Identification and Classification of Malicious Traffic within Network Intrusion Detection System

Annabelle Crescenzo
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ, USA
acrescen@stevens.edu

Skye Jorgensen
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ, USA
sjorgen1@stevens.edu

Angel Ordonez Retamar
Department of Computer Engineering
Stevens Institute of Technology
Hoboken, NJ, USA
aordonez@stevens.edu

Abstract—**Cybersecurity is one of the fastest growing fields in the world. Due to the sophisticated levels of malicious attacks, sophisticated methods of detecting these attacks are constantly being developed. Similar to cybersecurity, Machine Learning (ML) is constantly evolving and making advancements. ML makes predictions or derives insights based on data. Since malicious attacks constantly occur in cybersecurity, there is an endless amount of data to train these models. By combining the knowledge and purpose of cybersecurity with the computational power of ML, the idea of traditional cybersecurity is changing forever. This study aims to evaluate the performance of supervised binary and multiclass classifications models, such as Logistic Regression, Support Vector Machines (SVM), Decision Trees, and Artificial Neural Networks (ANN), for identifying cyberattacks in Network Intrusion Detection Systems (NIDS). Binary models distinguish between benign and anomaly events, while multiclass models distinguish between benign and specific cyberattacks: DDoS, PortScan, and Bot. For further analysis, unsupervised learning through K-Means Clustering was implemented to simulate real-world scenarios where labeled data is unavailable. While supervised models achieved high overall accuracy and recalls, the models demonstrated sensitivity to class imbalance, especially in detecting rare attack types. Similarly, the unsupervised clustering showed high accuracy for majority classes, but struggled with detecting uncommon network traffic. We also analyzed the information gain of high-dimensional features and selected only the most critical attributes to simplify model implementation. This study demonstrates the use of various models to build effective intrusion detection systems.**

I. Introduction

In today's technological landscape, cybersecurity is an ever-evolving field essential for maintaining the efficient and smooth functioning of society. Its purpose is to protect against cyberattacks that can disrupt operations and interfere with the daily functioning of individuals and businesses. More and more aspects of our lives, such as communication, shopping, and financial transactions, are becoming reliant on the internet and digital platforms. As this expansion continues, the potential for data breaches grows exponentially.

A. Background

Networks play a crucial role in the distribution of information and the transfer of data across interconnected computer systems worldwide. Implementing network security protects data from unauthorized access and modification from threat actors, which ensures the three focal points of security: confidentiality, integrity, and availability of the data. As more digital systems are created and threat actors refine their techniques, the ability to successfully detect malicious events on a network is required.

Anomaly detection was traditionally performed using a signature-based approach. This method involves comparing active network traffic against a known set of attack signatures and patterns, and then flagging activity that matches. However, signature-based Intrusion Detection Systems (IDS) struggle to detect new, evolving, or unknown cyberattacks that do not have previously collected corresponding signatures. This requires constant updates to the database of known signatures to remain relevant and effective, which quickly becomes a time-consuming process.

Integrating both supervised and unsupervised machine learning (ML) methods with IDS allows us to overcome the weaknesses of a traditional signature-based approach. ML models trained on historical data can detect deviations from normal network traffic, identifying activities that may indicate malicious intent. As network traffic increases, an increasing amount of data is collected; however, ML models are adept at handling large volumes of data. As threat actors develop new and emerging attacks, ML models can adapt to changing patterns and trends in network traffic, reducing the number of manual updates.

B. Our Methodology

In this study, we aim to develop a robust Network IDS (NIDS) that can classify network traffic as either a benign event or an anomaly using various machine learning procedures. Due to the large number of features collected by IDS, we will identify the most critical features that contribute to an anomaly, allowing efficient intrusion detection. Through the calculation of information gain, we can determine which features play an important role in distinguishing a benign event from a malicious one.

We will implement anomaly detection using binary classification to identify the occurrence of suspicious activity within the network. Common ML models for this procedure include: Logistic Regression and Support Vector Machine (SVM). In Logistic Regression, we will conduct hyperparameter tuning to determine the optimal values for penalty and regularization strength. In SVM, we will perform hyperparameter tuning to find the optimal values for kernel, C, and gamma. We will evaluate the performance of the models by visualizing the confusion matrix and calculating the following metrics: accuracy, precision, recall, and F1 score.

We will further our understanding of network IDS by identifying and classifying malicious traffic into specific intrusion types, such as DDoS, Port Scans, and Bot attacks, with multi-class classification procedures. Common ML methods for this procedure include: Decision Trees and Artificial Neural Networks (ANN). In Decision Trees, we will perform hyperparameter tuning to find the optimal values for maximum depth, minimum leaf sample, and minimum sample split. For ANN, we will perform hyperparameter tuning to determine the optimal values for the number of hidden layers, the number of neurons per hidden layer, and the learning rate. The performance of the model will be evaluated in the same manner as previously described for binary classification.

In real world implementations, the true labels of network traffic are unknown. As a result, supervised learning methods are not applicable, so we will apply unsupervised learning techniques, such as K-Means clustering, to group network traffic based on underlying patterns. Through clustering, we can understand the distinguishing traffic behaviors and identify characteristics of each group without prior knowledge of labels. These procedures will enhance our understanding of feature relationships and support effective NIDS development.

II. Related Work

Previous research has been conducted exploring the implementation of ML in NIDS for detection of malicious traffic. Specific to binary classification of intrusion detection, Chowdhury et al. [1] presented an approach focused on enhancing detection accuracy by using random feature selection together with Support Vector Machines (SVM). The ML techniques were evaluated on the UNSW-NB15 dataset, which are raw network packets from a collection of modern normal activities and contemporary cyberattacks. The approach achieved a 98.76% accuracy, demonstrating that robust detection is possible with limited features.

Previous work on supervised machine learning techniques for cloud security was conducted in 2016 by Bhamare et al. [2] They focused on three binary classification models: SVM, Logistic Regression, and Decision Trees, with attack and normal as the labels. This approach highlighted a training set imbalance; only 32% of data represented the normal traffic. This study emphasized the need for further research in handling resilience to class imbalance, which has since been conducted.

A 2024 study by Talukder et al. [3] introduced a machine learning-based network intrusion detection model tailored for large and imbalanced datasets. The model employs Random Oversampling to address data imbalance, Stacking Feature Embedding based on clustering results, and Principal Component Analysis (PCA) for dimensionality reduction. The ML techniques implemented include Decision Trees and Random Forest, and the models achieved an accuracy rate up to 99.99%, which outperformed many existing methods. This demonstrates significant advancements in network intrusion detection.

Along with Decision Trees, other common models used with this issue have included Support Vector Machine and K-Nearest Neighbors. Machine Learning techniques such as these are often compared to Deep Learning techniques, such as convolutional neural networks (CNN). In 2023, the use of CNN was studied by Assay et al. [4] by chaining the outputs of CNN hidden blocks as the input of the next CNN hidden blocks. The results produced by this model were comparable to traditional ML models.

In multi-class classification, data is not only labeled as normal, but also as different cyberattacks, such as Denial of Service, User to Root, and Remote to Local. A study conducted by Hindy et al. [5] evaluated six machine learning models for intrusion detection in IoT environments. The models included: Logistic Regression, Naive Bayes, K-NN, SVM, Decision Tree, and Random Forest. The study compared flow-based features and packet-based features. Their results showed that flow-based features are more effective for identifying attacks on the Message Queuing Telemetry Transport (MQTT) protocol, while packet-based features perform better in traditional network intrusion detection.

These studies demonstrate the evolving nature of machine learning techniques and approaches to intrusion detection, which shows the continuing improvements in handling imbalanced data and feature selection. They also highlight the variety of models used across binary and multiclass classification tasks.

III. Problem Statement

The current challenge is the lack of an adaptive and effective Network Intrusion Detection Systems that can accurately identify cyberattacks, as threat actors continue to develop the scale, complexity, and frequency of attacks. The traditional Signature-based IDS relies on known patterns that are correlated to each attack type for detection; however, this approach has many limitations. The reliance on outdated and poorly performing systems can have severe consequences, such as data breaches, service disruptions, and financial loss. Our motivation is to utilize various ML techniques to produce NIDS that accurately detect anomalies and identify the specific attack types. Supervised learning methods are applicable to binary and multi-class anomaly classification tasks, and unsupervised learning is used for real-life scenarios where labels are not available. Feature selection is helpful to determine critical attributes of network traffic that are key to distinguishing malicious events from benign ones. Through these applications, our goal is to create a NIDS that reduces the need for manual intervention in updating the system, adapts to new attack strategies, and effectively identifies malicious activity to protect against cyberattacks.

IV. Data and Preprocessesing

A. Dataset Overview

We are working with the publicly available dataset developed by the Canadian Institute for Cybersecurity Intrusion Detection System. The motivation behind this dataset is to provide a reliable set of testing data for intrusion detection, as many existing datasets are out of date and do not contain a current representation of traffic volume and cyberattack diversity. We are exploring all data collected on Friday, July 7, 2017 between the hours of 9am and 5pm. There are 3 files that pertain to this date, and each file has the same structure. The dataset consists of 78 numerical features, represented as integers or floats. The target variable has four distinct classes: Benign, PortScan, DDoS, and Bot. Across all 3 files, there are a total of 703245 observations.

B. Preprocessing Techniques

Before combining all 3 data files, we identified any missing values in the attributes. We found that the "Flow Bytes/s" feature was the only one with null values in all files. There are a total of 47 missing values in this feature and, since this is an extremely small amount compared to the size of the total dataset, we decided to remove the rows that contained a null value. The descriptive statistics for each attribute revealed that 'Flow Bytes / s' and 'Flow Packets / s' both had a mean and a maximum value of 'inf' and a standard deviation of NaN. Between all 3 files, there are a total of 480 infinite values for both features. We replaced the infinite values with nulls and then imputed the null values with the median of that column. Including missing and infinite values can disrupt statistical calculations and introduce errors in classification and predictive modeling, so it is beneficial to only work with complete observations.

After handling null and infinite values, we combined all the files into one dataset that includes the Benign, PortScan, DDoS, and Bot observations. Since the current target column includes all 4 classes, we included an additional target column "Bin Label" with binary labels to prepare our data for binary classification. The binary classes are Benign and Anomaly, where the cyberattacks (PortScan, DDoS, and Bot) are considered an Anomaly and the Benign events remain labeled as Benign. The column that represents the multiclass labels was renamed as "Multi Label" to ensure consistency in the dataset.

C. Data Visualization

For our research, the target variable we are predicting is observed in its original multi-class form and in a binary form. In the multi-class form, each observation of the intrusion detection system can be classified as: Benign, Bot, DDoS, or PortScan. This will also be condensed into binary classes: Benign and Anomaly. Here, the Anomaly class comprises the observations not classified as Benign, so it is a combination of Bot, DDos, and PortScan. The distribution of the data can be observed in the following pie charts. The observations are 58.9 Benign and 41.1% Anomaly. This Anomaly class comprises: 22.6% PortScan, 18.2% DDoS, and 0.3% Bot.

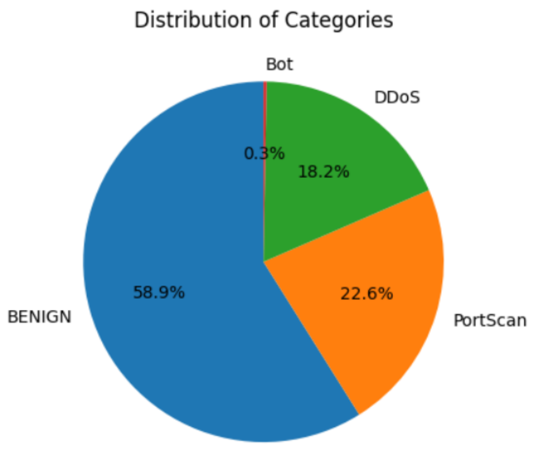


Fig. 1. This the distribution of Multi-Class Classification

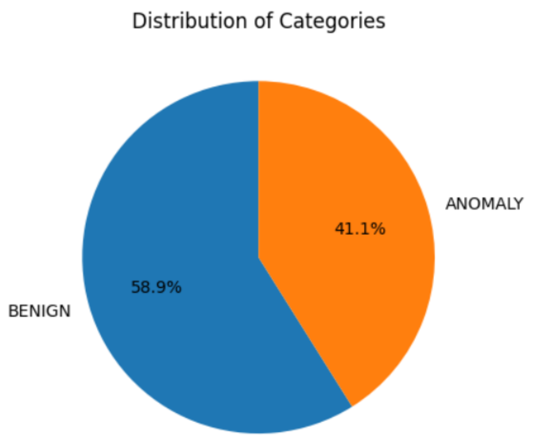


Fig. 2. This the distribution of Binary Classification

V. System Methodology

This section will describe the various ML algorithms and models that will be implemented throughout our project for binary and multi-class classifications, for identifying important features in classification, and for unsupervised learning.

A. Information Gain

To reduce the complexity of the dataset and improve model efficiency, we begin by calculating the information gain of each feature. This metric evaluates how much uncertainty (entropy) in the dataset is reduced when splitting on a specific feature.

$$G(A) = H(S) - H(S|A) \tag{1}$$

where A is an attribute and S is a set of examples.

$$H(S) = -\sum_{i=1}^{n} P(S=i)\log_2 P(S=i) \tag{2}$$

is the entropy of set S.

$$H(S|A) = \sum_{i=1}^{k} \frac{|S_i|}{|S|} H(S_i) \tag{3}$$

is the conditional entropy after splitting on A. Features with the highest information gain contribute the most to distinguishing between benign and anomalous activity. By selecting only the most informative features, we can reduce overfitting and improve model training time.

B. Logistic Regression

Logistic Regression is a linear model used for binary classification problems, which is appropriate for identifying whether an individual event over the network is benign or anomalous. It is easy to interpret and computationally efficient. The logistic regression utilizes the sigmoid function to transform the output of linear regression into a probability between 0 and 1 to represent the likelihood of an outcome. The sigmoid function is as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4}$$

To prevent overfitting and enhance generalization, a penalty term is added to the loss function. This study considers two types of regularization: L1 (LASSO) and L2 (Ridge). The optimal regularization method will be determined through hyperparameter tuning and performance comparison.

The expression for L1 (LASSO) and L2 (Ridge) is as follows:

$$R_1(\theta) = \lambda \sum_{j=1}^{n} |\theta_j| \tag{5}$$

$$R_2(\theta) = \frac{\lambda}{2} \sum_{j=1}^{n} \theta_j^2 \tag{6}$$

where the cost function is

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) \right] \tag{7}$$

C. Support Vector Machine (SVM)

Support Vector Machine is a classification algorithm that aims to find the optimal hyperplane that separates classes by the maximum margin. SVM is especially useful for binary classification making it appropriate for distinguishing between benign and malicious network traffic activity. The optimal hyperplane of the model that acts as a decision boundary is given by:

$$w_0 x + b_0 = 0 \tag{8}$$

The hyperplanes of the positive class and negative class are as follows:

$$w_0 x + b_0 = 1 \tag{9}$$

$$w_0 x + b_0 = -1 \tag{10}$$

We will perform hyperparameter tuning for the kernel type, regularization parameter (C), and kernel coefficient (gamma) to enhance model accuracy and improve generalization. The kernel type determines the type of decision boundary, C decides the trade-off between maximizing the margin and minimizing the training error, and gamma decides the amount of curvature present in the decision boundary.

D. Decision Trees

Decision Trees are a hierarchical model that recursively splits data based on a feature using criteria such as information gain or Gini impurity. Since our labels are discrete, we are generating a classification tree. The model is well-suited for multi-class classification, which makes it an appropriate choice for distinguishing between different types of cyberattacks (DDoS, PortScan, Bot). Decision Trees are beneficial because they can handle both numerical and categorical data. We will use hyperparameter tuning to optimize max depth, min samples leaf, and min samples split to balance bias and variance for the model. To choose the best attribute for splitting feature, refer to (1) for information gain and the equation for Gini impurity is

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2 \tag{11}$$

E. Artificial Neural Networks (ANN)

Artificial Neural Networks are modeled after the human brain and use interconnected nodes organized in layers to process data. ANN is appropriate for classifying the different types of cyberattacks because it is highly effective with multi-class classification tasks. ANN can learn abstract patterns and adapt to the evolving aspects of cyberattacks and network traffic. We will design an ANN with at least one hidden layer and use ReLU activation in the hidden layers and softmax activation in the output layer for multi-class classification. Hyperparameter tuning will be performed for the number of hidden layers, number of neurons per layer, and learning rate to improve classification accuracy and convergence.

F. K-Means Clustering

K-Means clustering is an unsupervised ML technique that groups similar instances based on patterns within the features and without relying on labels. It works by randomly initializing cluster centroids, assigning each data point to the nearest centroid, and then iteratively updating the centroids based on the mean of the cluster points until convergence. The nearest centroid is determined through the Euclidean distance formula, where x is a data point, c is a cluster centroid, and n is the number of features:

$$d(\mathbf{x}, \mathbf{c}) = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2} \tag{12}$$

K-Means clustering is an appropriate model because each resulting cluster represents network traffic with similar characteristics. Since real world network traffic has unknown labels, applying K-Means will group similar network behaviors together, allowing us to discover innate patterns that may be associated with benign activity or different cyberattacks. This improves our ability to identify and characterize anomalies in network traffic.

VI. Our Implementations

This section describes the steps and results of the various ML techniques we implemented. For all models, the data was split 80% for training and 20% for testing.

A. Determining Critical Features

Calculating the information gain for each feature for the binary and multi-class target variables allows us to determine the most important features that impact network traffic as a benign and malicious event. We selected the top 20 attributes to build more accurate and efficient models. Since the dimensionality of attributes is high, selecting certain features reduces overfitting, simplifies the models, and improves the training and testing times. The Tables IV and V display the top 20 attributes with the highest information gain values for multi-class and binary classification, respectively.

B. Binary Classification

1) Logistic Regression: To develop a binary classification model for anomaly detection, we implemented Logistic Regression using the top 20 features selected through information gain analysis. In order to use the best performing model, hyperparameter tuning was conducted using a GridSearchCV procedure. Some of the combinations checked included penalty type (L1 or L2) and regularization strength (C) were optimized through 3-fold cross-validation. The optimal model was found to use L1 regularization with a regularization strength of C = 0.01, and the saga solver was selected to efficiently handle the large dataset.

2) SVM: There are various ways to implement Support Vector Machines (SVMs). Choosing the correct model, or hyperparameter tuning, meant comparing the different strengths and seeing how they best aligned with our data. The first implementation of SVM utilized kernel support, or a model that can utilize the "kernel trick" to capture nonlinear relationships. Due to the sheer size of our data this model, Support Vector Classification, did not finish running even after an extensive amount of time. The reason behind this was that the model tried to build a kernel matrix which takes a time complexity of O (n^2). This meant the final model we used, Linear Support Vector Classifier, would not have kernel support. Due to the lack of kernel support, in order to implement it we had to first scale the inputs. Scaling the inputs is required to ensure that no numerical feature dominates the model due to size, rather than its significance. To scale we used StandardScaler, which performs Z-score normalization to each input.

3) Ensemble Model: In order to improve overall performance of binary classification, we created an ensemble model. The main function of an ensemble model is to combine independent models, in our case we combined Logistic Regression with SVM. The ensemble model took in initialized models of Logistic Regression and SVM. For SVM, it was still required to scale the inputs. These initialized models make up our list of estimators that already are fit to the data. To reduce the amount of variance in overall performance, we chose a VotingClassifier ensemble model. This was done with the goal of seeing the result for combining the predictions of our estimators. For the voting type we needed to use "hard" voting. This meant a vote was either casted in one binary classification output, or the other. The other kind of voting, "soft", was not an option for our model because this meant averaging class probabilities. Since these probabilities are not available in Linear Support Vector Classification, we used "hard".

C. Multi-Class Classification

1) Decision Tree: When implementing Decision Trees for multiclass classification, the extreme class imbalance between Benign, DDos, PortScan, and Bot classes resulted in biased and inaccurate predictions. To address the class imbalance, we performed under-sampling, which reduces the number of samples in the majority classes to match the number of observations of the minority class. The original class distribution and the resampled class distribution are shown in Table VI. An initial Decision Tree was trained on the top 20 features based on information gain and with a maximum of 20 leaf nodes. The resulting tree is shown in Fig 18. To ensure, the model is performing at its best, hyper-tuning was conducted on the following parameters:

- Criterion = function to measure the quality of a split
- Max_depth = maximum depth of the tree
- Min_samples_split = minimum number of samples required to split an internal node, min
- Min_samples_leaf = minimum number of samples required to be at a leaf node

The values considered for each parameter are shown in Table I. The resulting hyper-tuned tree is shown in Fig 20. After tuning, the tree demonstrated improved accuracy, but it became significantly larger and more complex with multiple nodes and splits. This complexity hindered transparency and interpretability of the tree, which is one of the primary advantages of this model.

To address the complexity of the tree, we introduced a new hyperparameter: max_leaf_nodes. This parameter places a limit on the number of leaf nodes that can exist in the tree. The parameter values we explored for max_leaf_nodes are shown in Table II. The tree produced with this added constraint is shown in Fig. 22, and it is notably more compact and interpretable compared to its unrestricted tree.

TABLE I
Hyperparameters: No Limit on Leaf Nodes

Parameter	**Values**
`criterion`	`['gini', 'entropy', 'log_loss']`
`max_depth`	`[5, 10, 15, 20, 25, 30, None]`
`min_samples_split`	`[2, 5, 10]`
`min_samples_leaf`	`[1, 2, 4]`

2) ANN: To implement an ANN model, we performed one-hot encoding to represent the categorical target variables as numerical values that are suitable for the model to use. The features were standardized using StandardScalar in order to improve the efficiency of the model and increase the model convergence during training. We constructed an ANN model with 2 hidden layers that used the ReLU activation function.

TABLE II
HYPERPARAMETERS: LIMITED LEAF NODES

Parameter	Values
`criterion`	`['gini', 'entropy', 'log_loss']`
`max_depth`	`[5, 10, 15, 20, 25, 30, None]`
`min_samples_split`	`[2, 5, 10]`
`min_samples_leaf`	`[1, 2, 4]`
`max_leaf_nodes`	`[5, 10, 15, 20, 25, 30, 35, 40]`

Each hidden layer is followed by a dropout layer with a dropout rate of 0.5, meaning half of the neurons are disabled in each iteration. A final output layer uses the softmax activation function and has 4 units, representing the 4 classes we are looking to classify. The ANN model was trained using the Adam optimizer.

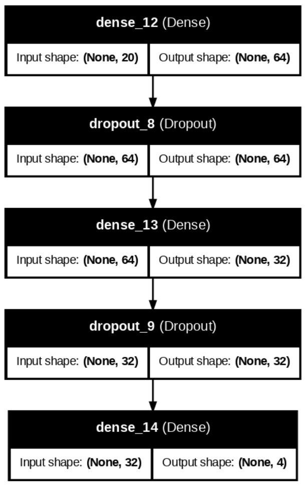


Fig. 3. ANN Model Structure.

To determine the optimal learning rate, we conducted hyperparameter tuning over four learning rates: 0.1, 0.01, 0.001, and 0.0001. For each learning rate, a model was trained for 10 epochs with a batch size of 128 and with 20% of the training data for validation. All models were evaluated with the full testing dataset, and the learning rate that produced the highest test accuracy was the learning rate in the final model.

The final ANN model had the same structure and used the Adam optimizer. It was trained for 25 epochs with a batch size of 32 and with 20% of the training data for validation. Early stopping was also implemented with a patience of 5 epochs to prevent overfitting and retain the best-performing weights by stopping the training process once the model's performance on the validation set began to degrade. The final model architecture is summarized in Fig. 3.

D. Unsupervised Learning

To implement the unsupervised learning model, K-Means Clustering, we standardized the features using StandardScaler. This normalizes the scale of each feature, which is important for models that are based on distance like K-Means. To determine the optimal k value, which is the number of clusters, we performed the Elbow Method. We computed the inertia for k values 1 to 10 and plotted the inertia values versus the corresponding k value to identify the elbow point, which represents the optimal cluster count. By looking at the Fig 4, we can identify the optimal k is 4 clusters. This corresponds to the different types of network traffic in the dataset: Benign, DDoS, PortScan, and Bot.

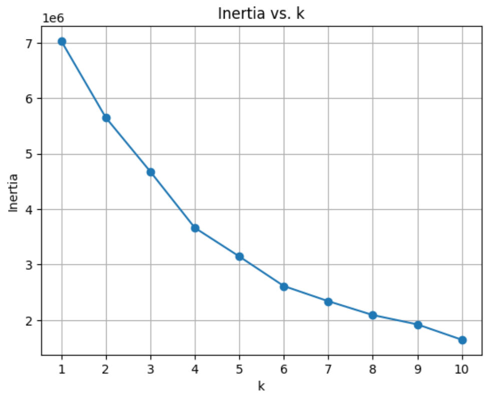


Fig. 4. Inertia v k Plot for Elbow Method.

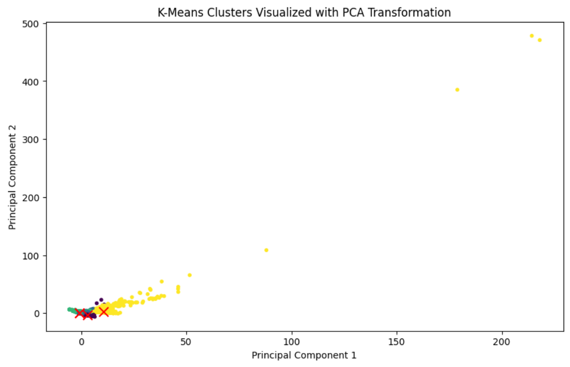


Fig. 5. k=4, K-Means Clusters.

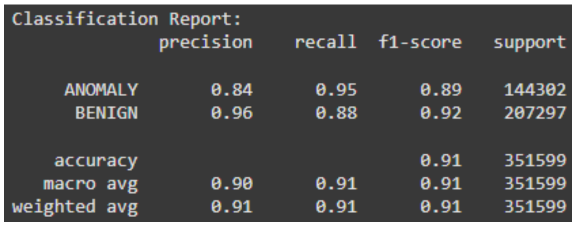
The K-Means model was trained with the optimal k = 4, and we reduced the data into 2 principal components using Principal Component Analysis (PCA) in order to visualize the clusters. The centroids of each cluster are also displayed on the cluster scatter plot in Fig 5. A confusion matrix was constructed in order to conduct cluster-to-class mapping to identify the most frequent true class label within each cluster. The points in the clusters were then assigned the corresponding highest occurring network traffic type as its target label, which is shown in Table III.

VII. PERFORMANCE EVALUATION

A. Binary Classification

1) Logistic Regression: Evaluating the model's performance on the test set, the Logistic Regression model achieved an accuracy of 91%, a precision of 91%, a recall of 91%, and an F1-score of 91%. The confusion matrix indicated that while the model had a slight tendency to misclassify some anomalous events as benign, it demonstrated strong recall in detecting anomalies (95%) and strong precision in identifying benign events (96%). This balance between precision and recall aligns well with the goal of the project: ensuring that suspicious network activity is reliably detected while minimizing the number of false alarms.

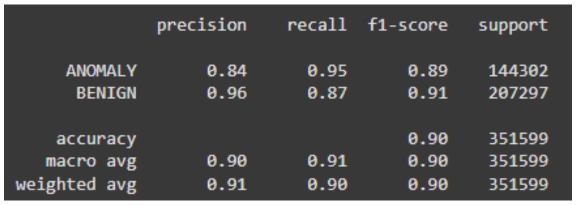
The use of L1 regularization not only helped prevent overfitting but also encouraged sparsity in the model, effectively reducing the influence of less important features. These results confirm that Logistic Regression, when combined with feature selection based on information gain, can serve as an interpretable and effective method for real-time anomaly detection within NIDS.



```
Classification Report:
              precision    recall  f1-score   support

     ANOMALY       0.84      0.95      0.89    144302
      BENIGN       0.96      0.88      0.92    207297

    accuracy                           0.91    351599
   macro avg       0.90      0.91      0.91    351599
weighted avg       0.91      0.91      0.91    351599
```

Fig. 6. Classification Report for Logistic Regression.

2) Support Vector Machines (SVM): Overall, the SVM model performed well. It produced an overall accuracy of 90%. For anomalies there was a precision of 84%, a recall of 95%, and a F1-score of 89%. For benign instances there was a precision of 96%, a recall of 87%, and a F1-score of 91%. This shows that for SVM, the model was very good at detecting real anomalies, as indicated by that class's high recall. This leaning toward classifying instances as anomalies is better for the high-risk situations that cyber attacks can occur in.

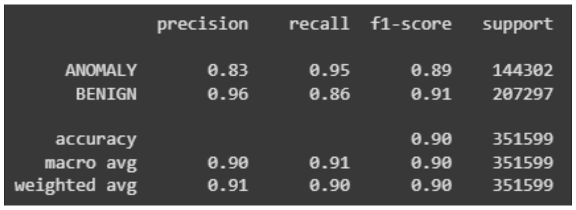


```
              precision    recall  f1-score   support

     ANOMALY       0.84      0.95      0.89    144302
      BENIGN       0.96      0.87      0.91    207297

    accuracy                           0.90    351599
   macro avg       0.90      0.91      0.90    351599
weighted avg       0.91      0.90      0.90    351599
```

Fig. 7. Classification Report for SVM.

3) Ensemble Model: At first, the ensemble model took in the Logistic Regression model previously created along with the SVM model previously created. This produced an overall accuracy of 82%, which was lower than both the other binary classifications separately. In the final version of the model, a new, and untrained, model for both Logistic Regression and Support Vector Machines was used as the estimators. Given these parameters the model performed with an accuracy of 90%. In addition to its accuracy, for anomaly detection the model had a precision of 83%, a recall of 95%, and a F1-score of 89%. For the benign classification, the model had a 96% precision, a recall of 86%, and a F1-score of 91%.

The better performance of new models compared to already trained models may indicate that those models may be overfit to the data. By using the new model, more generalizations could be obtained. Overall, the high recall that was obtained in the final model was most aligned with the high-risk implementations it is intended for.

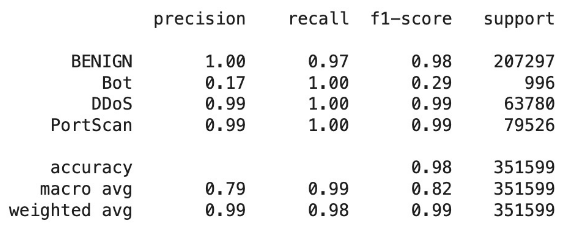


```
              precision    recall  f1-score   support

     ANOMALY       0.83      0.95      0.89    144302
      BENIGN       0.96      0.86      0.91    207297

    accuracy                           0.90    351599
   macro avg       0.90      0.91      0.90    351599
weighted avg       0.91      0.90      0.90    351599
```

Fig. 8. Classification Report for Binary Ensemble Model.

B. Multiclass Classification

1) Decision Model: The initial decision tree model with a max of 20 leaf nodes achieved an overall model accuracy of 98%, indicating a strong performance in general; however, the classification report shown in Fig. 9 reveals the precision, recall, F1-score of each class. Despite under-sampling to combat class imbalance, there is still a severe disparity between the successful classification of Bot cyberattacks and the remaining network traffic types within the testing set. The Bot class has a low precision of 17% and high recall of 100%, indicating that the model identifies most of the actual instances successfully, but produces many false positives. From the confusion matrix shown in Fig. 19, the model correctly classified all 996 Bot samples, but over 4,700 Benign samples were misclassified as Bot. The DDos and PortScan cyberattacks both had high precision and high recall. Additionally, very few instances of DDoS and PortScan traffic were misclassified as seen in the confusion matrix. The Benign events were the majority class and resulted in high precision and recall. The macro-averaged F1-score of 82% reflects the imbalance within the class performances.

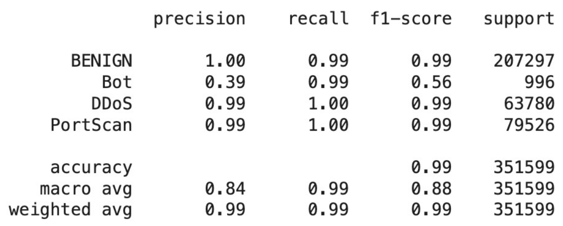


```
              precision    recall  f1-score   support

      BENIGN       1.00      0.97      0.98    207297
         Bot       0.17      1.00      0.29       996
        DDoS       0.99      1.00      0.99     63780
    PortScan       0.99      1.00      0.99     79526

    accuracy                           0.98    351599
   macro avg       0.79      0.99      0.82    351599
weighted avg       0.99      0.98      0.99    351599
```

Fig. 9. Classification Report for Initial Decision Tree.

The first hypertuned tree with no limit on the number of leaf nodes improved its overall model accuracy to 99%. The class imbalance was still visible in the classification report shown in Fig. 10; however, the precision of the Bot class increased significantly to 39%, which is over double the precision from the initial model. This indicates that the model can better balance correctly identifying Bot events and reducing false positives.

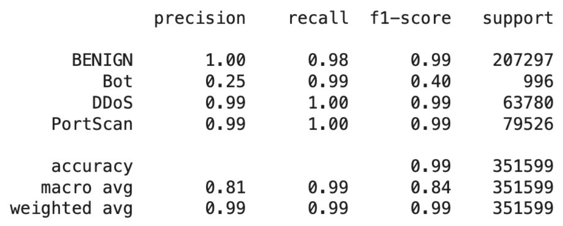
From the confusion matrix shown in Fig. 21, misclassification of Benign instances as Bots decreased by over 68%, which is a notable reduction in false positives. The DDoS and PortScan classes continue to have high precision and recall values and minimal errors, suggesting the model can consistently classify high volume traffic well. The Benign events remain accurately classified with an F1-score of 99%. The macro-averaged F1-score increased to 88% shows a stronger performance across all classes.



```
              precision    recall  f1-score   support

      BENIGN       1.00      0.99      0.99    207297
         Bot       0.39      0.99      0.56       996
        DDoS       0.99      1.00      0.99     63780
    PortScan       0.99      1.00      0.99     79526

    accuracy                           0.99    351599
   macro avg       0.84      0.99      0.88    351599
weighted avg       0.99      0.99      0.99    351599
```

Fig. 10. Classification Report for First Hyper-tuned Decision Tree.

The final hypertuned tree with a limit on the number of leaf nodes also achieved an overall accuracy of 99%. The classification report shown in Fig. 11 showed that Bot instances increased from 17% precision of the initial model to 25% precision. The recall remained extremely high, allowing for an improved F1-score compared to the initial model, but a decreased one compared to the previous hypertuning model. The confusion matrix shown in Fig. 23, shows an increase in the misclassification of Benign events as Bots compared to the other hypertuning model, but not as severe as the initial model. The DDoS and PortScan instances were classified well with high, unchanged precision and recall values and only slight variations in misclassifications compared to both previous models. The Benign events were still classified successfully with an F1-score of 99%, though there were the minor increases in misclassifications. The macro-averaged F1-score increased slightly to 84%, indicating a slight increase in performance, but there still remains an imbalance across classes.



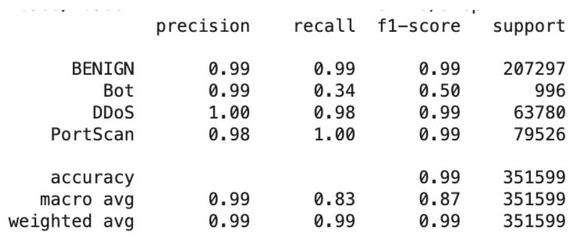
```
              precision    recall  f1-score   support

      BENIGN       1.00      0.98      0.99    207297
         Bot       0.25      0.99      0.40       996
        DDoS       0.99      1.00      0.99     63780
    PortScan       0.99      1.00      0.99     79526

    accuracy                           0.99    351599
   macro avg       0.81      0.99      0.84    351599
weighted avg       0.99      0.99      0.99    351599
```

Fig. 11. Classification Report for Final Hyper-tuned Decision Tree.

The final decision tree demonstrates that simplifying a model through structural limitations, like maxing the number of leaf nodes, can produce better generalization and interpretable results without compromising accuracy and model performance.

One advantage of decision tree models is that they provide clear visuals into the decision process, making them highly accessible for human understanding. The models are computationally efficient, which allows for faster training and evaluation, which is very beneficial when conducting multiple hyperparameter tuning processes. However, the disadvantages include being prone to overfitting, especially without constraints like max_leaf_nodes. Decision trees may not perform well on imbalanced datasets, as seen by the poor performance in classifying Bot cyberattacks. It is likely that the model fails to detect the minority class effectively.

2) ANN: To construct the best model, we evaluated four learning rates and chose the one with the best accuracy. As shown in Table VII, the learning rate of 0.001 outperformed the others with an accuracy of 98.42%. When using this learning rate in the final ANN construction, the model achieved a test accuracy of 98.95% and a test loss of 0.0323. This suggests that the model performs strongly on the validation and testing sets.



```
              precision    recall  f1-score   support

      BENIGN       0.99      0.99      0.99    207297
         Bot       0.99      0.34      0.50       996
        DDoS       1.00      0.98      0.99     63780
    PortScan       0.98      1.00      0.99     79526

    accuracy                           0.99    351599
   macro avg       0.99      0.83      0.87    351599
weighted avg       0.99      0.99      0.99    351599
```

Fig. 12. Classification Report for ANN Model.

The classification report shown in Fig. 12 displays the high precision, recall, and F1-scores for Benign, DDoS, and PortScan classes. The Bot class resulted in a high precision of 99% and low recall of 34%, indicating the model is highly accurate when predicting an event as a Bot cyberattack, but it fails to detect the majority of true instances of Bot cyberattacks. From the confusion matrix displayed in Fig. 24, the majority of misclassifications occurred for the Bot class, where 644 Bot instances were misclassified as Benign events. There were minimal errors in the other classes, which further emphasizes the class imbalance. The macro average F1-score was 87% and shows that the model performs very well overall, but the performance is uneven across classes.

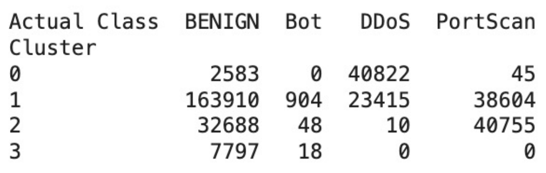
One advantage of ANN is the ability to model complex, non-linear patterns and capture generalizations effectively. This model achieved a high F1-score of 99% for Benign, DDoS, and PortScan, and a reasonable F1-score of 50% for the Bot class, despite class imbalance. However, a major disadvantage of ANN is that it requires significant training time and computational resources. It also has limited interpretability, since its process is not accessible to view. The ANN models are also sensitive to the learning rate, making it an essential process. Failing to hypertune the model effectively could result in poor performance overall.

C. Unsupervised Learning

The K-Means clustering technique resulted in moderate performance when labeling network traffic data. Due to the class imbalance, the Bot class was not a majority class in any cluster. From Fig 13, it is shown that Cluster 1 has 904 Bot instances, which is 93% of the Bot observations. However, there are over 163,000 Benign events that occur in Cluster 1, mapping this cluster to the Benign class.

TABLE III
CLUSTER TO LABEL MAPPING

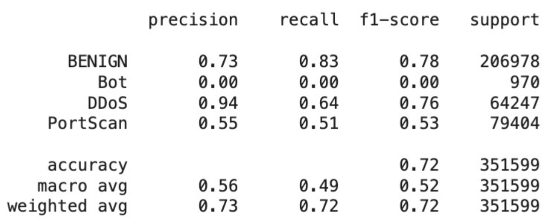
Cluster ID	Mapped Label
0	DDoS
1	BENIGN
2	PortScan
3	BENIGN



```
Actual Class  BENIGN  Bot   DDoS  PortScan
Cluster
0               2583    0  40822        45
1             163910  904  23415     38604
2              32688   48     10     40755
3               7797   18      0         0
```

Fig. 13. Clusters v Label.

The classification report that evaluates the performance of K-Means at successfully labeling the network traffic is displayed in Fig. 14. The model achieved an overall accuracy of 72%, with particularly strong precision for the DDoS class at 94% and high recall for the Benign class at 83%. The F1-score for PortScan revealed moderate success at classification with a value of 53%. Since the model failed to correctly identify a cluster as the Bot class, the Bot class held 0 values for the precision, recall, and F1-score. The macro-average F1-score was relatively low at 52%, emphasizing the unequal performance across classes from the class imbalance.



```
              precision    recall  f1-score   support

      BENIGN       0.73      0.83      0.78    206978
         Bot       0.00      0.00      0.00       970
        DDoS       0.94      0.64      0.76     64247
    PortScan       0.55      0.51      0.53     79404

    accuracy                           0.72    351599
   macro avg       0.56      0.49      0.52    351599
weighted avg       0.73      0.72      0.72    351599
```

Fig. 14. Classification Report for K-Means.

One advantage of K-Means is its simplicity and computational efficiency, making it suitable for large datasets. It also does not require labeled data, which is useful for realistic interpretation of network traffic data, since events will not be labeled in true implementation of NIDS. However, disadvantages of K-Means include sensitivity to initial centroids and poor performance with imbalanced data, as demonstrated by the inability to successfully identify the Bot class. There is also the need to predefine the number of clusters, which can impact the performance. Additionally, K-Means lacks interpretability and may produce inconsistent cluster-to-label mappings, causing the model to be difficult to understand.

VIII. CONCLUSION AND FUTURE WORK

This study showed the effectiveness and limitations of various machine learning models used to identify and classify malicious cyberattacks within network traffic. We implemented Logistic Regression and SVM for binary classification, and Decision Trees and ANN for multiclass classification. All classification models that we implemented performed well overall and with high overall accuracies; however, none of the models excelled in correctly identifying all classes.

For binary classification, Logistic Regression performed strongly with a high overall accuracy and a strong recall value for the anomaly class. This is sought after, especially in intrusion detection, because network traffic data can be very valuable and hold sensitive information. SVM focused on high anomaly recall, which is an important characteristic for high-risk environments.

For multi-class classification, ANN delivered highly accurate results across classes and was the best at identifying the minority Bot class. Decision Trees provided interpretable models and benefited from hyperparameter tuning to improve evaluation metrics, but it was heavily sensitive to the class imbalance. The Bot class was difficult for all models to identify due to the limited number of events that were available in the data.

Unsupervised learning performed through K-Means clustering further showed the challenges of dealing with imbalanced datasets. The clustering correctly identified the majority classes well, but it failed to distinguish minority cyberattacks, like Bot instances. While our current approaches achieve high accuracy on previously collected, organized, and labeled datasets, a significant next step is enabling the models to function effectively for real-time network intrusion detection. Real-time adaptation will behave differently from our current implementation because the models will continuously process incoming network traffic and adapt to evolving attack types without manual intervention.

Various changes will be applied to enable real-time functionality in our models. The first step is to replace static datasets with streaming data pipelines. This allows continuous capturing, processing, and delivering of network traffic data for immediate actions in our models. Popular streaming platforms to explore are Apache Kafka, Apache Flink, and Amazon Kinesis. Next, we will need to redesign the feature engineering and data preprocessing procedures to process raw packets. Our implementation of information gain is computed once on static data and used throughout our models; however, to save computation time when dealing with streaming data, it would be beneficial to use a set of previously selected critical features based on previous analysis. A final step would be to implement a real-time alert system that triggers warnings when a malicious event is encountered by the models.

REFERENCES

[1] M.N. Chowdhury, K. Ferens, M. Ferens, ”Network intrusion detection using machine learning”, *Proceedings of International Conference on Security Management (SAM)*, Las Vegas, USA, 2016, pp. 1–7.

[2] D. Bhamare, T. Salman, M. Samaka, A. Erbad, and R. Jain, "Feasibility of supervised machine learning for cloud security," in *Proc. Int. Conf. Information Science and Security (ICISS)*, IEEE, 2016.

[3] Md A. Talukder, Md Manowarul Islam, et al, "Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction," *Journal of Big Data*, vol. 11, no. 33, February, 2024.

[4] Ahmed Tamer Assy, Yahia Mostafa, Ahmed Abd El-khaleq, Maggie Mashaly, "Anomaly-Based Intrusion Detection System using One-Dimensional Convolutional Neural Network," *Procedia Computer Science*, vol. 220, pp. 78-85, March, 2023.

[5] 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., Lecture Notes in Networks and Systems, INC 2020, Springer, Cham, 2021.

IX. Appendix

TABLE IV
TOP 20 FEATURES WITH HIGHEST INFORMATION GAIN FOR MULTICLASS CLASSIFICATION

Feature	Information Gain
Flow Bytes/s	1.2861
Average Packet Size	1.2772
Total Length of Fwd Packets	1.2015
Subflow Fwd Bytes	1.2015
Packet Length Mean	1.1802
Flow Duration	1.1732
Fwd Packets/s	1.1692
Flow Packets/s	1.1691
Destination Port	1.1632
Flow IAT Mean	1.1433
Packet Length Std	1.1342
Packet Length Variance	1.1341
Flow IAT Max	1.0946
Fwd Packet Length Mean	1.0846
Avg Fwd Segment Size	1.0846
Fwd Packet Length Max	1.0785
Total Length of Bwd Packets	1.0543
Subflow Bwd Bytes	1.0543
Bwd Packets/s	1.0522
Init_Win_bytes_forward	1.0373

TABLE V
TOP 20 FEATURES WITH HIGHEST INFORMATION GAIN FOR BINARY CLASSIFICATION

Feature	Information Gain
Flow Bytes/s	0.8626
Average Packet Size	0.8548
Total Length of Fwd Packets	0.7795
Subflow Fwd Bytes	0.7795
Packet Length Mean	0.7590
Flow Duration	0.7540
Fwd Packets/s	0.7494
Flow Packets/s	0.7493
Destination Port	0.7408
Flow IAT Mean	0.7235
Packet Length Std	0.7194
Packet Length Variance	0.7194
Flow IAT Max	0.6760
Fwd Packet Length Mean	0.6703
Avg Fwd Segment Size	0.6703
Fwd Packet Length Max	0.6687
Total Length of Bwd Packets	0.6431
Subflow Bwd Bytes	0.6431
Bwd Packets/s	0.6356
Init_Win_bytes_forward	0.6231

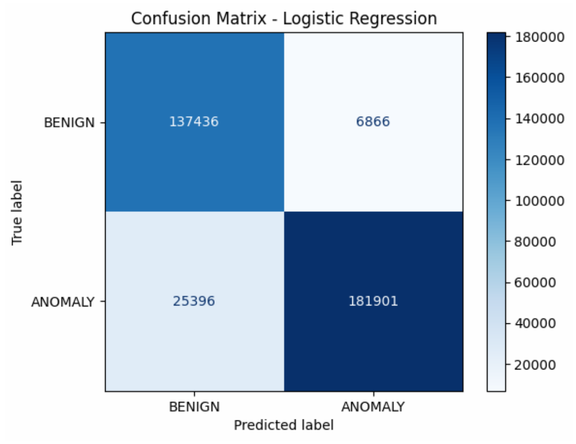


Fig. 15. Confusion Matrix for Logistic Regression.

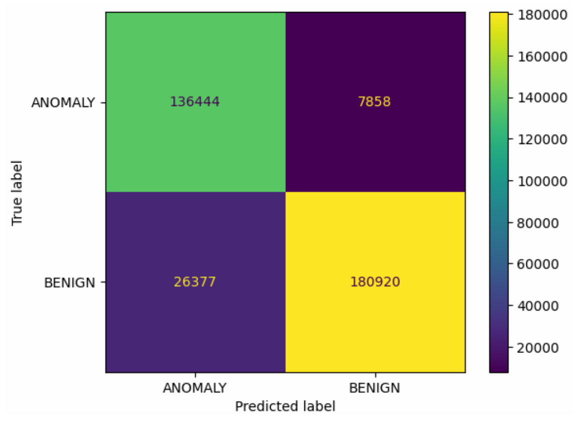


Fig. 16. Confusion Matrix for SVM.

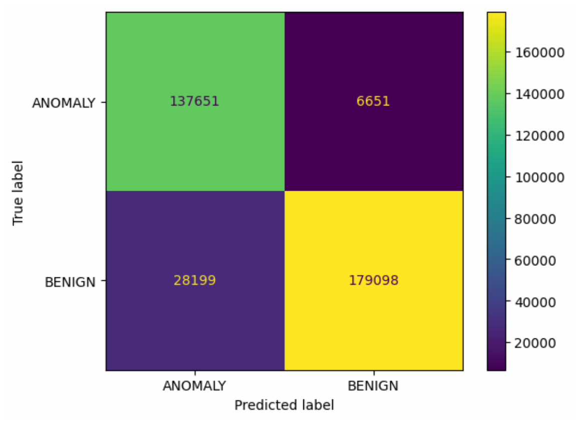


Fig. 17. Confusion Matrix for Ensemble Model.

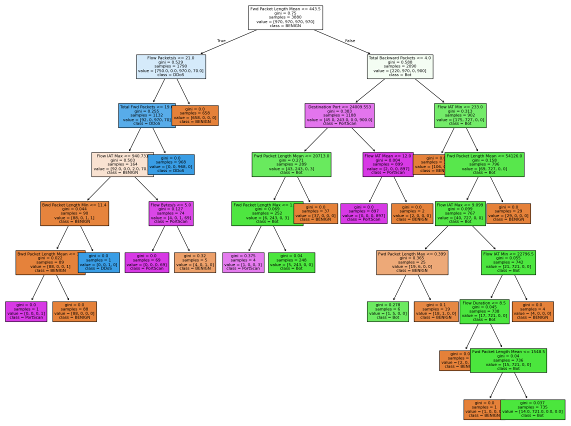


Fig. 18. Initial Decision Tree with max 20 leaf nodes.

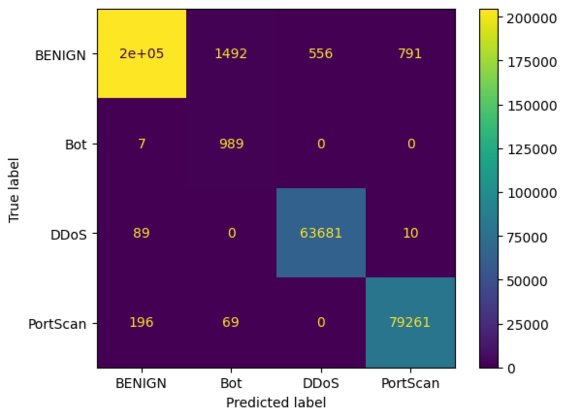


Fig. 21. Confusion Matrix for First Hyper-tuned Decision Tree.

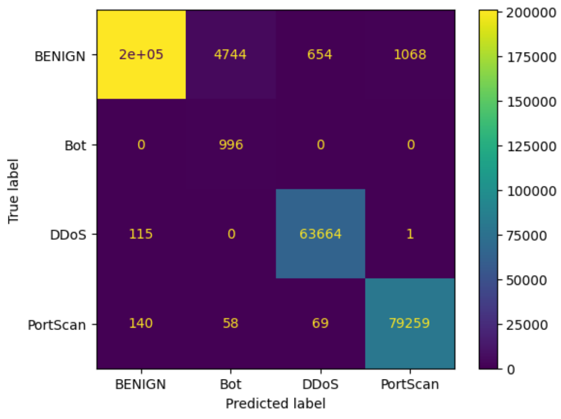


Fig. 19. Confusion Matrix of Initial Decision Tree.

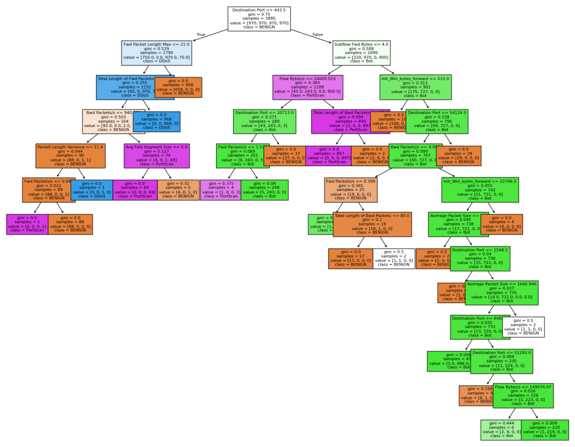


Fig. 22. Hyper-tuned Decision Tree with limit on leaf nodes.

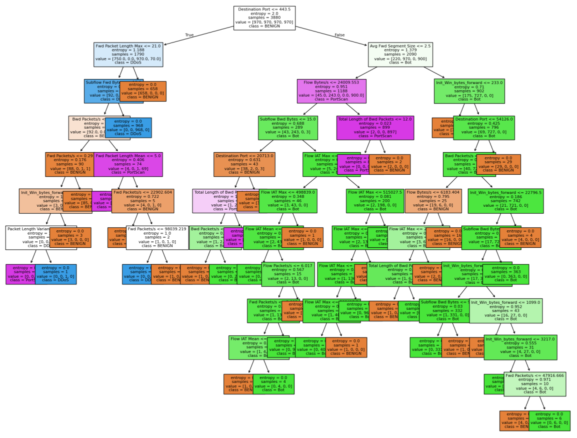


Fig. 20. Hyper-tuned Decision Tree with no limit on leaf nodes.

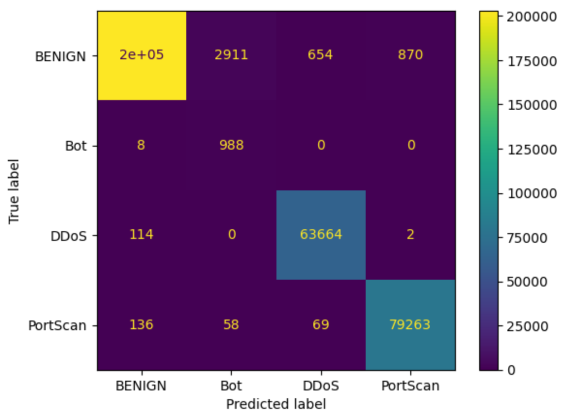


Fig. 23. Confusion Matrix for Final Hyper-tuned Decision Tree.

TABLE VI
CLASS DISTRIBUTION BEFORE AND AFTER RESAMPLING

Class	Original Distribution	Resampled Distribution
BENIGN	206978	970
PortScan	79404	970
DDoS	64247	970
Bot	970	970

TABLE VII
ANN TEST ACCURACY AND LOSS FOR DIFFERENT LEARNING RATES

Learning Rate	Test Accuracy	Test Loss
0.1	0.5896	0.9750
0.01	0.9734	0.0758
0.001	0.9842	0.0537
0.0001	0.9638	0.1280

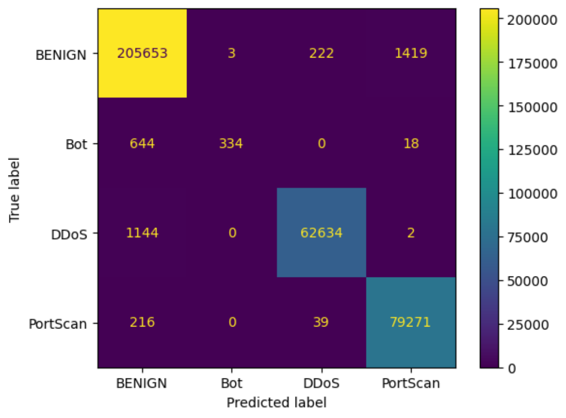


Fig. 24. Confusion Matrix for ANN Model.

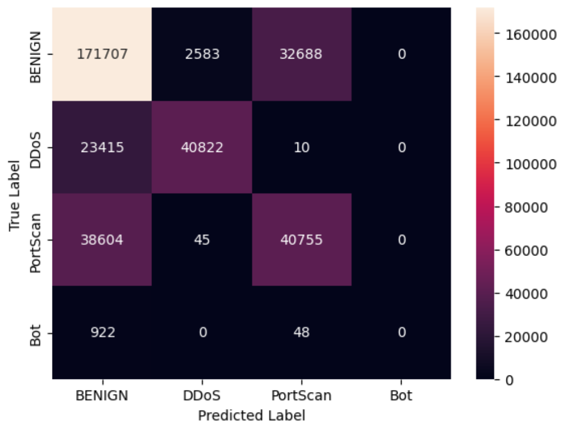


Fig. 26. Confusion Matrix for K-Means.

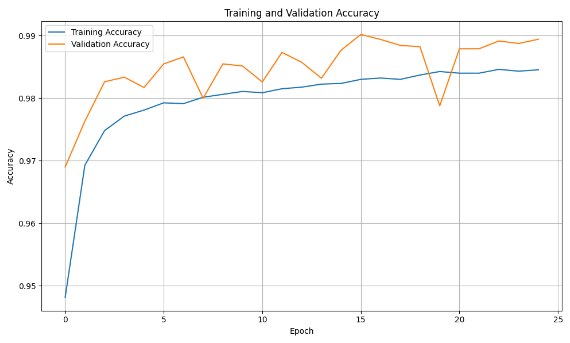


Fig. 25. Training and Validation Accuracy of ANN Model.

TABLE VIII
K-MEANS CLUSTERING CENTROIDS

Row	Column 1	Column 2
0	2.9682	-2.8122
1	-0.6962	0.2818
2	-0.7513	0.5317
3	10.7676	2.4582