats. In extension to the NSL-KDD dataset, the authors used the previously mentioned CICIDS2017 [4] dataset and the IOTID20 [15] dataset which is emerging as popular datasets in the field, adding 6 more attack records and 2 million+ data points to the previous dataset. As for the model, the authors went for a deep learning approach, employing a Pearson correlation coefficient CNN (PCC-CNN) architecture. To train the model, a total of 122 and 128 features were used for binary and multiclass classification respectively. They achieved over 98% accuracy, nearly perfect precision, and recall, and an F1 score above 0.95, with a low false alarm rate of 0.01 for binary classification. In multiclass classification results dipped, with 0.95 accuracy, 0.85 precision, 0.78 recall, 0.76 F1 score, and the same low false alarm rate. More work is needed to improve the performance for multiclass classification. Additionally, using machine learning for its comparatively light computational needs may be advantageous.

H. Xu et al. [16] argues that, especially when considering multiclass classification, most anomaly-based IDS are limited due to their lack of performance and classification accuracy. To help with this, they employ the use of multiple preprocessing steps on the KDD99 [13] dataset. First, they use the synthetic minority oversampling technique (SMOTE) to balance the datasheet. This technique achieves balancing through the random oversampling of less frequent classes while also randomly under sampling the more frequent classes.

Next, the researchers compute the mutual information of each feature with respect to the class label. The mutual information is used here to determine the dependence of the class label on each of the features. From their calculations, they found that 5 features were not related to the classification of attacks, and thus were dropped from the dataset. As for the model, the researchers utilized Auto-ML to obtain the best model from a bagged ensemble of decision trees. Auto-ML helps with finding the best model with the best hyperparameters by going through multiple iterations until the best results are achieved. To evaluate the resulting model, the researchers used the accuracy, normalized mutual index (NMI), F1 score, and Kappa metrics. Their model achieved 0.9960 training accuracy, 0.9970 testing accuracy, 81.4615 NMI, 0.9904 F1-Score, and 0.9966 Kappa metric. Overall, the paper presents a solid process and great results, and it would be even more beneficial if newer datasets or real-world data were used to further validate the results obtained.

M. Bhavsar et al. were not the only researchers to run models on the NSL-KDD dataset. M. Almiani et al. [17] uses the dataset to develop and train a deep recurrent neural network (RNN) for IoT intrusion detection systems. Their proposed model is split into two components: the traffic analysis engine and classification engine. The traffic analysis engine preprocesses the data so that it is ready for classification. The RNN model is used to classify 'Normal' and 'Attack' IoT network traffic. They reported an accuracy of 92.18% and an F1 score of 92.29 %. This paper tackles binary classification of normal and attack traffic but does not further classify the type of attack.

The IoTID20 dataset was also used by A. A. Alsulami et al. [18] who employed machine learning to implement an IDS for IoT traffic with improved data engineering. Their system aims to detect normal and anomaly network traffic, as well as classify them into five categories: normal, Mirai attack, denial of service attack, Scan attack, and man-in-the-middle attack. They test multiple machine learning models, including shallow neural networks (SNN), bagging trees (BT), DT, KNN, and SVM. In their results, all proposed models achieved a 100% accuracy for detection, and over 99.40% for classification. They also reported their confusion matrices, where other than KNNs, all other models show abnormally high precision, recall, F1 score, and specificity, many of which are 100%. The paper's credibility is brought into question due to its reported 100% accuracies and F1 scores, suggesting potential overfitting or methodological issues that require further investigation.

The analysis presented in [19] investigates the benefit of using a Random Forest (RF) based model alongside the SVM and KNN approaches for malware and intrusion detection. The dataset used was the Aposemat IoT-23 dataset [20], which is a public dataset that has collected IoT data from 2018 to 2019. The performance of RF, SVM, and KNN was evaluated using its accuracy in both malware and intrusion detection. The performance was measured in terms of accuracy, with RF, SVM, and KNN reaching peak accuracies of 92.96%, 86.23%, and 91.48% respectively in intrusion detection, and 92.27%, 83.52%, and 89.80% in malware detection. The results show RF outperforms its counterparts in both malware and intrusion detection. Although the paper attempts to provide a comprehensive evaluation of ML methods in the context of IoT cybersecurity, it does not address the existence of more recent approaches such as deep learning approaches, which could have added a more comprehensive view on the field of IoT cybersecurity.

Taking a hybrid approach, M. Saharkhizan et al. [21] runs the data through an Long short-term memory (LSTM) model, then aggregates the output by a DT and assigns the right label to each network connection. They use a Modbus/TCP network traffic dataset generated in [22], where network traffic is classified as either clean traffic, man-in-the-middle attack, Ping DDoS Flood attack, Modbus Query Flood attack, or TCP SYN DDoS Flood attack. Their model reported an accuracy of 99.62% and an F1 score of 99.30%. While their proposed model shows great performance, the scope of the project is limited to Modbus traffic, making its value limited to general IoT traffic.

In the context of Internet of Medical Things (IoMT), G. Zachos et al. [23] explore the usage of an anomaly-based IDS to secure IoMT to prevent risk of patient data and halt of hospital operation. The authors proposed a complete network that includes IoMT sensors, monitoring, and data acquisition (MDA) components that collect monitor and acquire certain features from the IoMT devices, and a central detection (CD) component that monitors all network data going through the gateway to predict possible attacks. To train and

test the model, the authors used two datasets. The first is the TON_IoT Telemetry dataset, specifically the Train_Test Network sub-dataset [24]. The dataset is based on data collected from a testbed which researchers simulated various attack scenarios on. The other dataset generated by [25] is split into two powertraces: the benign powertrace and the malicious powertrace. Similar to the previous dataset, it is based on a testbed that simulates the behavior of an IoT network. The benign powertrace captures "benign" IoT network scenarios that represent normal operation. On the other hand, the malicious powertrace represents network data that includes a malicious actor within an IoT network. As for the models, the authors used the datasets to train a Naïve Bayes (NB) model, DT model, RF model, LR model, SVM model, and KNN model. To split the test and train data, the authors employ the use of four-fold cross validation. To evaluate the models, the authors calculated the accuracy, precision, recall, and F1 score for each of the trained models. They found that the DT, RF, and KNN models were the best performers on both datasets, with accuracy, precision, recall, and F1 score all in the 0.95-0.99 range. Overall, the authors' results seem to be good; however, few details were shared about the hyperparameter tuning steps they took and there was no mention of if binary or multiclass classification was evaluated in their tests.

Methods and Datasets

Dataset

The RT-IoT2022 dataset [1] is a labeled tabular dataset derived from a real-time IoT infrastructure that encompasses the network traffic information from different IoT devices. This information is classified depending on whether it is a network attack or normal traffic. The dataset has 123,117 samples, and 85 features in total. These features include network traffic information such as the network prototype, the service, and the flow_duration, as well as information about the packets being communicated such as the total, average, standard deviation, minimum and maximum statistical values for the forward and backward packets, flags, payloads and others. The dataset has 12 classes, 9 for attack patterns, and 3 for normal patterns. The descriptions of the classes are listed in the table below.

Table 1 - Description of RT-IoT2022 classes

	Network Patterns	Count
Attack Patterns	DOS_SYN_Hping	94659
	ARP_poisioning	7750
	NMAP_UDP_SCAN	2590
	NMAP_XMAS_TREE_SCAN	2010
	NMAP_OS_DETECTION	2000
	NMAP_TCP_scan	1002
	DDOS_Slowloris	534
	Metasploit_Brute_Force_SSH	37
	NMAP_FIN_SCAN	28
Normal Patterns	MQTT	8108
	Thing_speak	4146
	Wipro_bulb_Dataset	253

Data Preprocessing Methods

Encoding

In this project, we apply two common encoding techniques for preprocessing categorical data in order to better facilitate the training process: One-Hot Encoding and Label Encoding. One-hot encoding is a process used to convert categorical variables into a form that could be provided to ML algorithms to improve prediction. For each categorical value a new binary feature (1 or 0) is introduced such that for any number of categorical features, the one-hot encoding would return a bit representation such that only 1 bit is '1' indicating the occurrence of that specific feature. This ensures that for any given observation or record, the encoded binary vector highlights the presence of a specific feature with a '1' and the absence of others with '0's. The "OneHotEncoder" from scikit-learn was used to transform the 'proto' and 'service' columns of the input data, then the "fit_transform" method is used to fit the encoder to the data and transform it. The encoded columns were then concatenated to the original input data's dataframe. As for the label encoding, it is used to assign a unique integer to each category value. The "LabelEncoder" from scikit-learn was used to convert target values into numerical values.

Scaling

When using models that are sensitive to the scale of the input data, scaling methods are used in order to transform features to be of a similar scale, ensuring a consistent and comparable range across all features. In this project, the "MinMaxScaler" from scikit-learn was used in order to scale the largest occurring data point to the maximum value of '1' and the smallest one corresponds to the minimum value of '0'.

Balancing

In the context of this project, balancing refers to addressing class imbalance within the dataset to ensure that each class is equally represented during model training. Two common techniques utilized for balancing are undersampling and oversampling.

Undersampling is the reduction in the number of instances that majority classes have in order to match minority classes. In this project, the "ClusterCentroids" function from the imbalanced-learn library is employed for undersampling. The function randomly selects a subset sample from each majority class such that the resulting dataset is more balanced. This can be done by setting a parameter called the "sampling_strategy" to a specified number, which is the number of samples to take for each. In this project, the majority classes we undersample to have 3000 samples.

On the other hand, oversampling aims to increase the number of instances from minority classes to match that of the majority classes. The Synthetic Minority Over-sampling Technique (SMOTE) method from the imbalanced-learn library was used for oversampling.

Machine Learning Methods

Decision Trees

Decision trees are a type of supervised learning algorithm that can be used for binary classification, multiclass classification, and even regression. They are built upon the tree data structure, where each node represents a decision point based on a single feature from the dataset, and the branches of the node represent the outcome of the decision taken at the node. The tree begins with the root node which represents the whole unsplit dataset and is followed by a series of decision nodes that split the dataset into smaller and smaller subsets. The tree ends with leaf nodes that represent the possible outcomes from the tree. Figure 1 represents a typical decision tree layout.

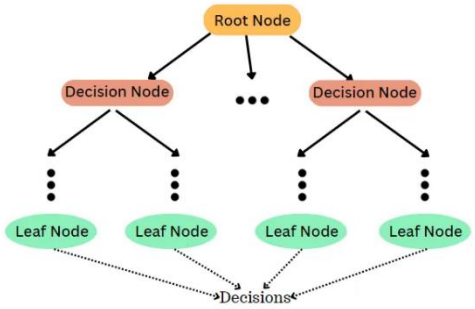

Figure 1 - Typical decision tree structure [26]

The objective of each split in a decision tree is to create purer subsets until, ideally, each subset includes only data from one class. To help with this, the best feature to be used as the decision factor at each node can be determined through various impurity metrics such as the Gini index, entropy, and classification error. Equations 1-3 define the metrics used for splitting:

$$GINI(t) = 1 - \sum_{i=0}^{c-1} p_i(t)^2 \quad (1)$$

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) \log_2(p_i(t)) \quad (2)$$

$$Classification\ error = 1 - [p_i(t)] \quad (3)$$

$$Where\ p_i(t) is\ the\ frequency\ of\ class\ i\ at\ node\ t\ and\ c\ is\ is\ the\ total\ number\ of\ classes$$

To assess the impurity after splitting, weighted average of the Gini index and entropy can be calculated as seen in equations 4 & 5:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(t) \quad (4)$$

$$Entropy_{split} = -\sum_{i=1}^{k} \frac{n_i}{n} Entropy(t) \quad (5)$$

$$Where\ ni\ is\ number\ of\ records\ at\ child\ i\ and\ n\ is\ number\ of\ records\ at\ parent\ p$$

Finally, to choose which split is best, the information gain can be calculated:

$$Information\ gain = P - M \quad (6)$$

$$Where\ P\ is\ the\ impurity\ (GINI\ or\ Entropy)\ before\ splitting\ and\ M\ is\ the\ impurity\ after\ splitting$$

Usually, a decision tree that is based on the greedy approach would split until either all subsets are completely pure or until all attributes at a given node are identical (no more possible splits). Hyperparameters can also be used to adjust the tree structure, complexity, and performance. Key hyperparameters include max tree depth, min samples per split, min samples per leaf, max number of features, max number of lead nodes, etc.

K-nearest Neighbors

The K-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning model that is primarily used for classification problems. It works by predicting the category of a new data point based on its distance from other samples provided in training. The data point's classification will be based on the 'k' closest points as per the distance measure, where the algorithm could either:

- Simply place the new point in the majority class of the k-nearest samples
- Consider the distance, where a weight factor is calculated used in the summation

The weight factor can be calculated as follows:

$$Weight\ factor = \frac{1}{d^2} \tag{7}$$

To calculate the distance, two measures can be calculated:

$$Euclidean\ distance = D(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{8}$$

$$Cosine\ distance = cos\ (\theta) = \frac{AB}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{9}$$

$$Where\ i\ is\ a\ feature\ from\ the\ features\ list$$

Typically, the Euclidean distance is used more commonly; however, there are certain data where using the cosine distance is better, such as when dealing with documents.

For better results with KNNs, the data should be normalized/scaled to prevent a feature to dominate the distance measure and skew the results. Additionally, 'k' should be chosen such that it's not too small or large so that it does not suffer from noisy points or from points from other classes respectively.

Naïve Bayes

The Naïve Bayes classification algorithm is a probabilistic classifier based on Bayes' Theorem. Bayes' Theorem allows us to calculate the posterior probability P(Y|X) as:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{10}$$

Additionally, P(X|Y) can be expressed as follows:

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)} \tag{11}$$

The idea behind the Naïve Bayes algorithm is that we want to estimate the probability of class Y given the list of attributes X_1, X_2, …, X_d. We then assign the new point to the class which maximizes the posterior probability $P(Y|X_1, X_2, \ldots, X_d)$. To get this probability:

$$P(Y|X_1, X_2, \ldots, X_d) = \frac{P(X_1, X_2, \ldots, X_d|Y)P(Y)}{P(X_1, X_2, \ldots, X_d)} \tag{12}$$

$P(X_1, X_2, \ldots, X_d|Y)$ can then be estimated from the train data through the assumption that all features X_1, X_2, …, X_d are independent which allows us to express the conditional probability as:

$$P(Y) = P(Y) * P(Y) * \ldots * P(Y) \tag{13}$$

From this, each conditional probability can be defined from the frequency of features X_i for samples in class Y:

$$P(Y) = \frac{n_c}{n} \tag{14}$$

$$n_c\ is\ the\ frequency\ of\ attribute\ X_i = c\ beloging\ to\ Y\ and\ n\ is\ frequency\ of\ class\ Y$$

The Naïve Bayes algorithm works on continuous attributes too. This is done by either discretizing the continuous attributes which would change them to ordinal values, or by estimating the conditional probability by assuming that the attributes are normally distributed. In the latter case, the following equation can be used:

$$P(Y_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(X_i - \mu_i)^2}{2\sigma_{ij}^2}} \tag{15}$$

One of the issues faced by the Naïve Bayes algorithm is that, because of the assumption of independence of attributes, if one of the conditional probabilities $P(X_i|Y) = 0$, then the whole expression becomes 0. To combat this, two other probability estimation methods can be used:

$$Laplace\ Estimate: P(Y) = \frac{n_c + 1}{n + v} \tag{16}$$

$$m - estimate: P(Y) = \frac{n_c + mp}{n + m} \tag{17}$$

$$Where\ v\ is\ the\ number\ of\ unique\ values\ for\ X_i, p\ is\ the\ inital\ estimate\ of\ P(Y), and$$

$$m\ is\ the\ hyperparameter\ for\ confidence\ in\ p$$

Support Vector Machines

The Support Vector Machines (SVM) is a type of supervised machine learning algorithm used for classification and regression problems. The principle behind is trying to find a hyperplane that best divides the dataset into classes while also maximizing the margins between the nearest points of each class and the hyperplane. Maximizing the margin is done to improve the model's generalization capabilities. There are two types of SVMs:

1. Linear SVMs: Used when the data is linearly separable. The hyperplane used is linear.
2. Nonlinear SVMs: Used when the data is not linearly separable. A kernel function is used, and the hyperplane will be nonlinear.

For linear SVMs. The model finds the hyperplane using the following condition:

$$\vec{w} * \vec{x} + b = 0 \tag{18}$$

$$Where\ w\ is\ the\ weigh\ vector, x\ is\ the\ input\ data, and\ b\ is\ the\ bias$$

For the nonlinear SVMs, a kernel function maps the data to a higher dimensional space where linear separation is possible. Common kernel functions include:

$$Polynomial\ kernel: K(x, y) = (x * y + 1)^2 \tag{19}$$

$$Radial\ Basis\ Function\ (RBF): K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}} \tag{20}$$

$$Hyperbolic\ tangent\ (sigmoid)\ kernel: K(x, y) = tanh\ (kxy - \delta) \tag{21}$$

After applying the kernel function, the hyperplane can be found using the following condition:

$$\vec{w} * \phi(\vec{x}) + b = 0 \tag{22}$$

There are a several parameters that can be tuned in an SVM model:

- Linear or nonlinear SVM
- Kernel type
- For RBF kernel, C, which controls the trade-off between achieving low error on training data and a smooth decision boundary (the higher the C, the lower the error)
- For RBF kernel, Gamma, which defines how fat the influence of a single training example reaches. The higher the gamma, the close the reach, and thus usually the less smooth the hyperplane
- For polynomial kernel, the degree of the kernel. The higher the degree, the higher the dimensionality

Feature selection and Dimensionality Reduction

Dimensionality reduction via feature selection can be done to reduce the complexity of our models while retaining the relevant attributes. The idea is that the dimensionality of the data can be reduced by selecting the most relevant attributes. In our case, the "SelectPercentile" univariate feature selection method can be used for the reduction. This method works by running a scoring function on each attribute and then choosing the highest scoring features that are within the percentile specified. Multiple scoring functions are available that depend on

statistical analysis to measure the correlation between each attribute and the class, with the default one being the ANOVA test. To optimize the reduction while still maintaining high accuracy F1-score, the "SelectPercentile" method is run at various percentile levels to identify the most effective configuration.

Model Selection

To effectively apply the classifiers mentioned previously, the dataset needs to be divided into training and testing subsets. This is done using the holdout method. The holdout method splits the data using random or stratified sampling, maintaining the representativeness of the data in both sets. Subsequently, the models are trained on the training subset, and their predictive performance is evaluated on the test subset, producing evaluation metrics such as accuracy, precision, recall, and F1-score. To mitigate the influence of randomness inherent in the data splitting process, this procedure is repeated 10 times with different random splits. Finally, the average metrics across all iterations are calculated, providing a more accurate assessment of the models' ability to predict unseen data.

Evaluation Metrics

Accuracy

Accuracy is the most widely used metric when it comes to evaluating the performance of an ML model. It calculates the ratio of correct predictions to all predictions made. This metric is particularly useful if the dataset at hand is balanced. Therefore, for unbalanced models, accuracy might not be the best metric to rely on solely, as it does not give the full picture of how well the model is performing.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{23}$$

Precision

Precision is the proportion of the number of true positive predictions to the total number of positive predictions. This means that a high precision indicates a low false positive rate.

$$Precision = \frac{TP}{TP + FP} \tag{24}$$

Recall

Recall is the proportion of actual positives that are correctly identified by the model. A high recall indicates a low rate of false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{25}$$

F1 Score

F1 score is the harmonic mean of precision and recall, which means that these metric balances both precision and recall metrics. F1 score is a more robust measure of the performance of the model compared to accuracy alone, especially in the case of imbalance data.

$$F1\ Score = \frac{2(Precision * Recall)}{Precision + Recall} \tag{26}$$

AUC

The AUC (Area Under Curve) is the area under the ROC (Receiver Operating Characteristic) curve. The curve plots the true positive rate against false positive rate at various threshold settings. The AUC metric measures the ability of a model to distinguish between classes and is used as a summary of the model's performance across all classification thresholds.

Results

Data Preprocessing

The data underwent several preprocessing stages. Initially, the two features 'id.orig_p' and 'id.resp_p', representing the origin and response ports, were dropped to avoid overfitting the model using these features, since most attacks in this model were manually generated from fixed devices at fixed ports. Next, the qualitative data were encoded. One hot encoding was performed on the two categorical features of the dataset: the network prototype and the service. In addition, the labels were encoded using label encoding. After encoding all categorical features, MinMax scaling was used to ensure all data shared the same scale. The resulting data was transformed into values from 0 to 1. Finally, a copy of the data has also been balanced using under-sampling and over-sampling until all labels had 3000 corresponding samples. This was done using imbalanced-learn's ClusterCentroids function and the SMOTE function.

Machine Learning Results

Decision Trees

To obtain the optimal Decision Tree model, four hyperparameters were tuned. These hyperparameters are namely the maximum depth (max_depth), the minimum number of samples required to split a node (min_sample_split), the minimum number of samples required to be considered a leaf node (min_sample_leaf), and the complexity parameter used for Minimal Cost-Complexity Pruning (ccp_alpha). The following are the results of the analysis of each hyperparameter to find the best parameter with the best accuracy-complexity tradeoff to be used to tune the model.

Unbalanced dataset:

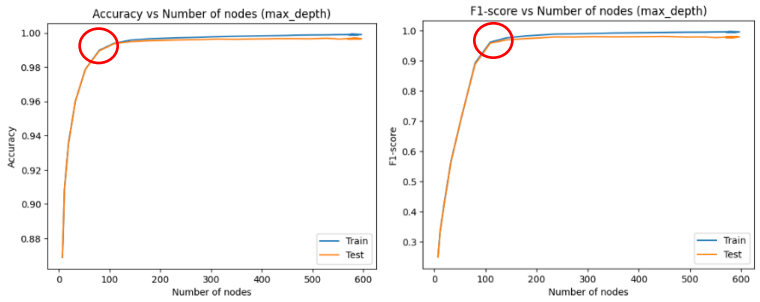

Figure 2 – Unbalanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on max_depth

Figure 2 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's max_depth and training on the unbalanced dataset. The range of tested max_depth is from 2 to 100, with a unit interval. The optimal max_depth is found to be 7, where the average accuracy is 0.995, the average 'macro' F1 score is 0.969, and the average number of nodes is 141.

Figure 3 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's min_samples_split and training on the unbalanced dataset. The range of tested min_samples_split is from 2 to 2000, with an interval of 10. The optimal min_samples_split is found to be 122, where the average accuracy is 0.995, the average 'macro' F1 score is 0.964, and the average number of nodes is 151.

Figure 4 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's min_samples_leaf and training on the unbalanced dataset. The range of tested min_samples_leaf is from 2 to 2000, with an interval of 10. The optimal min_samples_leaf is found to be 22, where the average accuracy is 0.994, the average 'macro' F1 score is 0.961, and the average number of nodes is 171.

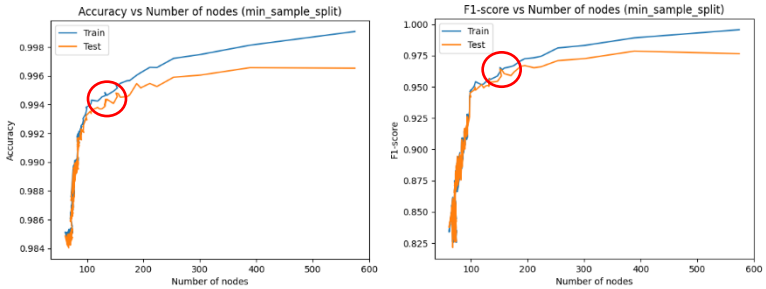

Figure 3 - Unbalanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on min_sample_split

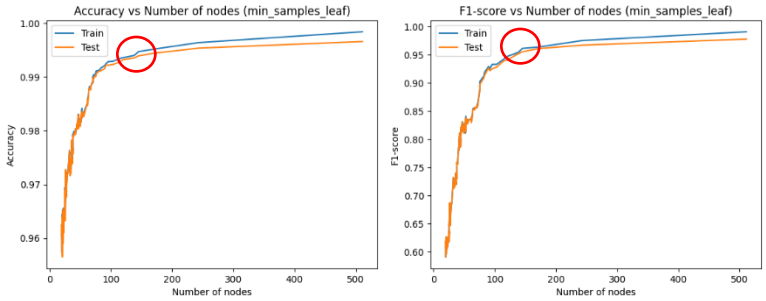

Figure 4 - Unbalanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on min_samples_leaf

Figure 5 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's ccp_alpha and training on the unbalanced dataset. The range of tested ccp_alpha is from 0 to 0.001, with an interval of 0.00001. The optimal ccp_alpha is found to be 0.00008, where the average accuracy is 0.995, the average 'macro' F1 score is 0.971, and the average number of nodes is 106.

Of the four tuned parameters, all of them have shown similar level of accuracy; however, ccp_alpha has the best F1-score with a relatively lower number of nodes than the other hyperparameters. Therefore, a new model was fit with ccp_alpha tuned at 0.00008. This model was tested with and without feature selection. Figure 6a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9955, a 'macro' F1-score of 0.9732, a 'macro' precision of 0.9754, a 'macro' recall of 0.9713, and a 'macro' ROC AUC of 0.99863. In addition, the number of nodes in the tree was 107. Figure 6b shows the confusion matrix resulted from the model fitted on 50% of the features after applying feature selection. The model has an accuracy of 0.9949, a 'macro' F1-score of 0.9730, a 'macro' precision of 0.9765, a 'macro' recall of 0.9701, and a 'macro' ROC AUC of 0.99843. In addition, the number of nodes in the tree was 113.

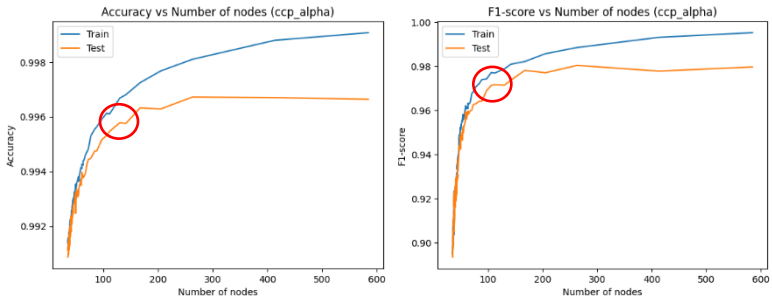

Figure 5 - Unbalanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on ccp_alpha

Figure 6 - Confusion matrix of ccp_alpha tuned decision tree for unbalanced dataset. a) all features used b) 50% features used

Balanced dataset:

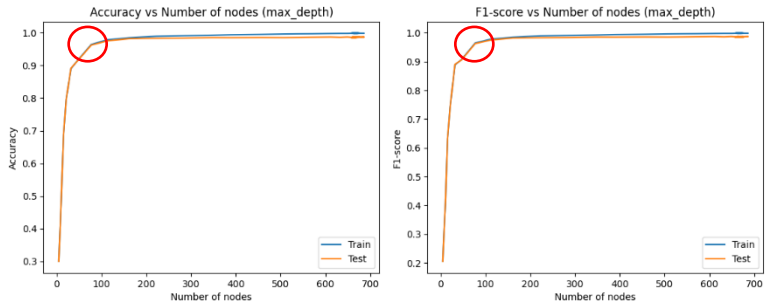

Figure 7 - Balanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on max_depth

Figure 7 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's max_depth, and training on the balanced dataset. The range of tested max_depth is similarly from 2 to 100, with a unit interval. The optimal max_depth is found to be 10, where the average accuracy is 0.975, the average 'macro' F1 score is 0.975, and the average number of nodes is 113.

Figure 8 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's min_samples_split and training on the balanced dataset. The range of tested min_samples_split is from 2 to 2000, with an interval of 10. The optimal min_samples_split is found to be 252, where the average accuracy is 0.980, the average 'macro' F1 score is 0.980, and the average number of nodes is 115.

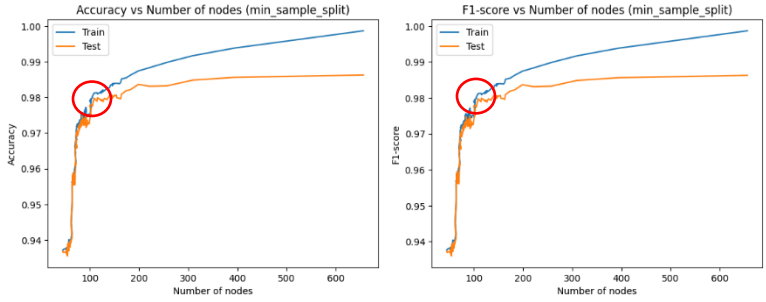

Figure 8 - Balanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on min_sample_split

Figure 9 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's min_samples_leaf and training on the balanced dataset. The range of tested min_samples_leaf is from 2 to 2000, with an interval of 10. The optimal min_samples_leaf is found to be 72, where the average accuracy is 0.972, the average 'macro' F1 score is 0.972, and the average number of nodes is 102.

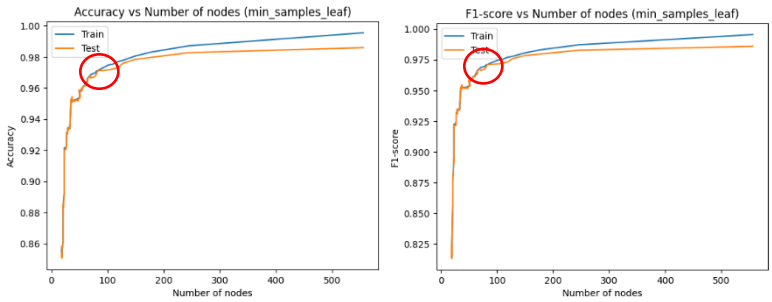

Figure 9 - Balanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on min_samples_leaf

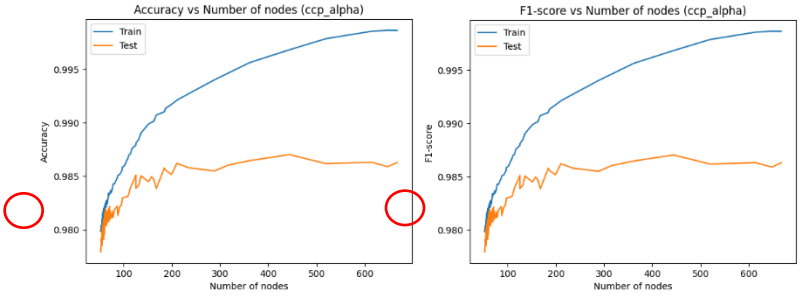

Figure 10 - Balanced dataset's decision tree's a) accuracy and b) F1 score vs number of nodes based on ccp_alpha

Figure 10 shows the relationship between the model's accuracy and the F1-score versus the number of nodes when tuning the decision tree's ccp_alpha and training on the balanced dataset. The range of tested ccp_alpha is from 0 to 0.001, with an interval of 0.00001. The optimal ccp_alpha is found to be 0.00025, where the average accuracy is 0.983, the average 'macro' F1 score is 0.983, and the average number of nodes is 101.

All hyperparameters show very close average accuracies and F1-scores at similar complexities. However, ccp_alpha has a slightly better average accuracy and F1-score than the other parameters. Therefore, a new model was fitted with ccp_alpha tuned at 0.00025. Figure 11a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9848, a 'macro' F1-score of 0.9848, a 'macro' precision of 0.9849, a 'macro' recall of 0.9848, and a 'macro' ROC AUC of 0.99842. In addition, the number of nodes in the tree was 107. Figure 11b shows the confusion matrix resulted from the model fitted on 30% of the features after applying feature selection. The model has an accuracy of 0.9847, a 'macro' F1-score of 0.9847, a 'macro' precision of 0.9850, a 'macro' recall of 0.9847, and a 'macro' ROC AUC of 0.99821. In addition, the number of nodes in the tree was 123.

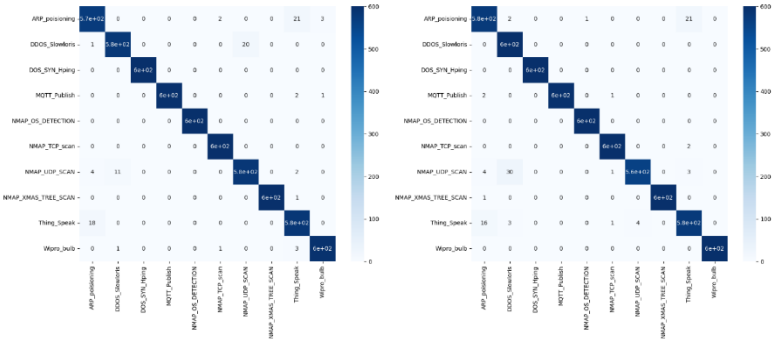

Figure 11 - Confusion matrix of ccp_alpha tuned decision tree for balanced dataset. a) all features used b) 30% features used

Naïve Bayes

Under the assumption that the data follows a normal distribution, a Gaussian Naïve Bayes Model was fitted on the dataset. The following are the results of fitting the model on both the unbalanced and balanced datasets.

Unbalanced dataset:

Figure 12a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9344, a 'macro' F1-score of 0.7367, a 'macro' precision of 0.7528, a 'macro' recall of 0.8239, and a 'macro' ROC AUC of 0.99217. Figure 12b shows the confusion matrix resulted from the model fitted on 40% of the features after applying feature selection. The model has an accuracy of 0.9512, a 'macro' F1-score of 0.8082, a 'macro' precision of 0.8294, a 'macro' recall of 0.8453, and a 'macro' ROC AUC of 0.99513.

Figure 12 - Confusion matrix of Naïve Bayes for unbalance dataset. a) all features used b) 40% features used

Balanced dataset:

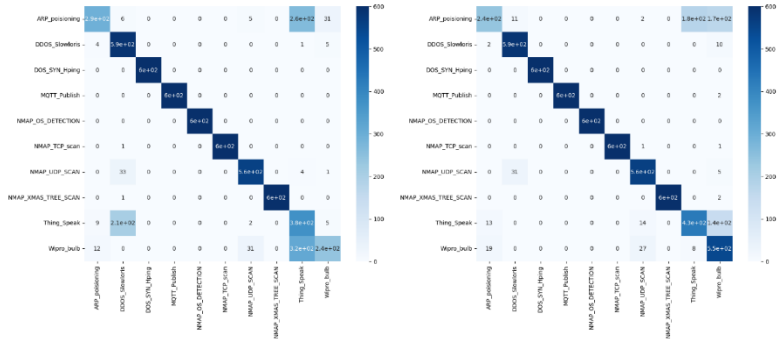

Figure 13 - Confusion matrix of Naïve Bayes for balanced dataset. a) all features used b) 30% features used

Figure 13a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.8435, a 'macro' F1-score of 0.8422, a 'macro' precision of 0.8805, a 'macro' recall of 0.8435, and a 'macro' ROC AUC of 0.98363. Figure 13b shows the confusion matrix resulted from the model fitted on 30% of the features after applying feature selection. The model has an accuracy of 0.8933, a 'macro' F1-score of 0.8876, a 'macro' precision of 0.9054, a 'macro' recall of 0.8933, and a 'macro' ROC AUC of 0.98735.

K-nearest Neighbor

To obtain the optimal KNN model, the number of neighbors (n_neighbors) were tuned. The following are the results of the analysis of the n_neighbors to find the optimal number with the best F1 score to be used to tune the model.

Unbalanced dataset:

Figure 14 shows the relationship between the model's accuracy and the F1-score versus n_neighbors when tuning the KNN model's n_neighbors and training on the unbalanced dataset. The range of tested n_neighbors is from 1 to 200, though Figure 14 shows a closer look at the range 1 to 20. The optimal n_neighbors is found to be 3, where the average accuracy is 0.997, and the average 'macro' F1 score is 0.981.

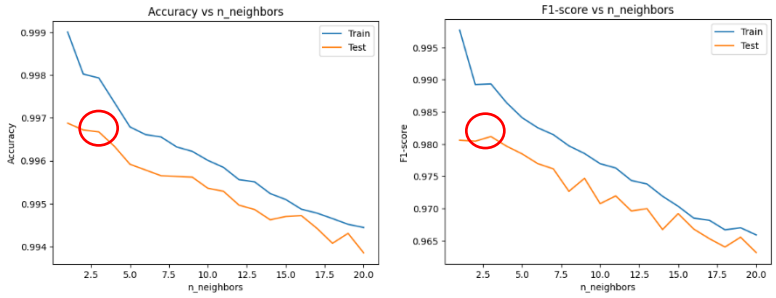

Figure 14 - Unbalanced dataset's KNN a) accuracy and b) F1 score vs number of neighbors

Figure 15 - Confusion matrix of KNN for unbalanced dataset. a) all features used b) 75% features used

Using the results from above, a new model was fit with n_neighbors tuned at 3. Figure 15a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9968, a 'macro' F1-score of 0.9804, a 'macro' precision of 0.9843, a 'macro' recall of 0.9767, and a 'macro' ROC AUC of 0.99356. Figure 15b shows the confusion matrix resulted from the model fitted on 75% of the features after applying feature selection. The model has an accuracy of 0.9968, a 'macro' F1-score of 0.9855, a 'macro' precision of 0.9815, a 'macro' recall of 0.9900, and a 'macro' ROC AUC of 0.99775.

Balanced dataset:

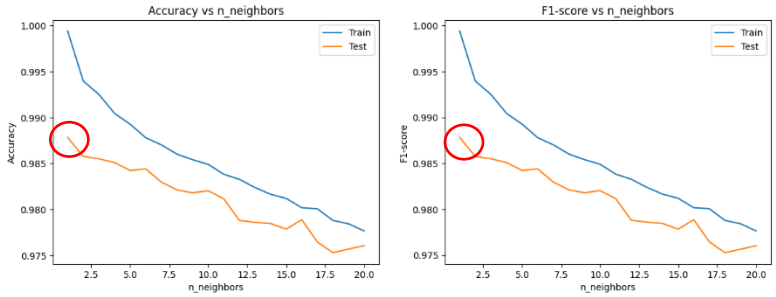

Figure 16 - Balanced dataset's KNN a) accuracy and b) F1 score vs number of neighbors

Figure 16 shows the relationship between the model's accuracy and the F1-score versus n_neighbors when tuning the KNN model's n_neighbors and training on the balanced dataset. The range of tested n_neighbors is from 1 to 200, though Figure 16 shows a closer look at the range 1 to 20. The optimal n_neighbors is found to be 1, where the average accuracy is 0.988, and the average 'macro' F1 score is 0.988.

Using the results from above, a new model was fit with n_neighbors tuned at 1. Figure 17a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9883, a 'macro' F1-score of 0.9883, a 'macro' precision of 0.9883, a 'macro' recall of 0.9883, and a 'macro' ROC AUC of 0.99352. Figure 17b shows the confusion matrix resulted from the model fitted on 75% of the features after applying feature selection. The model has an accuracy of 0.9887, a 'macro' F1-score of 0.9887, a 'macro' precision of 0.9887, a 'macro' recall of 0.9887, and a 'macro' ROC AUC of 0.99370.

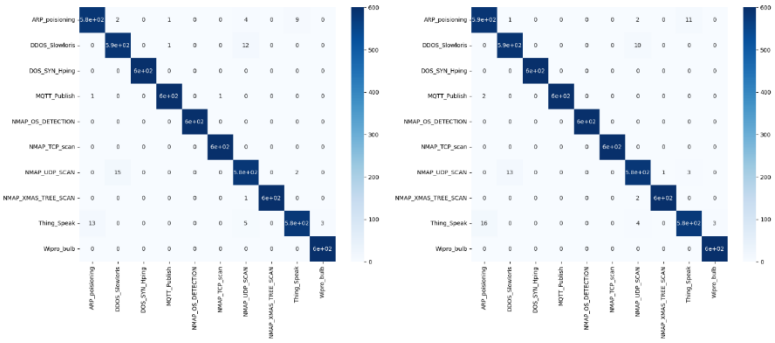

Figure 17 - Confusion matrix of KNN for balanced dataset. a) all features used b) 75% features used

Linear SVM

Next are the results of fitting a Linear SVM model on both the unbalanced and balanced datasets.

Unbalanced dataset:

Figure 18 - Confusion matrix of linear SVM for unbalanced dataset. a) all features used b) 70% features used

Figure 18a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9854, a 'macro' F1-score of 0.9391, a 'macro' precision of 0.9669, and a 'macro' recall of 0.9207. Figure 18b shows the confusion matrix resulted from the model fitted on 70% of the features after applying feature selection. The model has an accuracy of 0.9834, a 'macro' F1-score of 0.9360, a 'macro' precision of 0.9629, and a 'macro' recall of 0.9195.

Balanced dataset:

Figure 19a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9630, a 'macro' F1-score of 0.9628, a 'macro' precision of 0.9637, and a 'macro' recall of 0.9630. Figure 19b shows the confusion matrix resulted from the model fitted on 65% of the features after applying feature selection. The model has an accuracy of 0.9628, a 'macro' F1-score of 0.9627, a 'macro' precision of 0.9643, and a 'macro' recall of 0.9628.

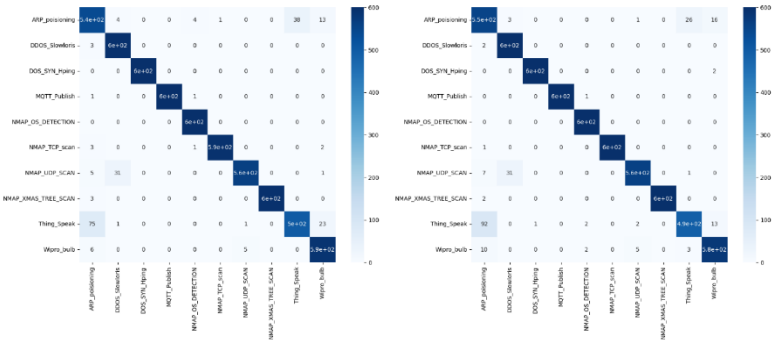

Figure 19 - Confusion matrix of linear SVM for balanced dataset. a) all features used b) 65% features used

Non-linear SVM

Two different kernels were tested for non-linear SVMs: the polynomial kernel and the radial basis function (RBF) kernel. For the polynomial kernel, the degree is tuned, while for the RBF kernel, C and gamma were tuned separately. Below are the results after tuning these parameters on both the unbalanced and the balanced datasets.

Unbalanced dataset:

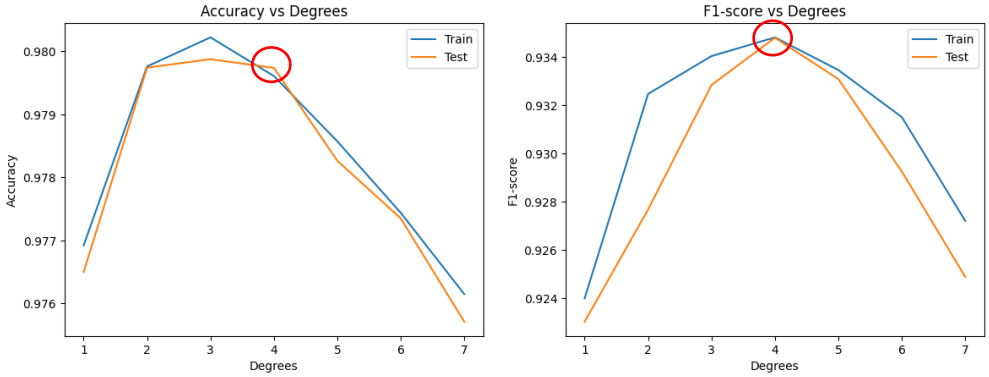

Figure 20 - Unbalanced dataset's nonlinear SVM with polynomial kernel a) accuracy and b) F1 score vs degrees

Figure 20 shows the relationship between the model's accuracy and the F1-score versus degrees when tuning the SVM model's degree for the polynomial kernel, and training on the unbalanced dataset. The range of tested degrees is from 1 to 7, with a step of 1. The optimal degrees is found to be 4, where the average accuracy is 0.980, and the average 'macro' F1 score is 0.935.

Figure 21 shows the relationship between the model's accuracy and the F1-score versus C when tuning the SVM model's C parameter for the RBF kernel, and training on the unbalanced dataset. The range of tested C is from 5 to 100, with a step of 5. The optimal C is found to be 35, where the average accuracy is 0.994, and the average 'macro' F1 score is 0.966.

Figure 22 shows the relationship between the model's accuracy and the F1-score versus gamma when tuning the SVM model's gamma parameter for the RBF kernel, and training on the unbalanced dataset. The range of tested gamma is from 5 to 100, with a step of 5. The optimal gamma is found to be 75, where the average accuracy is 0.994, and the average 'macro' F1 score is 0.965.

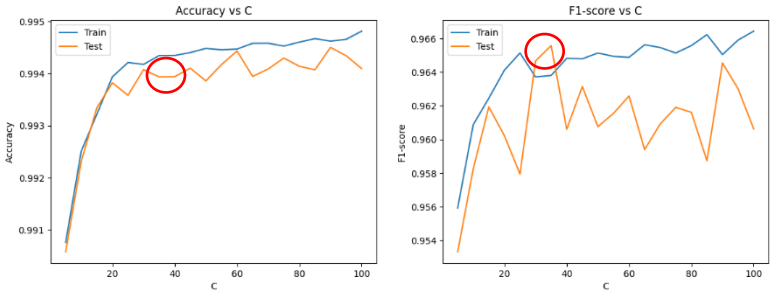

Figure 21 - Unbalanced dataset's nonlinear SVM with RBF kernel a) accuracy and b) F1 score vs C

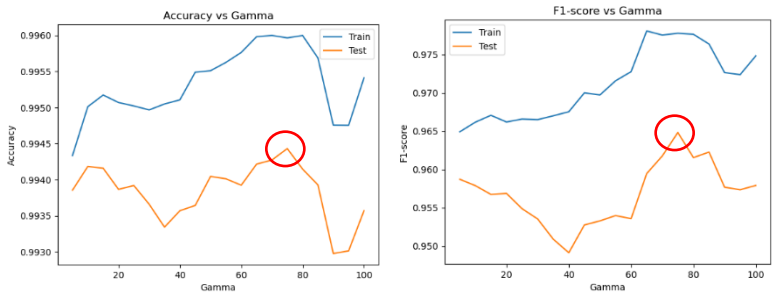

Figure 22 - Unbalanced dataset's nonlinear SVM with RBF kernel a) accuracy and b) F1 score vs gamma

Figure 23 - Confusion matrix of nonlinear SVM with RBF kernel unbalanced dataset. a) all features used b) 90% features used

The results above show that using the RBF kernel and tuning either C or gamma will result in optimal results. Therefore, a new model was fit with C tuned at 35. Figure 23a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9943, a 'macro' F1-score of 0.9649, a 'macro' precision of 0.9856, a 'macro' recall of 0.9488, and a 'macro' ROC AUC of 0.99538. Figure 23b shows the confusion matrix resulted from the model fitted on 90% of the features after applying feature selection. The model has an accuracy of 0.9941, a 'macro' F1-score of 0.9679, a 'macro' precision of 0.9882, a 'macro' recall of 0.9525, and a 'macro' ROC AUC of 0.99571.

Balanced Dataset:

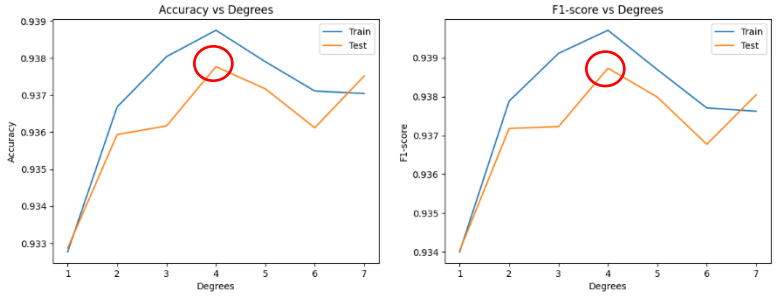

Figure 24 - Balanced dataset's nonlinear SVM with polynomial kernel a) accuracy and b) F1 score vs degrees

Figure 24 shows the relationship between the model's accuracy and the F1-score versus degrees when tuning the SVM model's degrees parameter for the polynomial kernel, and training on the balanced dataset. The range of tested degrees is from 1 to 7, with a step of 1. The optimal degree found is 4, where average accuracy is 0.938, and the average 'macro' F1 score is 0.939.

Figure 25 shows the relationship between the model's accuracy and the F1-score versus C when tuning the SVM model's C parameter for the RBF kernel, and training on the balanced dataset. The range of tested C is from 5 to 100, with a step of 5. The optimal C is found to be 40, where the average accuracy is 0.979, and the average 'macro' F1 score is 0.979.

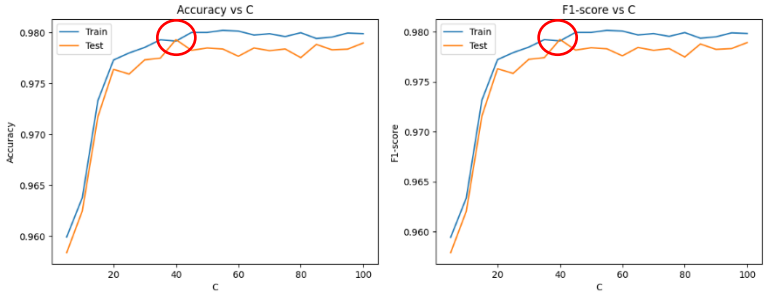

Figure 25 - Balanced dataset's nonlinear SVM with RBF kernel a) accuracy and b) F1 score vs C

Figure 26 shows the relationship between the model's accuracy and the F1-score versus gamma when tuning the SVM model's gamma parameter for the RBF kernel, and training on the balanced dataset. The range of tested gamma is from 5 to 100, with a step of 5. The optimal gamma is 10, where average accuracy is 0.978, and the average 'macro' F1 score is 0.978.

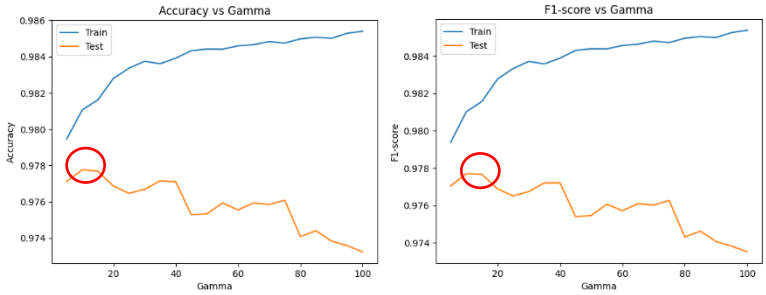

Figure 26 - Balanced dataset's nonlinear SVM with RBF kernel a) accuracy and b) F1 score vs gamma

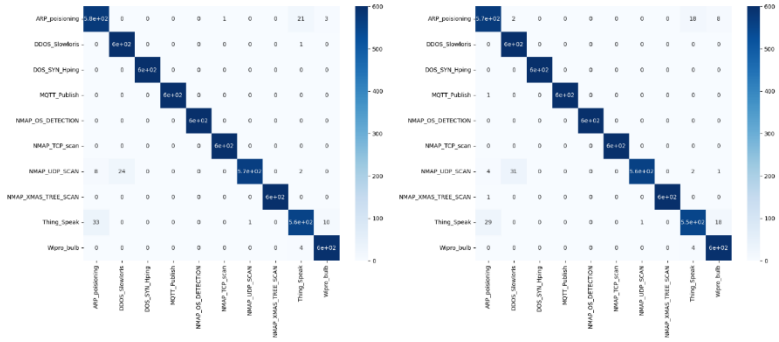

Figure 27 - Confusion matrix of nonlinear SVM with RBF kernel for balanced dataset. a) all features used b) 65% features used

The results above show that using the RBF kernel and tuning either C or gamma will result in optimal results. Therefore, a new model was fit with C tuned at 40. Figure SVM2a shows the confusion matrix resulted from the model fitted on the whole data with all features selected. The model has an accuracy of 0.9820, a 'macro' F1-score of 0.9820, a 'macro' precision of 0.9822, a 'macro' recall of 0.9820, and a 'macro' ROC AUC of 0.99853. Figure SVM2b shows the confusion matrix resulted from the model fitted on 65% of the features after applying feature selection. The model has an accuracy of 0.9800, a 'macro' F1-score of 0.9799, a 'macro' precision of 0.9803, a 'macro' recall of 0.9800, and a 'macro' ROC AUC of 0.99824.

Discussion

The results in Table 2 & 3 illustrate the performance of different models when trained with both balanced and unbalanced RT-IoT2022 datasets. As of now, only the Decision Tree (DT) model has been examined. It is important to note that selectPercentile feature selection technique was used and therefore the results reported in Table 3 cover results with feature selection.

Decision Tree (DT)

In the unbalanced dataset scenario, there was a relatively noticeable difference between the accuracy and macro F1-score being 0.9955 and 0.9702, respectively. This means that due to the imbalance the model tends to predict the majority class more accurately, leading to a high accuracy and a lower macro F1-score. This might suggest that while the model is good at predicting the majority class, it doesn't perform as well across all classes. However, in the case of the balanced dataset, the accuracy was 0.9867 and macro F1-score was 0.9867. This shows that while the model's accuracy on the balanced dataset is slightly lower, the high accuracy on the unbalanced dataset was misleading, masking its poor performance on the minority classes. Moreover, the balanced dataset achieved a higher macro F1-score as well as being comparable with the accuracy. This indicates that the model's predictive performance is more uniform across all classes. It is also worth noting that both macro precision and recall values exhibit improvement. This suggest that the balanced dataset although there is a slight decrease in accuracy, it is accompanied by an improvement in the macro F1-score, precision and recall, highlighting the fact that by addressing class imbalance through balancing techniques, the model achieves better generalization and reliability in classifying RT-IoT2022 dataset instances. Moreover, when introducing feature selection, DT yielded similar results, suggesting it may have not significantly affected the model's susceptibility to imbalance. Overall, the results suggest that DT are sensitive to class imbalances as they tend to over-predict the majority class, and therefore underperform compared to when the data has an even class distribution.

Naïve Bayes (NB)

The performance of NB was the worst compared to the rest of the models. This can be attributed to the fact that the gaussian NB function on sklearn assumes all the data is normal, which might not be the case for our dataset. In the unbalanced dataset, NB achieved an accuracy of 0.9344 suggesting that NB struggled to differentiate between the classes, especially the minority class, leading to more misclassifications. This is further supported by the significantly lower recall of 0.8239 and precision of 0.7528 of NB compared to other models. The F1-score of 0.7367 also reflects this, indicating a weaker overall performance for NB in the unbalanced scenario compared to other models. It's important to note that AUC of 0.99513 is high but can be misleading in an imbalanced scenario. Even after balancing the data, although NB improved in terms of precision, recall, and F1-score, it still lagged behind the other models. The precision, recall, and F1-score improved to 0.8805, 0.8435, and 0.8422, respectively, while the accuracy of balanced dataset using NB dropped from 0.9344 to 0.8435, better reflecting the true accuracy of the model. Moreover, feature selection with NB significantly improved results for both balanced and unbalanced dataset. For the unbalanced dataset, the accuracy increased from 0.9344 to 0.9512, while metrics such as precision, recall, and F1-score improved significantly from 0.7528, 0.8239, and 0.7367 to 0.8294, 0.8453, and 0.8082. As for the balanced dataset, the improvement of feature selection was relatively more than that of unbalanced having achieved accuracy of 0.8933 from 0.8435, precision of 0.9054 from 0.8805, recall of 0.8933 from 0.8435, and F1-score of 0.8876 from 0.7367.

K-nearest Neighbor

KNN achieved impressive overall results compared to other models. For the unbalanced dataset, the accuracy was the highest among all other model accuracies having achieved 0.9968. However, it can be noticed that due to a relatively lower precision and recall, the F1-score of KNN on unbalanced dataset was slightly compromised having achieved precision of 0.9843, recall of 0.9767, and F1-score of 0.9804. As for the balanced dataset, the accuracy dropped to 0.9883 but the model achieved better generalizability by having an increase in its precision, recall, and F1-scores to 0.9883 each. As for feature selection, the accuracy did not improve in the unbalanced dataset scenario, but precision, recall, and F1-score increased to 0.9815, 0.9900, and 0.9855, respectively. Moreover, KNN when feature selection was applied to balanced datasets achieved 0.9887 across accuracy, precision, recall, and F1-score metrics as well as getting 0.99370 AUC. This makes KNN when feature selection was applied to balanced dataset the best performing model due to achieving the highest F1-score of 0.9887.

Linear SVM

Linear SVM performed well compared to other models. For the unbalanced dataset, linear SVM achieved a high accuracy of 0.9854 but due to its considerably lower recall of 0.9207, the F1-score achieved was 0.9391. For the balanced dataset, although the accuracy dropped from 0.9854 to 0.9630, the results showcase a more robust model with a higher F1-score of 0.9628 from 0.9391. This is mainly attributed to the noticeable increase in the recall from 0.9207 to 0.9630. Moreover, applying feature selection to linear SVM for both balanced and unbalanced scenario exhibited negligible difference.

Non-Linear SVM

The non-linear SVM overall performed better than linear SVM. For the unbalanced dataset, the non-linear SVM scored an impressive 0.9943 accuracy compared to the linear SVM's 0.9854. However, it also showcases a lower F1-score (compared to accuracy) due to its considerably lower recall of 0.9488, bringing its F1-score to 0.9649. For the balanced dataset, although the accuracy dropped from 0.9943 to 0.9820, the results showcase a more robust model with a higher F1-score of 0.9820 from 0.9649. Like the linear SVM models, applying feature selection to linear SVM for both balanced and unbalanced scenario exhibited negligible difference. This shows that while non-linear SVM performs overall better than linear-SVM, they exhibit very similar behaviour when balancing or applying feature selection.

Table 2 - Summary of Results (without feature selection)

			Without Feature Selection				
			Accuracy	Precision	Recall	F1-Score	AUC
DT	Unbalanced		0.9955	0.9754	0.9713	0.9732	**0.99863**
	Balanced		0.9848	0.9849	0.9848	0.9848	0.99842
NB	Unbalanced		0.9344	0.7528	0.8239	0.7367	0.99217
	Balanced		0.8435	0.8805	0.8435	0.8422	0.98363
KNN	Unbalanced		**0.9968**	0.9843	0.9767	0.9804	0.99356
	Balanced		0.9883	**0.9883**	**0.9883**	**0.9883**	0.99356
SVM	Linear	Unbalanced	0.9854	0.9609	0.9207	0.9391	-
		Balanced	0.9630	0.9637	0.9630	0.9628	-
	Non-Linear	Unbalanced	0.9943	0.9856	0.9488	0.9649	0.99538
		Balanced	0.9820	0.9822	0.9820	0.9820	0.99853

Table 3 - Summary of Results (with feature selection)

			With Feature Selection				
			Accuracy	Precision	Recall	F1-Score	AUC
DT	Unbalanced		0.9949	0.9765	0.9701	0.9730	**0.99843**
	Balanced		0.9847	0.9850	0.9847	0.9847	0.99821
NB	Unbalanced		0.9512	0.8294	0.8453	0.8082	0.99513
	Balanced		0.8933	0.9054	0.8933	0.8876	0.98735
KNN	Unbalanced		**0.9968**	0.9815	**0.9900**	0.9855	0.99775
	Balanced		0.9887	**0.9887**	0.9887	**0.9887**	0.99370
SVM	Linear	Unbalanced	0.9834	0.9629	0.9195	0.9360	-
		Balanced	0.9628	0.9643	0.9628	0.9627	-
	Non-Linear	Unbalanced	0.9941	0.9882	0.9525	0.9679	0.99571
		Balanced	0.9800	0.9803	0.9800	0.9799	0.99824

Conclusion

In conclusion, we examined the performance of models such as DT, NB, KNN and SVMs on classifying attack and normal patterns within the RT-IoT2022 dataset. Since the dataset was unbalanced, oversampling and undersampling techniques such as ClusterCentroids and SMOTE were used in order to make the dataset balanced. Next, in order to reduce the dimensionality of our data, we performed SelectPercentile on our dataset for each model with k changing for each model to achieve optimal results. Subsequently, we reported all results with and without feature selection for both balanced and unbalanced dataset for each model. The best result achieved was for balanced KNN fitted on 75% of features. The model achieved the highest F1-score of 0.9887.

References

[1] R. N. B. S., "RT-IoT2022." [object Object], 2023. doi: 10.24432/C5P338.

[2] B. S. Sharmila and R. Nagapadma, "Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using RT-IoT2022 dataset," *Cybersecurity*, vol. 6, no. 1, p. 41, Sep. 2023, doi: 10.1186/s42400-023-00178-5.

[3] J. McHugh, "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000, doi: 10.1145/382912.382923.

[4] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. doi: 10.5220/0006639801080116.

[5] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov. 2015, pp. 1–6. doi: 10.1109/MilCIS.2015.7348942.

[6] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, Nov. 2019, doi: 10.1016/j.future.2019.05.041.

[7] T. Saba, A. Rehman, T. Sadad, H. Kolivand, and S. A. Bahaj, "Anomaly-based intrusion detection system for IoT networks through deep learning model," *Computers and Electrical Engineering*, vol. 99, p. 107810, Apr. 2022, doi: 10.1016/j.compeleceng.2022.107810.

[8] "Network Intrusion Detection." Accessed: Mar. 22, 2024. [Online]. Available: https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection

[9] I. Ortega-Fernandez, M. Sestelo, J. C. Burguillo, and C. Piñón-Blanco, "Network intrusion detection system for DDoS attacks in ICS using deep autoencoders," *Wireless Netw*, Jan. 2023, doi: 10.1007/s11276-022-03214-3.

[10] H. Hindy, E. Bayne, M. Bures, R. Atkinson, C. Tachtatzis, and X. Bellekens, "Machine Learning Based IoT Intrusion Detection System: An MQTT Case Study (MQTT-IoT-IDS2020 Dataset)," in *Selected Papers from the 12th International Networking Conference*, B. Ghita and S. Shiaeles, Eds., Cham: Springer International Publishing, 2021, pp. 73–84. doi: 10.1007/978-3-030-64758-2_6.

[11] M. Bhavsar, K. Roy, Z. Liu, J. Kelly, and B. Gokaraju, "Intrusion-Based Attack Detection Using Machine Learning Techniques for Connected Autonomous Vehicle," in *Advances and Trends in Artificial Intelligence. Theory and Practices in Artificial Intelligence*, H. Fujita, P. Fournier-Viger, M. Ali, and Y. Wang, Eds., Cham: Springer International Publishing, 2022, pp. 505–515. doi: 10.1007/978-3-031-08530-7_43.

[12] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Jul. 2009, pp. 1–6. doi: 10.1109/CISDA.2009.5356528.

[13] "KDD Cup 1999 Data." Accessed: Mar. 22, 2024. [Online]. Available: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[14] M. Bhavsar, K. Roy, J. Kelly, and O. Olusola, “Anomaly-based intrusion detection system for IoT application,” *Discov Internet Things*, vol. 3, no. 1, p. 5, May 2023, doi: 10.1007/s43926-023-00034-5.
[15] I. Ullah and Q. H. Mahmoud, “A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks,” in *Advances in Artificial Intelligence*, C. Goutte and X. Zhu, Eds., Cham: Springer International Publishing, 2020, pp. 508–520. doi: 10.1007/978-3-030-47358-7_52.
[16] D. Xu, L. Liu, T. Li, X. Jing, and D. Gao, “Influence of driving conditions on the emission characteristics of China VI heavy-duty vehicles,” EDP Sciences, 2022. doi: 10.1051/e3sconf/202235203034.
[17] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, and A. Razaque, “Deep recurrent neural network for IoT intrusion detection system,” *Simulation Modelling Practice and Theory*, vol. 101, p. 102031, May 2020, doi: 10.1016/j.simpat.2019.102031.
[18] A. A. Alsulami, Q. Abu Al-Haija, A. Tayeb, and A. Alqahtani, “An Intrusion Detection and Classification System for IoT Traffic with Improved Data Engineering,” *Applied Sciences*, vol. 12, no. 23, Art. no. 23, Jan. 2022, doi: 10.3390/app122312336.
[19] S. Strecker, R. Dave, N. Siddiqui, and N. Seliya, “A Modern Analysis of Aging Machine Learning Based IoT Cybersecurity Methods.” arXiv, Oct. 14, 2021. doi: 10.48550/arXiv.2110.07832.
[20] S. Garcia, A. Parmisano, and M. J. Erquiaga, “IoT-23: A labeled dataset with malicious and benign IoT network traffic.” Zenodo, Jan. 20, 2020. doi: 10.5281/zenodo.4743746.
[21] M. Saharkhizan, A. Azmoodeh, A. Dehghantanha, K.-K. R. Choo, and R. M. Parizi, “An Ensemble of Deep Recurrent Neural Networks for Detecting IoT Cyber Attacks Using Network Traffic,” *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8852–8859, Sep. 2020, doi: 10.1109/JIOT.2020.2996425.
[22] I. Frazão, P. H. Abreu, T. Cruz, H. Araújo, and P. Simões, “Denial of Service Attacks: Detecting the Frailties of Machine Learning Algorithms in the Classification Process,” in *Critical Information Infrastructures Security*, E. Luiijf, I. Žutautaitė, and B. M. Hämmerli, Eds., Cham: Springer International Publishing, 2019, pp. 230–235. doi: 10.1007/978-3-030-05849-4_19.
[23] G. Zachos, I. Essop, G. Mantas, K. Porfyrakis, J. C. Ribeiro, and J. Rodriguez, “An Anomaly-Based Intrusion Detection System for Internet of Medical Things Networks,” *Electronics*, vol. 10, no. 21, Art. no. 21, Jan. 2021, doi: 10.3390/electronics10212562.
[24] N. Moustafa, “ToN_IoT datasets.” IEEE, Oct. 16, 2019. Accessed: Mar. 22, 2024. [Online]. Available: https://ieee-dataport.org/documents/toniot-datasets
[25] I. Essop, J. C. Ribeiro, M. Papaioannou, G. Zachos, G. Mantas, and J. Rodriguez, “Generating Datasets for Anomaly-Based Intrusion Detection Systems in IoT and Industrial IoT Networks,” *Sensors*, vol. 21, no. 4, Art. no. 4, Jan. 2021, doi: 10.3390/s21041528.
[26] Nidhi, “Decision Trees: A Powerful Tool in Machine Learning,” Medium. Accessed: Mar. 22, 2024. [Online]. Available: https://medium.com/@nidhigh/decision-trees-a-powerful-tool-in-machine-learning-dd0724dad4b6