A Review of Machine Learning Methodologies in Identifying Attacks on IoT Networks

Oscar Olaya oolaya@udel.edu
Saeid Rajabi srajabi@udel.edu
Sydeny Hester sydneyph@udel.edu
Hari Chandana Palnati chandana@udel.edu

University of Delaware
Newark, Delaware, USA

Abstract

As new smart devices connect to the internet, IoT networks continue to grow in both size and complexity. With this development, the need for security against cyber attacks becomes increasingly critical. Detection of malicious network traffic is a necessary step in ensuring devastating attacks are detected and blocked before true damage is done. Today, companies are looking to integrate Machine Learning into detection systems to make them faster and more accurate. In this paper, we examine a variety of established Machine Learning models to determine which are the most promising for malicious package detection on IoT networks. Using a dataset containing 257,000 network traffic samples, we identified Multi-Layer Perception and Decision Trees as the two most promising candidates with 94.912% and 94.12% identification accuracy respectively.

1. Introduction

An IoT network refers to a network of physical devices that have a combination of sensors, software and an ability to connect to networks. These features allow smart devices to collect and share data with one another, providing users a wide range of services [1]. Notable examples of IoT devices include self driving cars, smart fridges and health monitors.

With the rise of IoT networks, there have been many tangible benefits, but has posed new security risks and challenges. There have been a multitude research publications outlining security challenges associated with IoT networks [2] [3] [4] [5]. Oftentimes, these attacks can be detected by examining package traffic on the network [6].

With the development of machine learning, many are looking for ways to train models to detect malicious activity on IoT networks. Due to varying goals of machine learning models, only a select few will be a good fit for our task. In order for machine learning to be effective, we need to research and discover which models are the best fit.

In this study, we examine the performance of a variety of machine learning models to deduce which would be best for malicious package detection on IoT networks. Our dataset, as published in the following papers [7] [8] [9] [10] [11], contains the raw data from IoT network packages. Using this data, we trained a variety of machine learning models and compared their results.

2. Related Works

In this section, we will review research papers similar to our own study and then discuss machine learning models that will serve as the basis for our research.

2.1. Previous Research

The dataset we used in our research has been utilized in many other IoT studies. Past research topics have included general network analysis [12] [13], network forensics [14], as well as SCADA Defense [15]. Of the works we found, none had performed an empirical study of IoT attack detection using machine learning.

Notably, we located a Kaggle project that utilized this dataset to train a variety of Machine Learning models [16]. Our training was able to achieve similar results, providing further legitimacy to the results we achieved.

2.2. Logistic Regression

Logistic regression is a machine learning algorithm used for binary classification tasks that uses supervised learning to adjust to a given dataset. The output of a logistic regression model is the probability that an input sample belongs to a specific class. The logistic regression model can be boiled down to a linear regression model whose output is fed into a Sigmoid function to generate a probability. The equation describing a logistic regression model can be seen in the equation 1. Although logistic regression is a simple

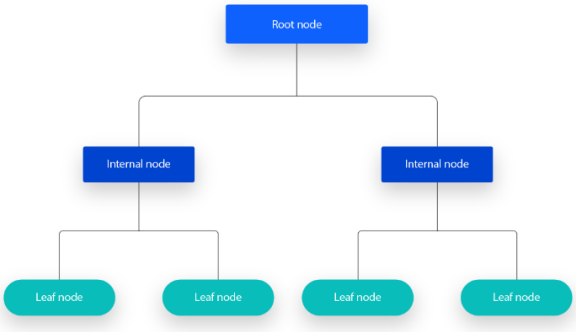


Figure 1. A figure representing a decision tree, sourced from reference[17]

model, it is very useful for binary classification tasks. One of the key advantages of this model is that its level of explainability is high, and then it can be used to get insights into the features and perform feature selection.

$$f_\theta(x) = \sigma(\sum_{i=0}^{d} \theta_i x_i) \tag{1}$$

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

2.3. Decision Tree

A decision tree is an unsupervised learning algorithm used for both classification and regression problems [18]. A decision tree is structured as a hierarchy of nodes connected with edges (commonly referred to as branches) [17]. There are 3 types of nodes found in a decision tree.

- **Root Node:** The node that is at the top of the hierarchy. It has no incoming branches.
- **Internal Node:** A node that has both an incoming branch and an outgoing branch.
- **Leaf Node:** A node that has an incoming branch and no outgoing branches

When attempting to make a classification, each node examines a subset of inputted features to determine whether it can predict a classification or it needs to be sent to another node for further parsing. Figure 1 provides an example of a decision tree.

2.4. Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a class of feedforward neural networks widely used for supervised learning tasks such as classification and regression. It contains layers of neurons, each of which connects to every neuron in the subsequent layer. This feature is referred to as a fully connected neural network (FCNN). FCCNs allow MLPs to model complex, non-linear relationships in data. We can summarize the components of an MLP model structure as follows.

Input layer. The input layer receives the raw data features and each neuron represents one feature.

Hidden layer. Hidden layers between the input and output layers transform the input data using a linear transformation followed by a non-linear activation function such as ReLU or sigmoid.

Output layer. The output layer produces the final predictions. For classification tasks, the output layer often uses a softmax function to convert logits into probabilities.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2}$$

As shown in Eq. 2,

- $\sigma(z)_i$: The output of the Softmax function for the i-th class. It represents the probability assigned to the i-th class.
- z_i: The input score (logit) for the i-th class, which is the raw output from the previous layer of the neural network.
- $\sum_{j=1}^{n} e^{z_j}$: The normalization term. It sums the exponentials of the input scores over all n classes to ensure that the outputs are valid probabilities that sum to 1.
- n: The total number of classes in the classification problem.

The Softmax equation ensures that the sum of the probabilities assigned to all classes is equal to 1. This property makes Softmax particularly useful for classification tasks where the model's output represents a probability distribution over multiple classes.

Regularization: Techniques such as dropout and L2 regularization prevent overfitting. The dropout randomly deactivates a percentage of neurons during training, which reduces the dependence on specific neurons.

Loss function: Cross-entropy (CE) loss is the most common loss function used for classification tasks. CE is used on the model after the softmax activation layer. It calculates the difference between an input-given prediction probability distribution and the true distribution from the ground truth dataset.

Since the CE loss directly optimizes the log-likelihood of the correct class, it is preferable to Mean-Squared Error (MSE). MSE is unsuitable due to vanishing gradients, which slow learning as predictions approach actual values, making it challenging for the model to update its weights, particularly in deeper networks.

The general form of CE loss is as follows, where C is the total number of classes, N is the number of datapoints

in the dataset, y_{ij} is ground truth label for the datapoint and $\hat{y_{ij}}$ is the predicted probability for class j of sample i.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) \tag{3}$$

To be more specific, the CE loss specifically for binary classification can be derived in the following format.

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{4}$$

2.5. SVM

Support Vector Machines (SVM) are among the most powerful and versatile supervised learning algorithms widely used in the fields of classification, regression, and outlier detection. The algorithm has gained significant attention due to its robustness and effectiveness across a variety of domains, including image recognition and text categorization.

The core idea of SVM is to find the optimal hyperplane that best separates data points from different classes in a feature space. This hyperplane is determined by maximizing the margin, which is the distance between it and the closest data point from any class. These data points are called support vectors and play a crucial role in defining the decision boundary.

The key components of SVM are as follows.

- **Hyperplane**: The hyperplane is the decision boundary that separates data points into different classes. In a two-dimensional space, it is a line, while in higher dimensions, it becomes a flat affine subspace.The hyperplane is defined by the equation: $\mathbf{w}\cdot\mathbf{x} + b = 0$, where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the feature vector, and b is the bias.

- **Support Vectors**: These are the critical data points closest to the hyperplane. They influence the orientation and position of the hyperplane. Support vectors are the only metric used to define the margin, making them essential to the model's training.

- **Margin**: The margin is the distance between the hyperplane and the nearest data points (support vectors) from each class. SVM aims to maximize this margin to improve generalization and reduce overfitting.

- **Slack variable** (ξ_i): In the soft-margin SVM, slack variables are introduced to handle misclassified points and allow some flexibility. The slack variable, ξ_i measures the degree of misclassifications for each data point.

- **Kernel Function**: The kernel function maps input data into a higher-dimensional space to make it linearly separable. Common kernel functions include:

  Linear Kernel, Polynomial Kernel, Radial Basis Function (RBF) Kernel, Sigmoid Kernel

- **Optimization objective**:The goal of SVM is to solve the following optimization problem to find the hyperplane with the maximum margin:

$$\min_{w,b}\frac{1}{2}\|w\|^2 \tag{5}$$

  Subject to constraints:

$$y_i(w\cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{6}$$

  Where C controls the trade-off between maximizing the margin and minimizing classification errors.

For non-linearly separable data, a slack variable is introduced to permit some misclassification. This leads to the soft-margin SVM formulation, in which the regularization parameter controls the trade-off between maximizing the margin and minimizing classification errors.

Real-world datasets are often not linearly separable. To address this, SVM employs kernel functions to map the data into a higher-dimensional space where a linear separation becomes possible. This process is known as the kernel trick and allows SVM to handle complex decision boundaries efficiently.

3. Methodology

This section outlines the machine learning methodologies used to classify IoT network traffic as malicious or benign. The process involves pre-processing raw data then delves into the mechanics used to implement individual models.

3.1. Data Processing

This section describes the steps taken to process the dataset from [19]. The dataset contains more than 257,000 network traffic samples labeled as malicious or benign. The data was classified using 49 features. Examples include flow features (*e.g.*, source IP, destination IP, source port number, protocol and etc.), basic features (*e.g.*, source to destination bytes), content features (*e.g.*, destination TCP window) and time features (*e.g.*, start-time, end-time)

In order to train classification models effectively, it is critical to consider the quality of the data used for training and testing. This meant our first step was to pre-process the original dataset. This would allow us to remove irrelevant features, clean the dataset of undefined or incorrect values,

encode the qualitative features into numbers, and normalize all features, ensuring that each contributes proportionally. In the following paragraphs, we describe all the steps performed as part of the pre-processing.

- **Remove irrelevant and undefined features**: In this step, we examined the meaning and purpose of each feature. This helped in determining whether or not the feature would be useful for training our model. Based on our analysis, the features *"id"* and *"rate"* were removed. The first feature, *"id"*, was removed because it was simply an index number. The second feature, *"rate"*, was removed because it was not defined in the documentation, making it impossible to determine whether or not it would contribute to a solution.

  Our next step was to complete another analysis to determine how many values populated each feature. The feature *"service"* was removed because 53% of its values were undefined. A data point with undefined value for the *"state"* feature was removed from the set entirely. The feature *"ct_ftp_cmd"* was removed because it held the same values as the feature *"is_ftp_login"*.

- **Remove wrong values**: When completing our analysis, we identified several features with values outside of the range of defined values. We identified features specified that required binary inputs (0 or 1), but located around 50 data points populated with 2 or 4. These data points were removed. Their absence was negligible in the context of our dataset.

- **Encoding qualitative features**: In order to train our models, we needed to encode the nominal or qualitative features into numbers. For each qualitative feature, an encoding process was performed. The process consisted of assigning an integer value to each of the possible qualitative values and updating the dataset with these numbers.

- **Normalization**: To ensure that different features would contribute proportionally during the training process, we applied normalization. In this project, we used the minmaxscaling technique. This method was selected because the features collected from the IoT traffic don't necessarily have a Gaussian distribution. This meant methods that assume normality were not well suited for our problem.

Because our goal is to classify network traffic, it is important to balance the dataset between benign and attack traffic. After analyzing our dataset, as shown in Fig. 2, we discovered that the initial dataset comprised of 63.9% benign traffic and 36.1% attack traffic.

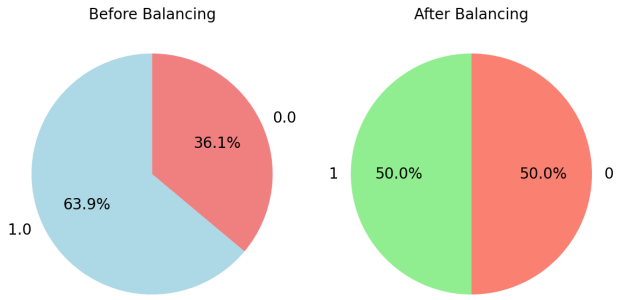


Figure 2. Class balancing before VS. after applying balancing technique

To address this imbalance, techniques we utilized techniques such as random oversampling, random oversampling, and SMOTE (Synthetic Minority Oversampling Technique). Using the SMOTE method effectively created synthetic samples of the minority class, balancing the dataset while preserving the original feature distribution. As shown on the right-hand side of Fig. 2, we were successful in balancing the number of samples in each class.

3.1.1 Principal component analysis (PCA)

With the intention of restricting the number of features to only the most relevant, we applied Principal Component Analysis (PCA) to the cleaned dataset. After obtaining the Eigenvalues, we analyzed them, identifying the Proportion of Variance Explained (PVE). We found that by using just 11 of our features we achieved 95.77% PVE as shown in Figure 3. Considering this result, we decided to perform a study where we trained a logistic regression model with only the 11 features selected using PCA , then compare it to a model trained using all the features. The results of this experiment are shown in the logistic regression Section 4.

```
Features to keep:
Index(['dur', 'proto', 'state', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'sttl',
       'dttl', 'sload', 'dload'],
      dtype='object')

Features to drop:
Index(['sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb',
       'dtcpb', 'dwin', 'tcprtt', 'synack', 'ackdat', 'smean', 'dmean',
       'trans_depth', 'response_body_len', 'ct_srv_src', 'ct_state_ttl',
       'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',
       'is_ftp_login', 'ct_flw_http_mthd', 'ct_src_ltm', 'ct_srv_dst',
       'is_sm_ips_ports'],
      dtype='object')

Percentage of the sum of Eigen values that is kept is: 95.7690252633386
```

Figure 3. Feature selection with PCA.

3.2. Models

3.2.1 Logistic Regression

The logistic regression model was implemented using Pytorch. The model was built using a linear layer followed

by a sigmoid function. For the loss function, Binary cross-entropy was used. During our training process, the training dataset was divided into smaller batches to update the model frequently and achieve faster convergence.

Before testing different logistic regression models, we tested the PCA results by training identical logistic regression models; one with all features and another with only the PCA selected features, comparing their accuracies. Considering the high level of explainability of the logistic regressions model, we utilized the results of the model trained with all the features to perform another feature selection process based on the weights of the model.

After selecting the most relevant features, we trained multiple models to find the best combination of learning rate, optimizer and features. Two optimizers (i.e. SGD and Adam) and three different learning rates (i.e. 0.001, 0.01, 0.1) were each tested. The results of the described methodology are shown in section 4.

3.2.2 Decision Tree

The decision tree model was implemented using the sklearn module for decision trees[20]. This software library provides the code needed to create a tree object as well as all functions needed to train and test the model. The model was constructed using the DecisionTreeClassifier() function and was trained by inserting data into the fit() function.

When training this model, all features present in the dataset were included. This is because the model took less than 1 second to train and there was no noticeable effect on accuracy. Data was split into training and testing sets using sklearn's train_test_split() function. This allowed us to easily generate a wide range of training and testing splits.

Testing consisted of using the predict() function and then comparing the predicted results to the originals. The results produced by the methodology are shown in Section 4.

3.2.3 Multilayer Perceptron

The MLP model was implemented using PyTorch and structured with an input layer, two hidden layers, and an output layer. The input layer accepts the feature vector for each sample, with the size dynamically set to match the number of input features. The first hidden layer consists of 128 neurons, followed by a ReLU activation function and a dropout layer with a rate of 0.3. The second hidden layer contains 64 neurons, also followed by a ReLU activation function and a dropout layer with a rate of 0.3. The output layer consists of 2 neurons corresponding to the two output classes. It uses a softmax activation function to output class probabilities.

We trained the model using the Adam optimizer with a learning rate of 0.001 and implemented the Cross-Entropy Loss function. The training was executed over 20 epochs with a batch size of 64. The PyTorch DataLoader was used for batch processing. During this process, the training data was shuffled to ensure complete learning.

ID	#H-Layer1	#H-Layer2	DR	LR
1	64	32	0.2	0.001
2	64	32	0.2	0.01
3	64	32	0.3	0.001
4	64	32	0.3	0.01
5	64	64	0.2	0.001
6	64	64	0.2	0.01
7	64	64	0.3	0.001
8	64	64	0.3	0.01
9	128	32	0.2	0.001
10	128	32	0.2	0.01
11	128	32	0.3	0.001
12	128	32	0.3	0.01
13	128	64	0.2	0.001
14	128	64	0.2	0.01
15	128	64	0.3	0.001
16	128	64	0.3	0.01

Table 1. Hyperparameter Configurations for Grid Search

To achieve optimal performance in classifying malicious and benign traffic, a grid search was conducted to explore various hyperparameter configurations. The goal of this exercise was to identify the best combination of hyperparameters that minimized loss and maximized accuracy on the testing dataset.

Grid search was performed over the following hyperparameters:

- Hidden Layer 1 Size: [64, 128]
- Hidden Layer 2 Size: [32, 64]
- Dropout Rate: [0.2, 0.3]
- Learning Rate: [0.001, 0.01]

The results from the hyperparameter configuration are displayed in Table 1.

3.2.4 MLP using the selected features

Considering the results obtained with the feature selection process using the logistic regression model, we decided to train an MLP model using only the 23 selected features. This would create a lighter model that could be compared with the MLP using all the features. To make the model even lighter, we elected to reduce the number of neurons in the hidden layers to 32.

3.2.5 SVM

The Support Vector Machine was implemented using Scikit-learn. Initially, we used all 39 features to train the

```
linear.weight tensor([[  2.3649,  16.2841,  -9.8615,   1.6511,  -0.2129,   1.3248,  -0.5436,
           0.7496,  11.1496,  -3.7827,  -5.4211,   1.8608,  -0.3563,  -4.1961,
          -0.4300,  -0.9423,   0.4033, -10.1055,   0.0237,   0.0183,  -0.2111,
          -8.2204,  -7.7824,  -1.6797,   0.4641,   8.5088,   0.9336,  -0.3792,
          -1.0759,  10.2377,  -0.2962,   4.2880,  15.1846,  -2.5759,   1.5659,
          -1.7993,  -0.4378,  -1.6498,  -5.3379]])
linear.bias tensor([-1.5334])

Features to keep:
Index(['dur', 'proto', 'state', 'spkts', 'sbytes', 'dttl', 'sload', 'dload',
       'sloss', 'sinpkt', 'swin', 'tcprtt', 'synack', 'ackdat', 'dmean',
       'ct_state_ttl', 'ct_src_dport_ltm', 'ct_dst_sport_ltm',
       'ct_dst_src_ltm', 'is_ftp_login', 'ct_flw_http_mthd', 'ct_srv_dst',
       'is_sm_ips_ports'],
      dtype='object')

Features to drop:
Index(['dpkts', 'dbytes', 'sttl', 'dloss', 'dinpkt', 'sjit', 'djit', 'stcpb',
       'dtcpb', 'dwin', 'smean', 'trans_depth', 'response_body_len',
       'ct_srv_src', 'ct_dst_ltm', 'ct_src_ltm'],
      dtype='object')

Percentage of the sum of Eigen values that is kept is: tensor(94.8182)
```

Figure 4. Feature selection with Logistic Regression.

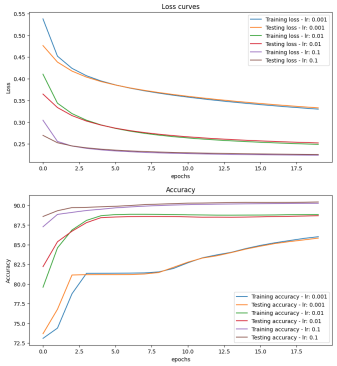


Figure 5. Loss and accuracies curves for Logistic regression using SGD and all the features.

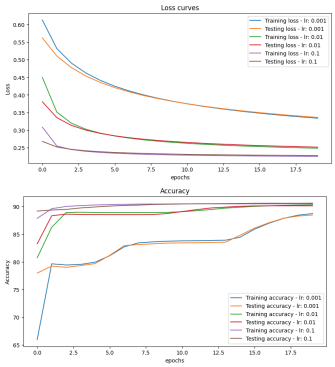


Figure 6. Loss and accuracies curves for Logistic regression using SGD and the selected features.

model. We trained multiple models with different combinations of hyperparameters to find the best results. We repeated the same training and testing process using the 23 selected features identified in our PCA analysis. The results of the described methodology are shown in Section 4.

4. Experiments and Results

4.1. Logistic Regression

The first step to test the logistic regression was to train the model using all the features. For this test, Stochastic Gradient Descent (SGD) was used as the optimizer, the learning rate was set to 0.01 and the cross-entropy loss function was used. We obtained an accuracy of around 88% when testing our model.

After training our standard model, we then trained a model using the 11 features identified by PCA. Surprisingly, this model had a testing accuracy around 81%, results are shown in Table 2. It is clear that using the 11 features selected with PCA resulted in a considerable drop of the testing accuracy. This means that more features are needed to train an effective model.

Table 2. Training and test accuracies with and without PCA selected features.

# of features	Training accuracy	Testing accuracy
39 (All)	**88.843**	**88.688**
11 (PCA)	83.067	81.351

Knowing the model trained with the PCA selected features experienced a considerable accuracy drop, we decided to perform another feature selection process. We accomplished this by taking advantage of the high level of explainability that the logistic regression model provides. To find our new features, we examined the magnitude of the parameters in the model that was trained using all the features. We then analyzed them to identify the ones whose magnitudes were close to zero. After obtaining the percentage contribution of each parameter magnitude to the overall sum of the magnitudes, we found that by using 23 features, it was possible to preserve around 94% of the parameters' contribution. These results are shown in Figure 4.

Table 3. Comparing different optimizers and learning rates for logistic regression.

# of features	Optimizer	Learning rate	Training accuracy	Testing accuracy	F1 score
39	SGD	0.001	86.017	85.841	85.557
39	SGD	0.01	88.843	88.688	88.311
39	SGD	0.1	90.263	**90.433**	**90.25**
23	SGD	0.001	88.706	88.44	87.839
23	SGD	0.01	90.284	90.105	89.965
23	SGD	0.1	90.468	90.59	90.439
23	Adam	0.001	90.157	90.111	89.85
23	Adam	0.01	90.333	90.413	90.256
23	Adam	0.1	90.255	90.408	90.183

Using the 23 newly selected features, we trained a logistic regression model. This model obtained a very similar accuracy to the model using all the features. We concluded that the updated feature selection process was successful, meaning the selected features were highly representative of the dataset and were well suited to solve the problem.

In order to find the optimal combination of optimizer (i.e. SGD and Adam) and learning rate, Multiple models were trained. The results of this test can be seen in the table 3. The best results were obtained using SGD and the highest learning rate. This model possessed both a fast convergence and a higher accuracy. The learning curves of the models were analyzed to ensure there were no indicators of overfitting. The loss and accuracy curves for the tests using SGD are shown in Figures 5 and 6.

4.2. Decision Tree

Following the methodology outlined in Section 3.2.2, we split our data into a 80%-20% testing to training ratio. After feeding the training data into the proper sklearn functions, the decision tree was calculated in less than 1 second.

Our trained Decision Tree achieved 94.12% accuracy and an F1 score of 95.0. Figure 7 is a confusion matrix visu-

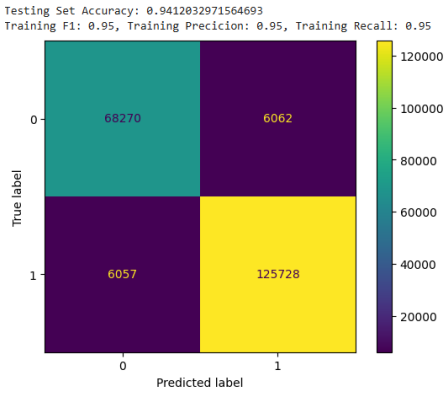


Figure 7. Confusion Matrix displaying the results from out 80-20 data split

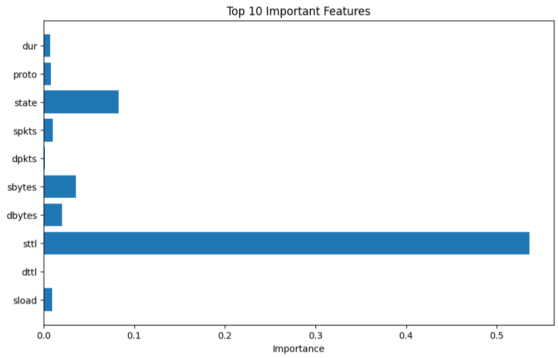


Figure 8. Top 10 most important features when training the decision tree model

alizing the results of our test. In total, we had 125,728 True Positivities, 68,270 True Negatives, 6,062 False Negatives and 6,057 False Positives. These results are very promising. It is likely that with a more finely tuned decision tree, better results could be achieved.

By examining the feature_importance_ attribute in sklearn, we leaned the most important features used in determining whether or not a pacakge was malicious. The top 3 features were ct_dst_src_ltm (7.4%), state (8.3%) and sttl (53.6%). Figure 8 is a bar graph visualizing the 10 most important features in constructing our decision tree. This information aligns with what we learned during our PCA analysis and will be useful in future research to identify malicious packages.

Because of how quickly a decision tree can be trained, we chose to run a experiment to see how effective a decision tree would be if it trained with less data. Table 4 catalogs the results of the different data splits. The results made it clear that even with a lack of training data, the decision tree model is capable of producing impressive results. Testing accuracy suffers less than 1% when the split goes from 80-20 to 90-10 and accuracy remains around 86.45 when there is a 99.9-0.1 training split.

Table 4. Performance metrics for different test/train splits.

Test/Train Split	Training Accuracy	Testing Accuracy	F1 Score	Precision	Recall
80/20	99.947	94.12	95.0	95.0	95.0
85/15	99.943	93.876	95.0	95.0	95.0
90/10	99.977	93.542	95.0	95.0	95.0
95/05	99.961	92.984	95.0	94.0	95.0
99/01	100.0	91.534	93.0	94.0	93.0
99.9/0.1	100.0	86.45*	89.0	92.0	87.0

These factors prove decision trees are an attractive choice for detecting malicious packages. This is because of the models high accuracy, short training times, and robustness in circumstances where the model is trained on small amounts of data.

Table 5. Results Summary of MLP Training for Various Hyperparameter Configurations

ID	#H-Layer1	#H-Layer2	DR	LR	Best Epoch	Test Loss	Test Accuracy (%)
1	64	32	0.2	0.001	20	0.1128	94.63
2	64	32	0.2	0.01	17	0.1186	94.37
3	64	32	0.3	0.001	19	0.1135	94.77
4	64	32	0.3	0.01	16	0.1211	94.28
5	64	64	0.2	0.001	18	0.1110	94.85
6	64	64	0.2	0.01	15	0.1287	94.42
7	64	64	0.3	0.001	15	0.1133	94.81
8	64	64	0.3	0.01	16	0.1292	94.29
9	128	32	0.2	0.001	20	0.1097	94.90
10	128	32	0.2	0.01	20	0.1237	94.37
11	128	32	0.3	0.001	18	0.1127	94.80
12	128	32	0.3	0.01	16	0.1253	94.15
13	128	64	0.2	0.001	18	**0.1080**	**95.01**
14	128	64	0.2	0.01	18	0.1273	94.35
15	128	64	0.3	0.001	18	0.1127	94.76
16	128	64	0.3	0.01	15	0.1373	94.13

4.3. MLP

By utilizing the methodology described in section 3.2.3, we performed a grid search which resulted in 16 unique configurations. Each configuration was trained for 20 epochs, and performance was evaluated in terms of training loss, testing loss, and testing accuracy. Our final model was chosen based on the configuration that performed the best. As shown in Table 5, the best configuration is 128 and 64, with a dropout ratio of 0.2, and Adam optimizer with a learning rate of 0.001.

Classification reports for the MLP models trained for 20 and 200 epochs are presented in Tables 6 and 7, respectively. These reports summarize the performance of the model in terms of precision, recall, F1-score, and accuracy for both classes: benign traffic (class 0) and malicious traffic (class 1).

20 Epochs: Table 6 shows the model performance on 20 epochs. The model achieved an overall accuracy of 94.72%. Class 0 had a precision of 91%, recall of 95%, and an F1-score of 93%, while class 1 achieved a precision of 97%, recall of 95%, and an F1-score of 96%.

200 Epochs: Table 7 shows the model performance on 200 epochs. With extended training, the model showed

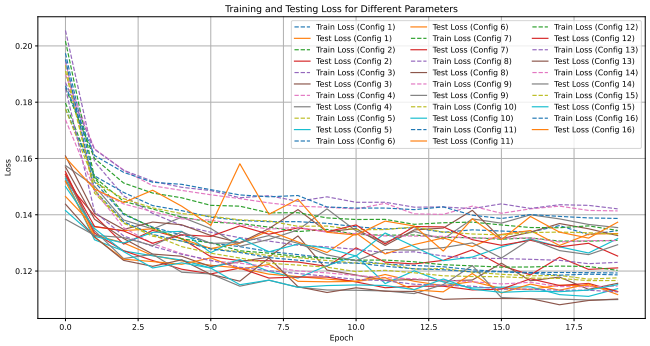


Figure 9. Training loss and testing loss, with different hyperparameters mentioned in the Table 1

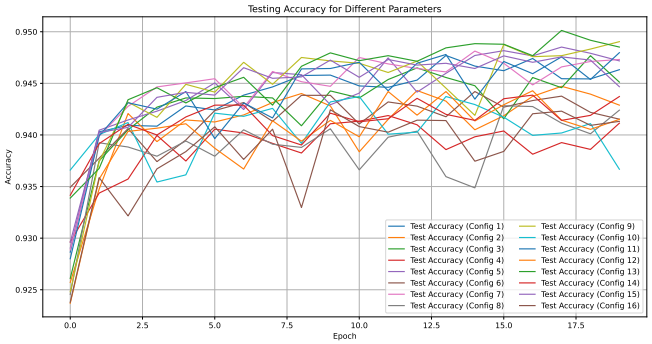


Figure 10. Testing accuracy, with different hyperparameters mentioned in the Table 1

Table 6. Classification Report for 20 Epochs

Class	Precision	Recall	F1-Score	Support
0	0.91	0.95	0.93	18601
1	0.97	0.95	0.96	32929
Accuracy		0.95 (51,530 samples)		
Macro Avg	0.94	0.95	0.94	51530
Weighted Avg	0.95	0.95	0.95	51530

Table 7. Classification Report for 200 Epochs

Class	Precision	Recall	F1-Score	Support
0	0.91	0.95	0.93	18601
1	0.97	0.95	0.96	32929
Accuracy		0.95 (51,530 samples)		
Macro Avg	0.94	0.95	0.95	51530
Weighted Avg	0.95	0.95	0.95	51530

a slight improvement, achieving an overall accuracy of 94.96%. Class 0 maintained similar metrics, while Class 1 showed consistent performance with no significant degradation or overfitting.

Both reports indicate the MLP model's robustness, with high precision and recall across classes. The macro and weighted averages further confirm the model's balanced performance on the dataset.

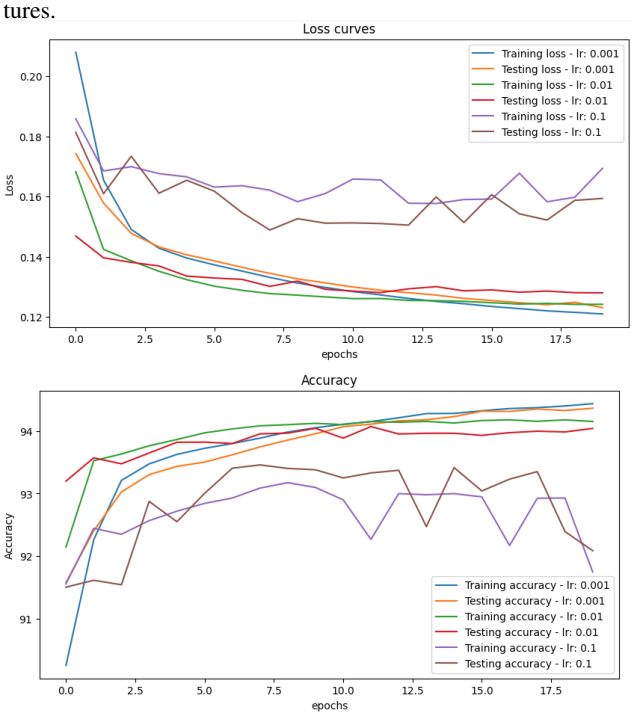
4.3.1 MLP using the selected features

With the intention of training an efficient and lighter model, an MLP was trained with the 23 features that resulted from the feature selection process. In this model, hidden layers of neurons were decreased to 32, and the Adam optimizer was selected. Because results obtained from the previous MPL models did not indicate an overfitting issue, the dropout layers were removed. The results that were obtained are shown in Table 8 and Figure 11.

Table 8. MLP using selected features.

# of features	Training accuracy	Testing accuracy	F1 score
23	94.433	94.361	94.336

Figure 11. Loss and accuracy curves for MLP with selected features.



These results show that is possible to train a lighter MLP model using the selected features and obtain an accuracy around 1% lower compared to the accuracy obtained using a bigger MLP with all the features. This makes the MLP model more feasible for users who lack significant computational resources.

4.4. SVM

To find the optimal SVM model, our first step was to train the model with a variety of hyper-parameters to find the optimal combination. The best performing combination was : **kernel**='rbf', **C**=1, **gamma**='scale'.The results obtained after training the SVM model with the optimal com-

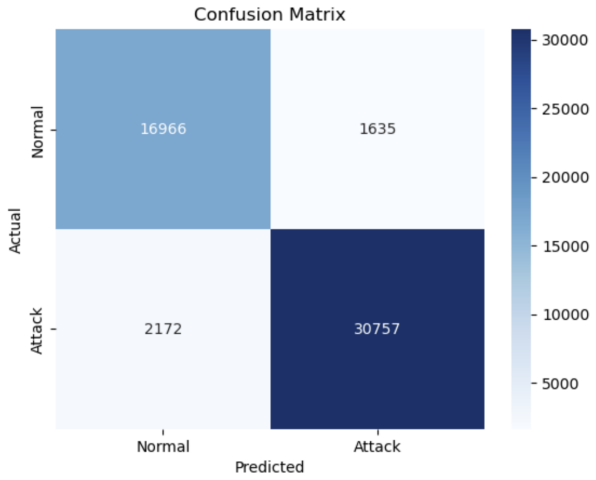


Figure 12. Confusion matrix with classes 'Attack' and 'Normal'

Figure 13. Training vs Testing accuracy for incremental sizes (from 10% to 100% of training data)

Table 9. Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.89	0.91	0.90	18601
1	0.95	0.93	0.94	32929
Accuracy		0.93		
Macro Avg	0.92	0.92	0.92	51530
Weighted Avg	0.93	0.93	0.93	51530

bination of hyperparameters are shown in Table 9. The testing and training accuracies obtained are 92.61% and 92.48%, respectively.

Figure 12 shows the confusion matrix which indicates the following:

- **Top-left cell (16966):** These are the instances where the actual class is "Normal" and the model correctly predicted "Normal" (True Positives for "Normal").
- **Top-right cell (1635):** These are the instances where the actual class is "Normal," but the model incorrectly predicted "Attack" (False Positives).
- **Bottom-left cell (2172):** These are the instances where the actual class is "Attack," but the model incorrectly predicted "Normal" (False Negatives).
- **Bottom-right cell (30757):** These are the instances where the actual class is "Attack" and the model correctly predicted "Attack" (True Positives for "Attack").

The model was trained with incremental sizes of the training data set from 10% to 100%. Figure 13 plotted their training vs testing accuracies. As per the plot, both training and testing accuracy improve as more data is added to the training set, showing that the model benefits from additional data. The gap between training and testing accuracy is minimal throughout, which suggests that the model generalizes well without significant overfitting or underfitting.

At 100% training data size, the training accuracy decreases slightly, which could result from the inclusion of difficult-to-learn examples or noise in the training set.

5. Conclusions

As the number of smart devices connecting to the Internet grows, the importance of monitoring IoT networks for cyber threats will be further emphasized. Trained machine learning models will be useful in detecting malicious packages traveling through an IoT network. If implemented correctly, they can identify and block malicious activity more quickly than traditional methods.

In our research, we examined a variety of machine learning techniques in an attempt to discover which would be most effective at detecting IoT threats. We found that the Multi-Layer Perception and the Decision Tree were the two most promising candidates, with 94.912% and 94.12% identification accuracy, respectively.

References

[1] IBM. (2024) What is the iot?? [Online]. Available: https://www.ibm.com/topics/internet-of-things

[2] J. Mohanty, S. Mishra, S. Patra, B. Pati, and C. R. Panigrahi, "Iot security, challenges, and solutions: A review," in *Progress in Advanced Computing and Intelligent Engineering*, C. R. Panigrahi, B. Pati, P. Mohapatra, R. Buyya, and K.-C. Li, Eds. Singapore: Springer Singapore, 2021, pp. 493–504.

[3] K. Shaukat, T. M. Alam, I. A. Hameed, W. A. Khan, N. Abbas, and S. Luo, "A review on security chal-

lenges in internet of things (iot)," in *2021 26th International Conference on Automation and Computing (ICAC)*, 2021, pp. 1–6.

[4] A. Bhattacharjya, X. Zhong, J. Wang, and X. Li, *Security Challenges and Concerns of Internet of Things (IoT)*. Cham: Springer International Publishing, 2019, pp. 153–185. [Online]. Available: https://doi.org/10.1007/978-3-319-92564-6_7

[5] M. Azrour, J. Mabrouki, A. Guezzaz, and A. Kanwal, "Internet of things security: Challenges and key issues," *Security and Communication Networks*, vol. 2021, no. 1, p. 5533843, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/5533843

[6] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, 2017.

[7] 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)*, 2015, pp. 1–6.

[8] ——, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016. [Online]. Available: https://doi.org/10.1080/19393555.2015.1125974

[9] N. Moustafa, J. Slay, and G. Creech, "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks," *IEEE Transactions on Big Data*, vol. 5, no. 4, pp. 481–494, 2019.

[10] N. Moustafa, G. Creech, and J. Slay, *Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models*. Cham: Springer International Publishing, 2017, pp. 127–156. [Online]. Available: https://doi.org/10.1007/978-3-319-59439-2_5

[11] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," *CoRR*, vol. abs/2011.09144, 2020. [Online]. Available: https://arxiv.org/abs/2011.09144

[12] N. Moustafa, B. Turnbull, and K.-K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4815–4830, 2019.

[13] N. Moustafa, G. Creech, and J. Slay, "Flow aggregator module for analysing network traffic," in *Progress in Computing, Analytics and Networking*, P. K. Pattnaik, S. S. Rautaray, H. Das, and J. Nayak, Eds. Singapore: Springer Singapore, 2018, pp. 19–29.

[14] N. Koroniotis, N. Moustafa, E. Sitnikova, and J. Slay, "Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques," in *Mobile Networks and Management*, J. Hu, I. Khalil, Z. Tari, and S. Wen, Eds. Cham: Springer International Publishing, 2018, pp. 30–44.

[15] M. Keshk, N. Moustafa, E. Sitnikova, and G. Creech, "Privacy preservation intrusion detection technique for scada systems," in *2017 Military Communications and Information Systems Conference (MilCIS)*, 2017, pp. 1–6.

[16] C. Kirstein. (2022) Unsw-nb15 modelling - 97.7. [Online]. Available: https://www.kaggle.com/code/carlkirstein/unsw-nb15-modelling-97-7

[17] L. Rokach and O. Maimon, *The Data Mining and Knowledge Discovery Handbook*. Springer, 2009, ch. Chapter 9: Decision Trees, pp. 165–192.

[18] IBM. (2024) What is a decision tree? [Online]. Available: https://www.ibm.com/topics/decision-trees

[19] 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)*. IEEE, 2015, pp. 1–6.

[20] 1.10. decision trees. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html