Intrusion Detection System (May 2024)

Brandon Lipscomb, Adam Parness, and Ian Sanchez

***Abstract*—Due to the exponential rise in the number of applications operating on computer networks and the dramatic growth in their usage, network security is becoming more and more important. All systems are vulnerable to security breaches, which could lead to an increase in attacks that could have a negative impact on the economy and overall security ranging from ordinary people to governing bodies. Thus, it is now more crucial than ever to accurately and quickly identify system vulnerabilities in the network. This paper will discuss, compare, and evaluate a variety of machine learning techniques implemented to complete this task.**

I. Introduction

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Intrusion Detection Systems (IDS) are critical for network security, serving as essential components for detecting unauthorized access and anomalies in network traffic. The continuous monitoring of network activities helps prevent potential breaches by identifying suspicious behavior on the network. Furthermore, IDS aids in maintaining compliance with regulatory standards. The deployment of an IDS increases an organization's security when integrated with other security technologies including firewalls and antivirus programs. Additionally, forensic analysis relies on IDS systems to generate extensive activity logs which are crucial in the aftermath of security incidents. These logs provide valuable insights into the origins and impact of the breach.

The presence of an IDS can deter potential attackers, minimize damages of an attack by isolating compromised systems, and adapt to new threats through behavioral analysis and machine learning. The adaptability and cost-effectiveness of an IDS instills confidence in customers and partners.

II. Task Description

This project focuses on the development and evaluation of various machine learning models with the goal of detecting network intrusions. We chose to use the KDD Cup 1999 dataset to train and test our models. Prior to creating the model we imported essential Python libraries for data manipulation and visualization including numpy, pandas, matplotlib, and scikit-learn. The data is processed by adding labels, checking for redundancy and null values, and scaled to numerical values between 0 and 1. Several machine learning algorithms were implemented, including Logistic Regression, Random Forest, K-Nearest Neighbors, Decision Trees, Support Vector Machines, and Neural Networks. The performance of these models is assessed using metrics including accuracy, precision, recall, and ROC-AUC scores. Various optimization techniques, including PCA, L1 and L2 Regularization, dropout layers were used. The project aims to find the optimal model which identifies and classifies potential threats in network traffic with the highest accuracy.

A. Data Processing

For this project, we selected a dataset comprising 148,517 records, partitioned into 125,973 entries for training the model and 22,544 for testing purposes. Each entry consists of 42 features, utilized to predict the target feature, 'attack'. Originally, this target feature was categorized into 23 distinct classes. To streamline our model, we condensed these 23 classes into two categories: 'attack' and 'normal'. This categorization process is illustrated in Figure 1.

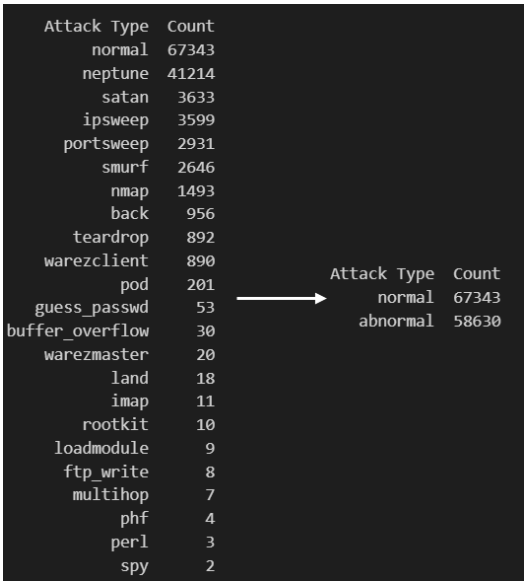

Fig. 1. Attack Type Class Transformation

The next step is to use a label encoder to convert strings into numbers, as numerical data is easier to process. After this transformation, we split the pandas DataFrame into two parts: 'x', containing the 42 features, and 'y', which holds the target feature.

We also preprocessed our training and testing data with Principal Component Analysis (PCA) to reduce the dimensionality of the data to speed it up when training and testing of the neural network. Later we will compare the performance of the preprocessed data with PCA and compared to the unprocessed data.

B. Data Visualization

Data visualization is essential to understanding a given

dataset. The KDD Cup 1999 dataset contains 42 features utilized to predict one target feature. With such a large number of features, it can be difficult to visualize how the features impact the target feature. We plotted any features against the target feature to visualize the relationship between features. Figure 2 shows the nine features that have the most visual impact on the target feature.

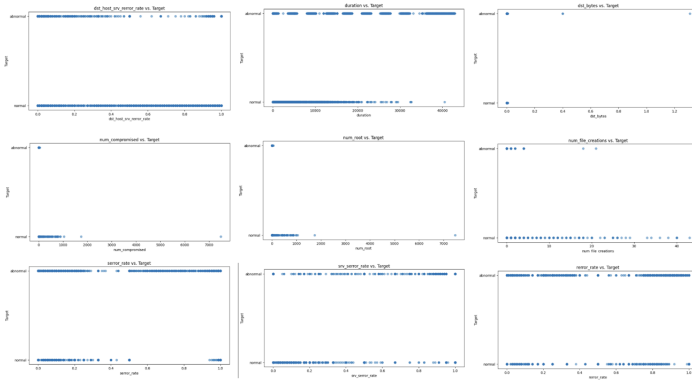

Fig. 2. Relationship Between Features and Target

Figure 3 displays a heat map of the mutual information between each feature and the target. Mutual information illustrates the amount of information that is shared between two features. A simple correlation measurement only finds the linear relationship between features. Mutual information values cover all types of relationships between features.

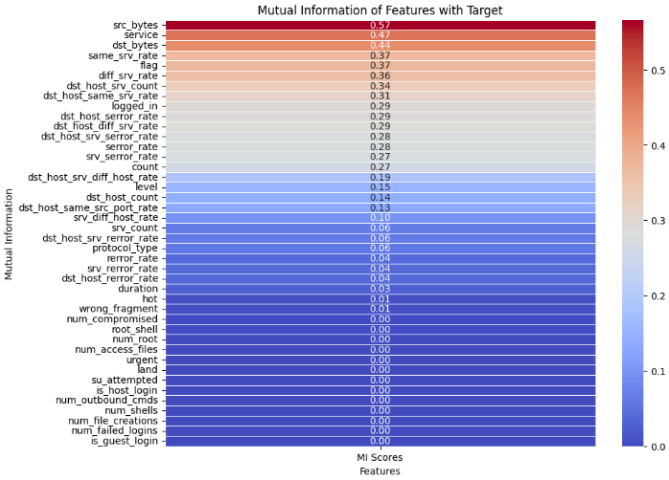

Fig. 3. Mutual Information Scores by Feature

Figure 4 shows the mutual information scores between the 20 features with the highest mutual information score to the target. This heat map reveals that there is redundancy between the features. The most redundant features include versions of serror rate, srv rate and count. This redundancy will be eliminated using techniques such as PCA.

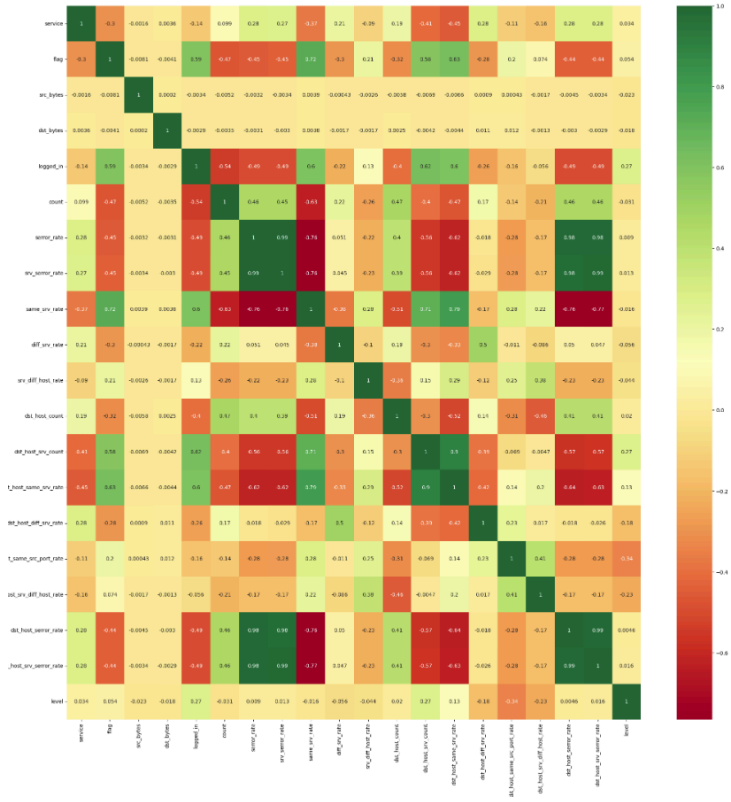

Fig. 4. Mutual Information Scores Between the Top 20 Features

III. Major Challenges and Solutions

The complex task of predicting a target feature from 42 distinct features presented significant challenges, which led us to evaluate a broad range of solutions. Initially, we built a neural network. We chose to create a neural network for its ability to see intricate patterns in large datasets. However, this approach provided limited success regarding accuracy. This might be attributed to the complexity of the feature set and our specific dataset characteristics.

Sci-kit learn provided general models for several machine learning algorithms. We used the evaluation of these models seen in figure 5 to determine which algorithm to recreate. We found success in optimizing the decision tree algorithm.

The optimized decision tree model achieved the highest accuracy. This success can be attributed to the robustness of the decision tree model and how it can find non-linear relationships between features. This vast difference in accuracy emphasizes the importance of selecting the optimal model to solve the specific challenges of the data at hand.

The next challenge with the neural network is addressing overfitting and underfitting. At first the neural network was overfitting with the validation loss being much greater than the training loss. To strike the right balance between model complexity and preventing overfitting requires techniques like regularization (adding constraints to the network) and early stopping (stopping training before overfitting occurs). Then even more tinkering with the regularization rates to tune to the best performance (lowest loss metric with the highest accuracy).

A. *Optimization:*

As shown within the models array containing the various models and their retrospective model accuracy, precision, and recall, the most consistent and confident model found within all was the decision trees model found within Python. We train our model based upon various models found within python libraries and complete it within these various items. Decision Trees always stood out as one of the best classifiers to choose to model our Intrusion System Model. Now we must utilize a new model that optimizes the prebuilt Decision Tables model created within these items.

The creation of the model begins within the creation within various classifiers, encompassing a spectrum of methodologies ranging from Logistic Regression to Random Trees to Naive Bayes to Decision Trees. Thus we would like to optimize and notice what are the optimal metrics to base the system upon. Next, our system looks towards how our system is able to tune our hyper parameters found within the system. We use the hyperparameter based upon GridSearchCV to find the optimal system found within the creation of the Decision Trees. The GridSearchCV looks over a plethora of hyperparameters found within the Decision Trees, as shown with max_depth, min_samples_split, and min_samples_leaf. Ultimately, adopt this optimized configuration, refining our decision tree model to change within these systems.

Next, the process begins with defining a parameter grid containing hyperparameter to optimize, such as 'max_depth', 'min_samples_split' and 'min_samples_leaf' which respectively denote the maximum depth of the decision tree and the minimum number of samples required to split an internal node and to be at leaf node. Next, employing 'GridSearchCV', an exhaustive search is conducted over the defined hyperparameter grid, utilizing cross-validation to assess each combination's performance. The decision tree classifier serves as the estimator under optimization. Subsequently, the best hyperparameters are identified using 'grid_search.best_params_', forming the foundation for initializing and training an optimized decision tree classifier ('optimized_decision_tree'). The classifier undergoes training on the provided training data ('x_train' and 'y_train'). Finally, the model's performance is evaluated by making predictions on the test data ('x_test') and computing key metrics such as accuracy, precision, recall.

IV. Evaluation

To get a baseline evaluation across all of our models, we computed the accuracy, precision and recall rates depicted in Figure 5.

	Accuracy	Precision	Recall
Logistic Regression	0.842841	0.916692	0.765040
Support Vector Machines linear	0.832195	0.918031	0.749034
Support Vector Machines plonomial	0.885025	0.967563	0.804934
Support Vector Machines RBf	0.857479	0.970549	0.763034
Decision Trees	0.862890	0.906498	0.801293
Random Forest	0.828380	0.968901	0.725108
Naive Bayes	0.608055	0.427247	0.558938
K-Nearest Neighbor	0.838272	0.970240	0.737303

Fig. 5. Table of Evaluation Metrics of Different Models

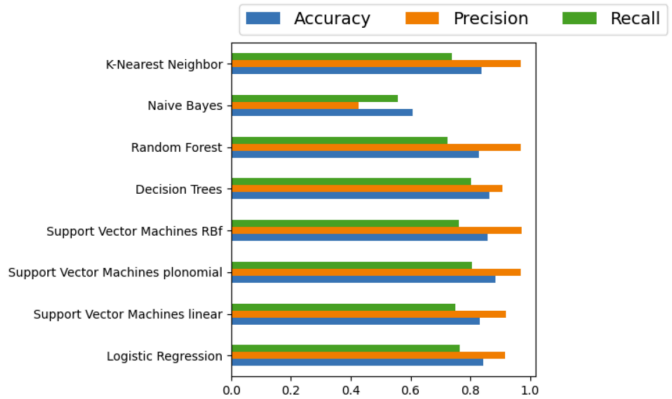

Fig. 6. Evaluation of Various Machine Learning Models

The polynomial support vector machine was the highest performing model in all three metrics of accuracy, precision and recall rate with decision trees trailing as a close second best model. After that all the models performed about equally well with the exception of naive bayes, which we can also see depicted in our ROC curves in the next column.

The ROC curves of all the models (with the exception of naive bayes) take a similar steep slope and level off towards the top of the graph, where again the support vector machines have the steepest ROC curves i.e. the best true positive rate vs false positive rate. Figure 7 show the ROC curves of all the models. Then the decision trees and k-nearest neighbors have a similar performing ROC curve where its not as steep as the support vector machine curves but greatly outperforming the naive bayes method.

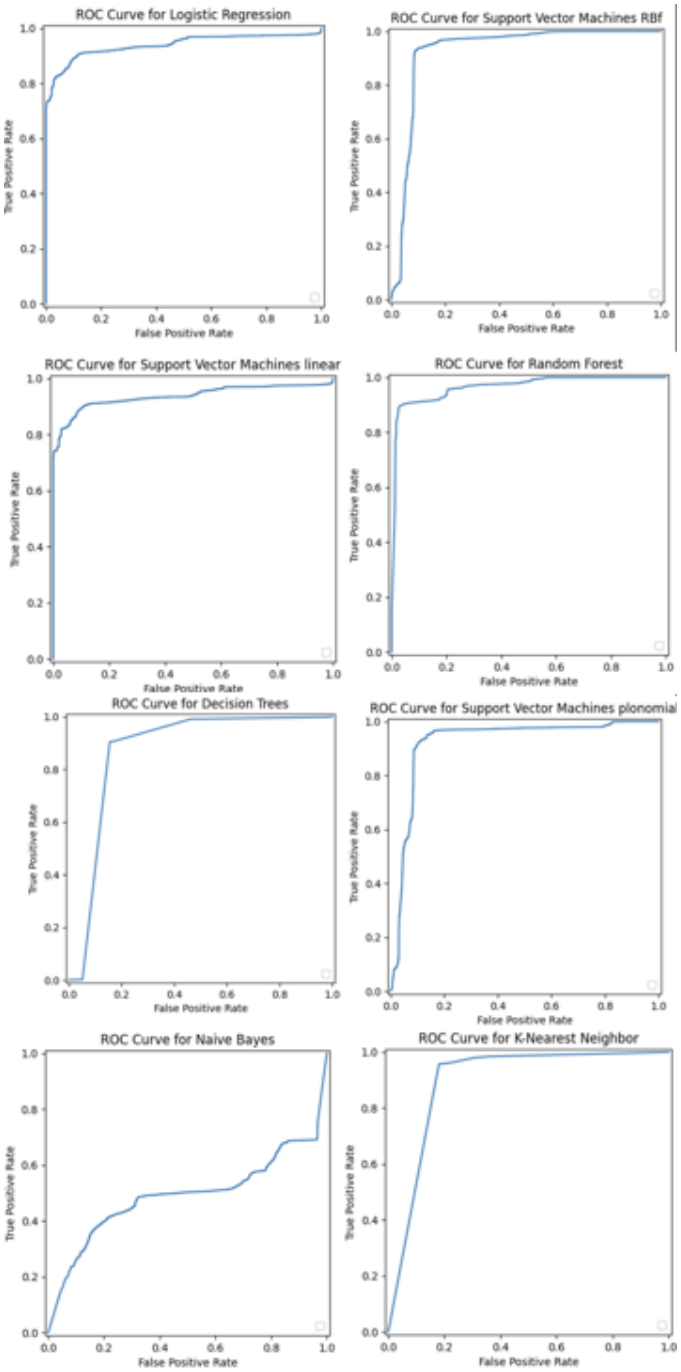

Fig. 7. ROC curves of Different Machine Learning Models

When designing the neural network we went through various testing to create the best performing architecture, which we settled on the following architecture below.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	512
batch_normalization (Batch Normalization)	(None, 128)	512
gaussian_dropout (Gaussian Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
batch_normalization_1 (Bat chNormalization)	(None, 64)	256
gaussian_dropout_1 (Gaussi anDropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65

Total params: 9601 (37.50 KB)
Trainable params: 9217 (36.00 KB)
Non-trainable params: 384 (1.50 KB)

Fig. 8. Neural Network Architecture

Figure 8 shows the architecture of our neural network which is a series of dense layers, with two hidden layers, with the first layer having 128 units, and the second layer 64 units. Both the hidden layers utilize the leaky rectified linear unit function with L1 and L2 regularization to help with overfitting. Then normalization and dropout layers are implemented after each hidden layer to also help with overfitting and making sure no one unit holds too much weight, to help our model generalize. Lastly for the output layer we use the sigmoid activation function since the purpose of the model is to classify if the packets are malicious or not.

We implemented early callbacks as well to the training portion of the neural network, which are useful to prevent overfitting, save training time and optimize our memory and cpu resources.

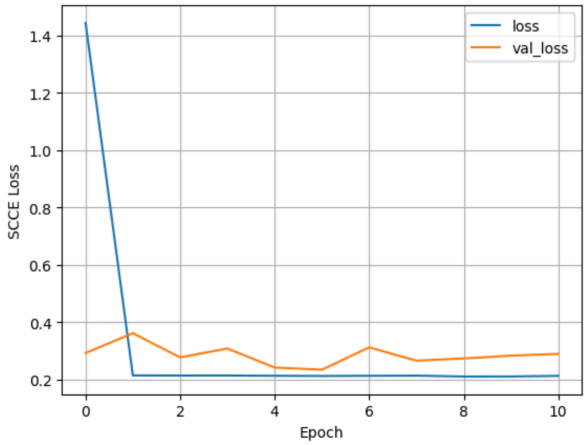

Fig. 9. Validation and Training Loss functions of Neural Network

Figure 9 shows the graph of the validation loss (the orange line) versus the training loss (the blue line), and from the graph we are able to see our model converges fairly quickly where the validation loss, and training loss have a narrow gap.

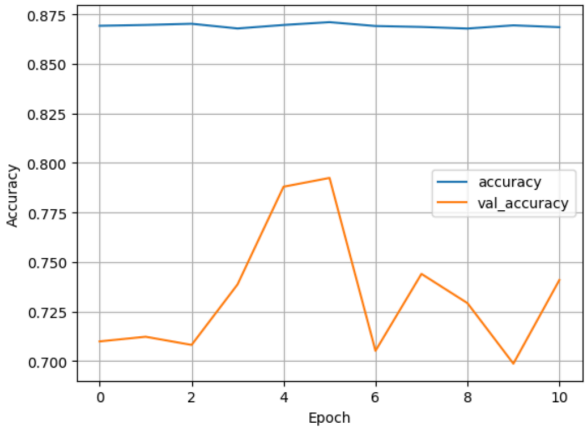

Fig. 10. Validation and Training Accuracy of Neural Network

Figure 10 visualizes the validation accuracy (the orange line) and the training accuracy (the blue line). While it may seem there is a large gap between our accuracies, looking at the y-axis the training accuracy is roughly 0.86 and the validation accuracy is roughly 0.75 which is not nearly as large of a difference compared to the graph. Meaning our neural network was able to generalize fairly well to unseen data being one of our best performing models we have tested.

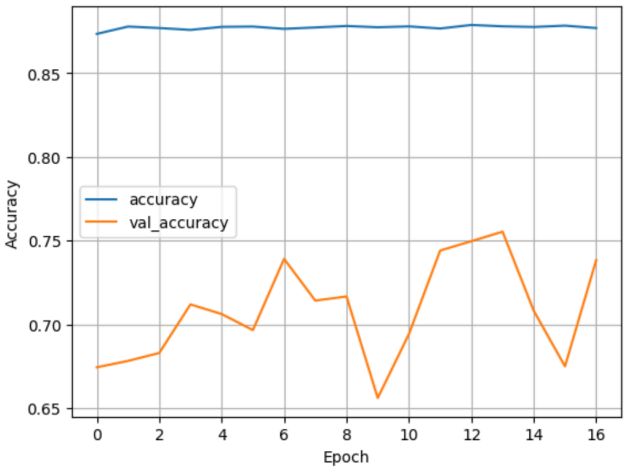

Fig. 11. Validation and Training Accuracy of Neural Network with PCA Input data

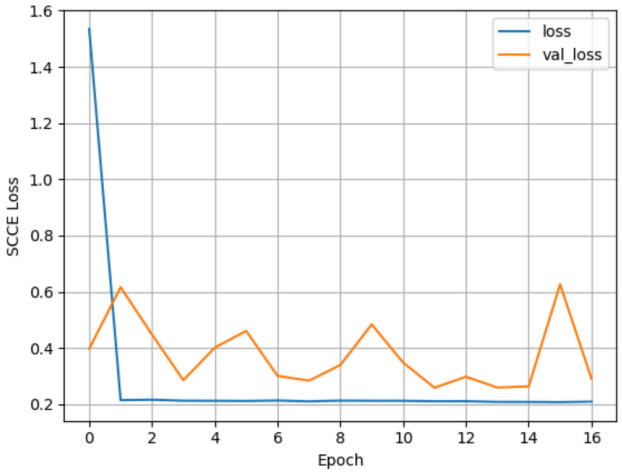

Fig. 12. Validation and Training Loss of Neural Network with PCA Input data

Figure 12 and 13 display the validation and training loss and accuracy of training using the whole dataset rather than the reduced 3-dimension PCA dataset. We can see the whole dataset overall underperforms compared to the PCA dataset where the validation accuracy is roughly a tenth worse and the validation loss is double from the PCA validation loss. So overall the PCA processed data seems to perform better.

V. Conclusion

To conclude, the implementation of an Intrusion Detection System (IDS) is crucial in safeguarding network security amidst the ever-growing threats in the digital landscape. This paper has delved into the heightened importance of IDS in detecting unauthorized access and anomalies in network traffic, thereby preventing potential breaches and ensuring compliance with regulatory standards. By integrating various machine learning techniques, such as Logistic Regression, Random Forest, and Neural Networks, this study aimed to identify optimal models for accurately classifying potential threats within network traffic. The evaluation towards these specific models, conducted using metrics like accuracy, precision, recall, and ROC-AUC scores, provided valuable insights into their performance and efficacy.

Furthermore, data processing played a grand role in this project, where the KDD Cup 1999 dataset was utilized, comprising a vast array of features aimed at predicting network intrusions. Techniques such as Principal Component Analysis (PCA) were employed to streamline data dimensionality and enhance computational efficiency. Visualization of data relationships through what maps and feature plots facilitated a deeper understanding of feature impacts and redundancy, informing preprocessing strategies and model optimization.

Challenges encountered during the study, such the overfitting in neural networks and selecting appropriate models for the dataset's characteristics underscored the importance of iterative experimentation and refinement. Despite initial setbacks, solutions such as regularization and

hyperparameter tuning led to significant improvements in model performance. Evaluation across various machine learning models highlighted the superiority of certain models, with Support Vector Machines and Decision Trees emerging as top performers in accuracy, precision and recall rates.

In the end, this study provides valuable insights into the complex landscape of network intrusion detection, emphasizing the importance of robust methodologies and continuous optimization efforts. By leveraging machine learning techniques and through data analysis organizations can enhance their cybersecurity posture and mitigate the risks posed by malicious activities in network traffic.

VI. Contributions

Adam's Code Contributions: Designed, optimized, and evaluated the neural network, compared neural network performance with PCA input, versus whole dataset input

Ian's Code Contributions: Built and optimized decision trees, Conclusion

Brandon's Code Contributions: Cleaned the data, preprocessed the data, ran evaluation metrics across different models.