Botnet IDS Cybersecurity Capstone Project

Tarleton State University

Mayfield College of Engineering
Department of Computer Science and Electrical Engineering

Ethan Conner, Jared Ricks, Joshua Harrison, Kolbe Williams-Wimmer

5/1/25

Abstract- This project aims to develop and use Machine Learning and Deep Neural Network (DNN) algorithms for our Network Intrusion Detection System (NIDS) to detect botnet traffic in the dataset. Botnets present a serious cybersecurity challenge due to their distributed and adaptive nature, often enabling them to bypass traditional detection methods. This project compares and contrasts how different Machine Learning and DNN techniques can be used to create an effective, scalable solution for botnet detection. This approach will focus on providing a system that can use different techniques to identify malicious network traffic, determine the best algorithm for the dataset, and offer a more efficient and automated alternative to rule-based systems commonly used in existing NIDSs. We will also develop an interface to display all these algorithms and allow for the potential use of different datasets.

Introduction

Scope:

This project delivers a modular NIDS capable of detecting botnet traffic using multiple machine learning and deep neural network algorithms. It includes a Python-based interface that allows users to select different models, visualize results, and possibly test with various datasets (depending on how the dataset is set up). The goal is to create a flexible, testable, and accurate tool for evaluating botnet detection strategies, as well as compare different models to determine which performs the best against the botnet data.

Relevance:

Botnets are especially dangerous because they allow attackers to control large networks of infected devices to launch coordinated attacks, steal data, or disrupt service while remaining hidden. Their evolving nature makes them hard to detect using traditional methods, which is why modern detection systems must adapt quickly. Botnet attacks are becoming more advanced by the day, which makes botnet detection a highly relevant problem in today's digital world.

Background

A network intrusion detection system (NIDS) is a network security device that is designed to monitor traffic on a network and determine if that traffic is normal or malicious. This can be done in several ways, but the approach demonstrated in the project utilizes machine learning to classify the traffic. There are also many ways to build upon an IDS, such as upgrading it to an intrusion prevention system (IPS) by making it drop malicious packets. However, that is not done in this project. An IDS can be implemented at any part of a network as well as on a single machine, making it a host intrusion detection system (HIDS). Furthermore, after malicious network traffic is identified, it is common to log the detection or send out some type of alert. An intrusion detection system can also view traffic as it is passing through the network, making it an in-line IDS, or it can analyze a copy of the traffic, making it a passive IDS.

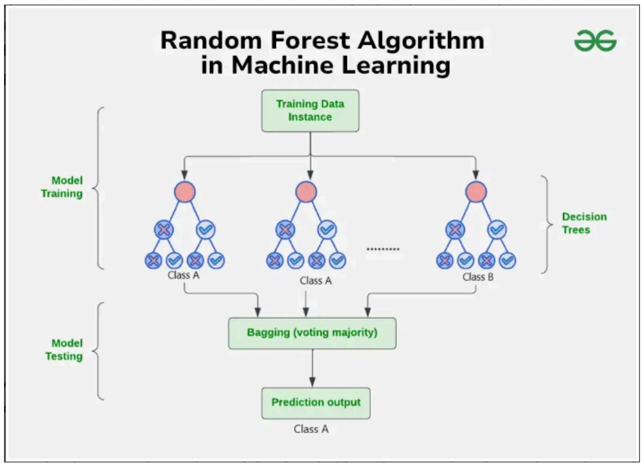
Legacy Systems

The history of the IDS, like many modern technologies, has been rich with change and improvement. It is also similar to many security technologies in that its change has been the byproduct of increasingly sophisticated cyber attacks. Thus, the evolution of the IDS is essentially an arms race against cyber attackers. Most legacy IDS devices utilize signature-based detection, which works well against known attacks but is less successful against zero-day attacks. These legacy systems had to be constantly updated with the newest attack signatures to stay up-to-date. While many modern IDSs still utilize attack signatures, they also often use anomaly-based detection, which is much more successful in identifying zero-day attacks. Anomaly-based detection creates a baseline of what normal traffic should look like, and potentially malicious traffic is determined by how far away from the baseline it is. Thus, the attack can be previously unseen but will still fall outside the baseline and be flagged as malicious. Many of these anomaly-based approaches, as well as the IDS we have created, utilize machine learning algorithms to detect malicious traffic.

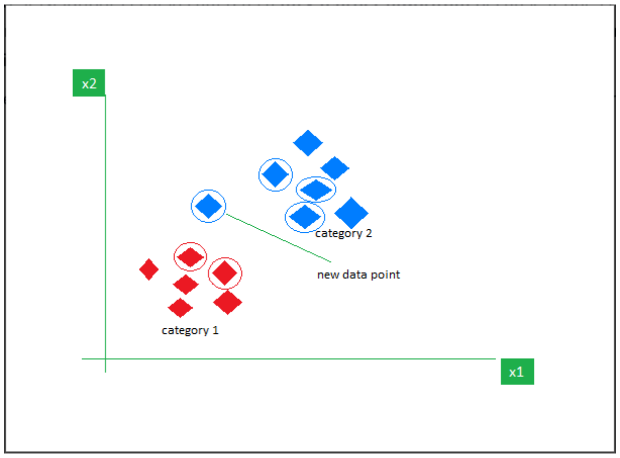
Machine Learning Models

There is a wide range of classification models throughout machine learning, but some are more commonly used than others. There is no one "correct" model or algorithm that can be used to classify any type of data. The best model depends on many things, including how the model processes data, the data being used, and the implementation of the model. Thus, to find out which model is the best, it is important to try many of them. In this project, five machine learning algorithms are implemented, including random forest, K-nearest neighbors (KNN), naive bayes, logistic regression, and support vector machine (SVM). In addition, three deep neural networks were evaluated. The neural networks used are an autoencoder, a recurrent neural network (RNN), and long short-term memory (LSTM). All of these models are explained in more detail below:

Random forest combines multiple decision trees that are created from the training data to predict a label. Each tree predicts the class, and the class with the most predictions from the decision trees wins. Random forest is also an extremely common algorithm implemented in many IDSs.

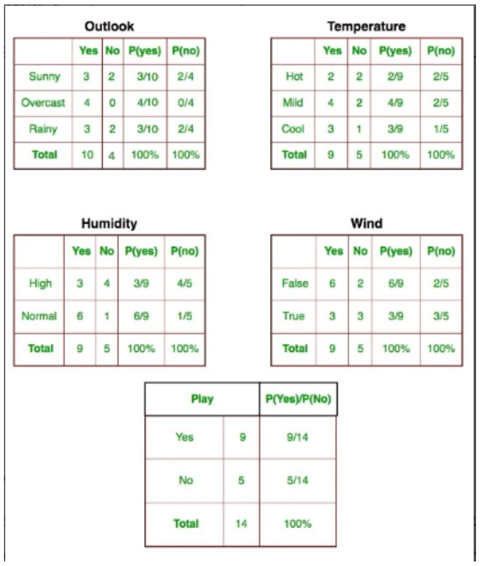


KNN takes each data point to be predicted and calculates the distance from it to every other point in the training data. Then, it finds the closest k points to that point, where k is a hyperparameter defined by the user. In the below example as well as in this project, k is equal to five, so it finds the five closest points and classifies the point as the most common class among those five closest points.



Naive Bayes utilizes Bayesian statistics to calculate the probability that a point to be predicted belongs to each class given its attributes and the training data. The class with the highest probability is the one that is chosen for the point. It gets its name from the fact that it assumes each of the attributes in the dataset is uncorrelated with the others. The example below shows the Bayes formula for calculating the probability that a person will play golf, given the weather and past days of playing and not playing golf.

$$P(No|today) = \frac{P(SunnyOutlook|No)P(HotTemperature|No)P(NormalHumidity|No)P(NoWind|No)P(No)}{P(today)}$$



Outlook

	Yes	No	P(yes)	P(no)
Sunny	3	2	3/10	2/4
Overcast	4	0	4/10	0/4
Rainy	3	2	3/10	2/4
Total	10	4	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

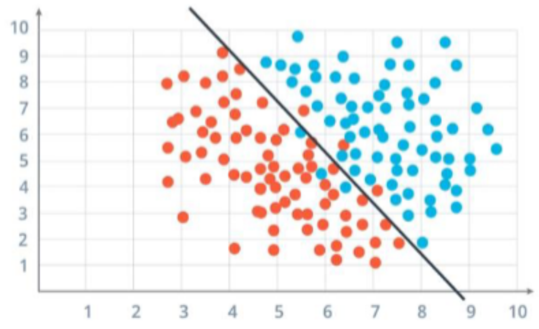
Wind

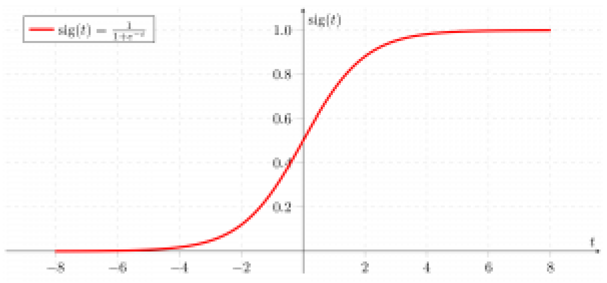
	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

Logistic regression attempts to find a line that separates the classes of each datapoint. Then, the coefficients of that line are used in the sigmoid function to calculate the probability that the point to be predicted is a part of each class. The highest probability class is the one given to the point. If it were multinomial classification instead of binary, the softmax function would be used instead of the sigmoid function.

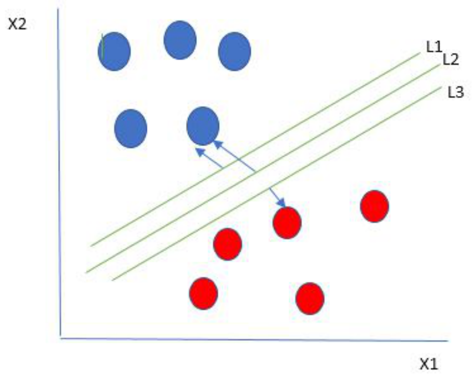
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



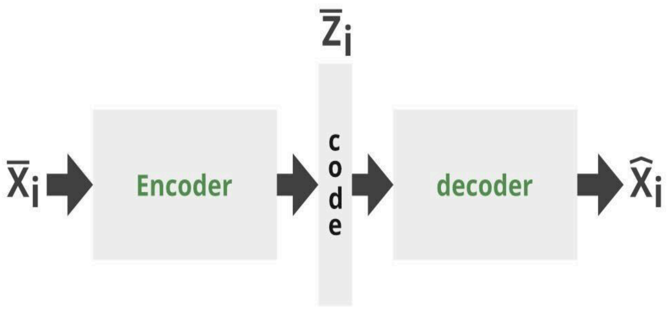


Sigmoid function

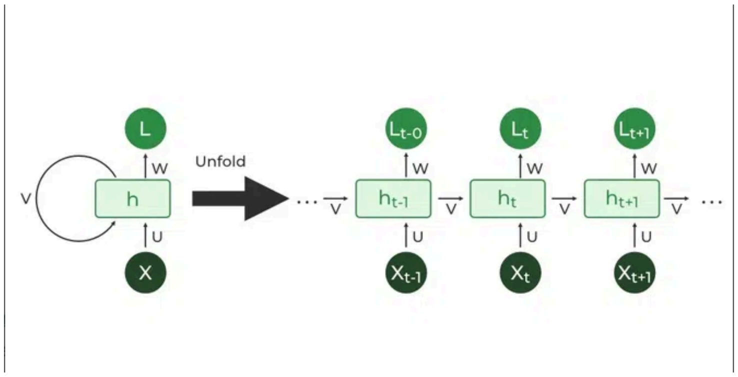
SVM attempts to find a hyperplane that separates the classes and maximizes the margin between itself and the closest datapoints of each class, known as support vectors. Then, a point can be classified based on which side of the hyperplane it lies on. There are multiple kernels that can be used to find the optimal hyperplane. The one in the example below shows a linear kernel for simplicity, but the one used in this project is known as an RBF kernel.



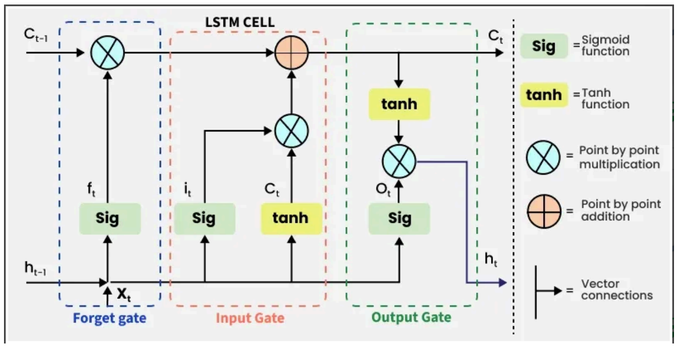
An autoencoder is a type of neural network that attempts to encode input data and reconstruct it to make its predictions. It is only trained on the normal data, and predictions are made based on how well it is able to reconstruct the encoded test points. If it is not able to reconstruct it, it would be classified as an anomaly and would not be classified as a part of the class that it was trained on, which would mean that it would be classified as malicious in this project's implementation.



An RNN is a type of neural network that is designed to handle time-series data. It is designed to remember previous input and make predictions based on current and previous inputs by saving the previous data in something called a hidden state. The network has a loop-like structure where loss is calculated with both the input data and the hidden states to fine-tune its weights and predictions. RNNs often struggle with the vanishing gradient problem, which affects their ability to learn long-term patterns. This issue occurs during training when the gradients computed from the loss become extremely small as they are backpropagated through many time steps. As a result, the weights stop updating effectively, and the network fails to learn dependencies from earlier in the sequence.



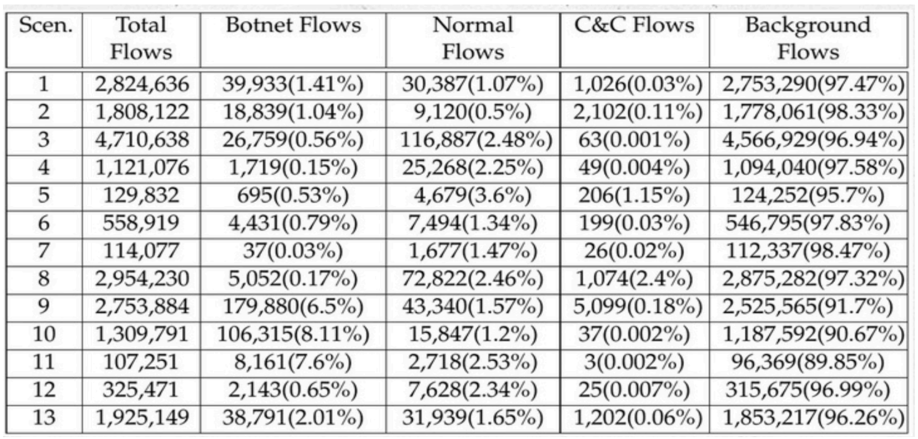
LSTM is a type of RNN that is often used for sequence modeling tasks and seeks to overcome some of the issues brought up by traditional RNNs, namely the vanishing gradient problem. Three gates are used within an LSTM memory cell. The input gate, forget gate, and output gate control what information is added, removed, and output by the cell. A cell state is also used to manage long-term memory over time, which solves the vanishing gradient problem. The cell state and hidden state are output from each LSTM memory cell to be used in the next. The hidden state can then be used to fine-tune weights through back propagation and make predictions.



Methodology

Dataset:

In order to properly evaluate how our models performed against botnet traffic, we trained and tested all of the models with the CTU-13 dataset, which is a dataset that contains network flows from both normal and botnet traffic. It was released in 2011 and widely used as a benchmark for testing botnet detection algorithms. It captures 13 unique scenarios that contain normal and botnet traffic for classification. Some of the botnet attacks it contains are click fraud, spam sending, port scanning, distributed denial-of-service (DDoS), and Command-and-control (C&C) communication. The dataset has 59 features that are used to show the network behavior. Some of the general information about the 13 unique scenarios can be seen below:



Scen.	Total Flows	Botnet Flows	Normal Flows	C&C Flows	Background Flows
1	2,824,636	39,933(1.41%)	30,387(1.07%)	1,026(0.03%)	2,753,290(97.47%)
2	1,808,122	18,839(1.04%)	9,120(0.5%)	2,102(0.11%)	1,778,061(98.33%)
3	4,710,638	26,759(0.56%)	116,887(2.48%)	63(0.001%)	4,566,929(96.94%)
4	1,121,076	1,719(0.15%)	25,268(2.25%)	49(0.004%)	1,094,040(97.58%)
5	129,832	695(0.53%)	4,679(3.6%)	206(1.15%)	124,252(95.7%)
6	558,919	4,431(0.79%)	7,494(1.34%)	199(0.03%)	546,795(97.83%)
7	114,077	37(0.03%)	1,677(1.47%)	26(0.02%)	112,337(98.47%)
8	2,954,230	5,052(0.17%)	72,822(2.46%)	1,074(2.4%)	2,875,282(97.32%)
9	2,753,884	179,880(6.5%)	43,340(1.57%)	5,099(0.18%)	2,525,565(91.7%)
10	1,309,791	106,315(8.11%)	15,847(1.2%)	37(0.002%)	1,187,592(90.67%)
11	107,251	8,161(7.6%)	2,718(2.53%)	3(0.002%)	96,369(89.85%)
12	325,471	2,143(0.65%)	7,628(2.34%)	25(0.007%)	315,675(96.99%)
13	1,925,149	38,791(2.01%)	31,939(1.65%)	1,202(0.06%)	1,853,217(96.26%)

Metrics:

After the dataset has been implemented, each of the models can be evaluated based on its prediction performance with the test data. In this project, four main metrics are used to evaluate each model: accuracy, precision, recall, and F1-score. Accuracy simply represents the percentage of correct predictions. The other metrics are slightly more complex. Precision measures the accuracy of positive predictions, recall measures how many of the true positives were predicted as positive, and F1-Score is the harmonic mean of precision and recall and indicates the balance between the two. The formulas for these are shown below:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

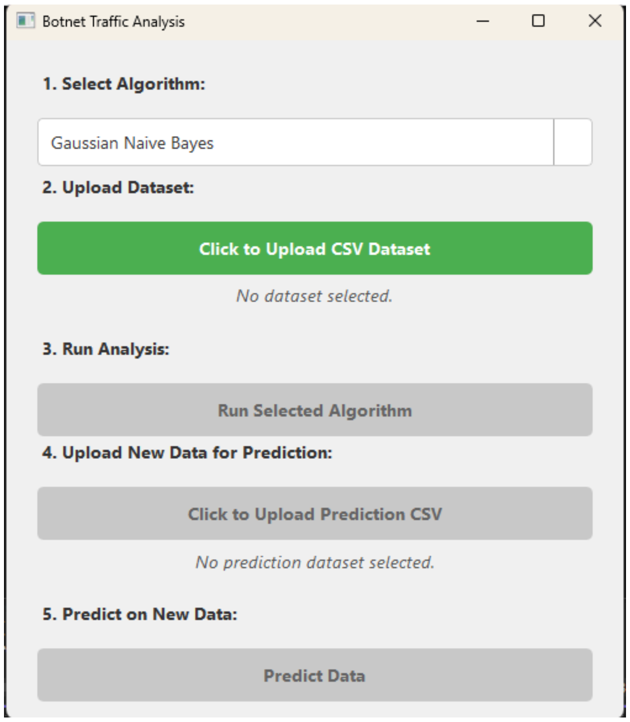
$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Applying the Models:

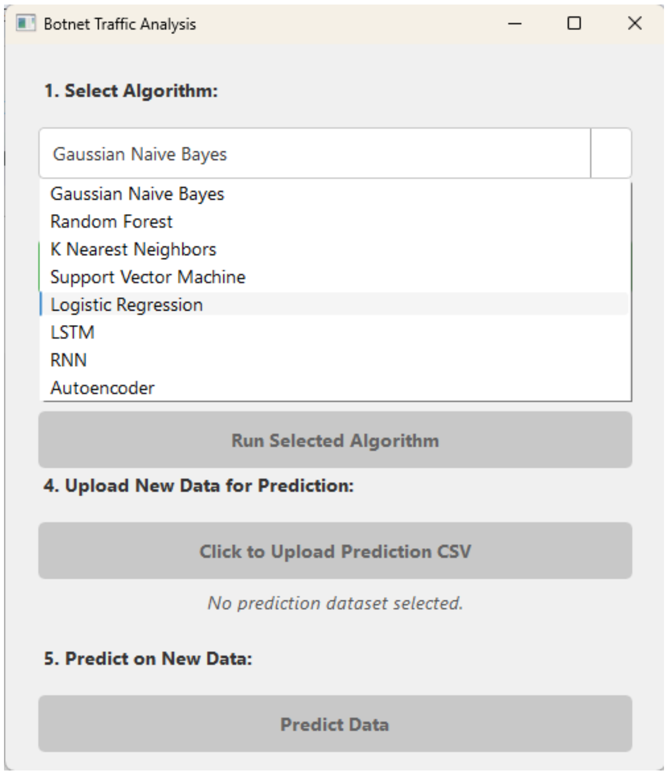
Each of the models in this project uses both training data from the dataset as well as test data to calculate the metrics used to evaluate them. Twenty-five percent of the dataset is used for testing, and the rest is used for training. The data is also preprocessed by ensuring that all of the attributes for each flow in the dataset have a value and by encoding any categorical data. Also, all of the data is scaled to ensure proper data handling. When an algorithm is selected, it is evaluated, and a short report is generated in a results window that contains all of the previously defined metrics as well as a few visuals, including a confusion matrix, receiver operating characteristic (ROC) curve, and learning/loss curves for the DNN models. The confusion matrix simply shows the number of correctly and incorrectly predicted points for each class (normal and attack). The ROC curve plots the true positive rate against the false positive rate, and the learning/loss curves generated for the DNN models plot the accuracy/loss across the epochs used during training. All of these visuals and metrics are saved within a created filesystem, and the weights are also saved for the DNN models. Additionally, the trained models can then be used to predict unknown data.

Using the Application

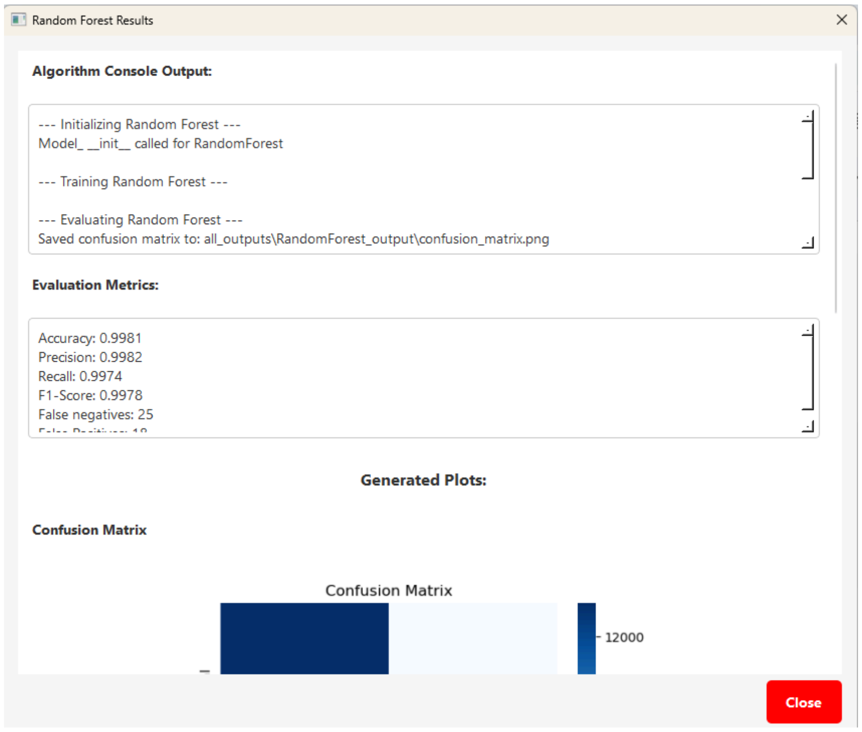
When the program first runs, the following window appears:



The drop-down can then be selected to pick a model to test on the dataset, and the green button is used to upload the dataset:



After an algorithm is selected and the dataset is uploaded, the "Run Selected Algorithm" button is available, which runs the algorithm and displays the results in a separate results window after the algorithm finishes running:



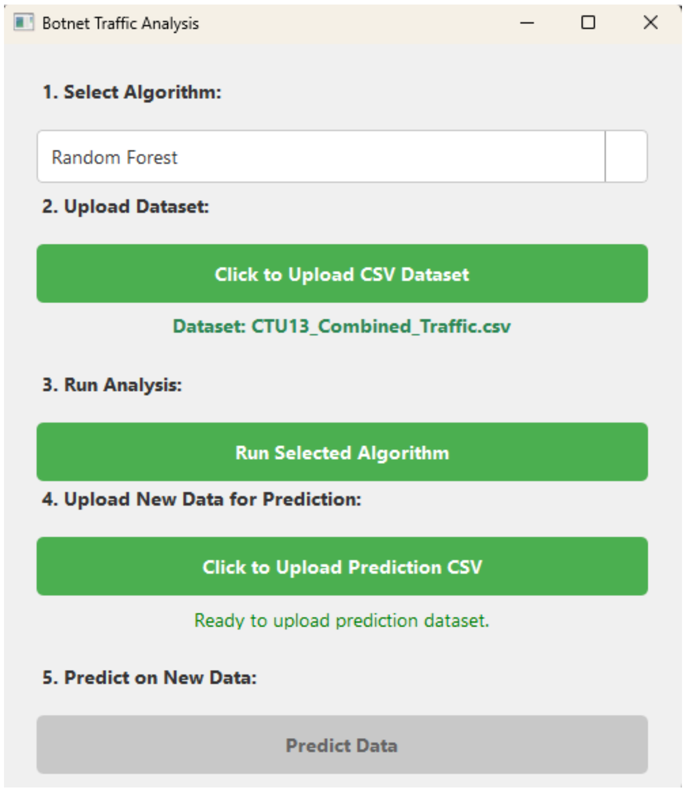
```
--- Initializing Random Forest ---
Model_ __init__ called for RandomForest

--- Training Random Forest ---

--- Evaluating Random Forest ---
Saved confusion matrix to: all_outputs\RandomForest_output\confusion_matrix.png
```

```
Accuracy: 0.9981
Precision: 0.9982
Recall: 0.9974
F1-Score: 0.9978
False negatives: 25
```

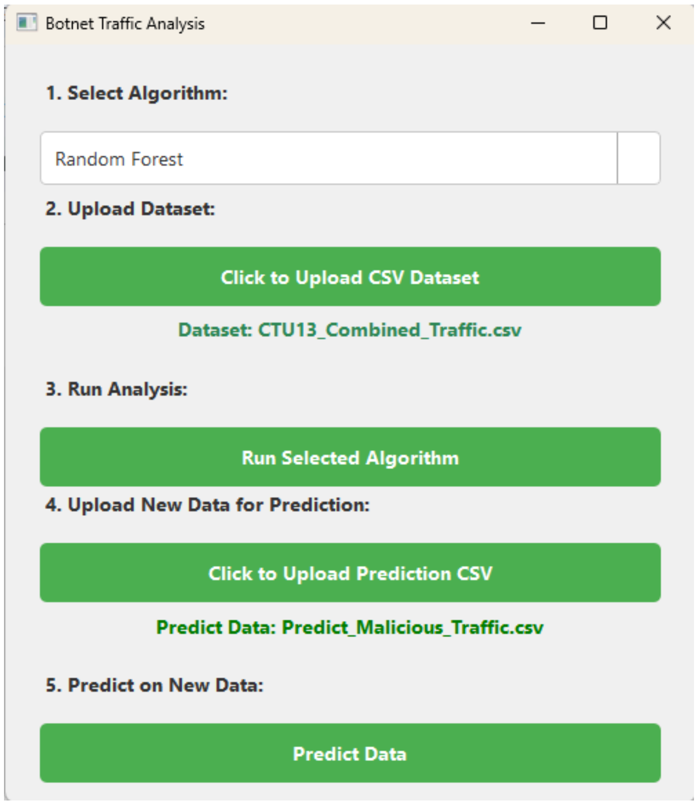
The results window has three sections, each of which holds a piece of the output inside a scrollable box. The first section, titled "Algorithm Console Output," holds information about what occurred during training, testing, and any errors or warnings that occurred. The second section holds metrics, including accuracy, precision, recall, F1-score, false negatives, and false positives. Finally, the third section has any graphs associated with the results of the model. These graphs include a confusion matrix and an ROC curve, and learning and loss curves for the neural network models. After the results window is closed, the original window appears again with the additional ability to upload a dataset to make predictions on. This allows the user to upload their own data to test against the trained model. This could be used in production networks as long as the correct attributes are extracted and placed into a CSV file to upload to the program:



Dataset: CTU13_Combined_Traffic.csv

Ready to upload prediction dataset.

After the data to predict is uploaded, the "Predict Data" button appears and can be clicked to use the trained model to make predictions:


Botnet Traffic Analysis

1. Select Algorithm:

Random Forest

2. Upload Dataset:

Dataset: CTU13_Combined_Traffic.csv

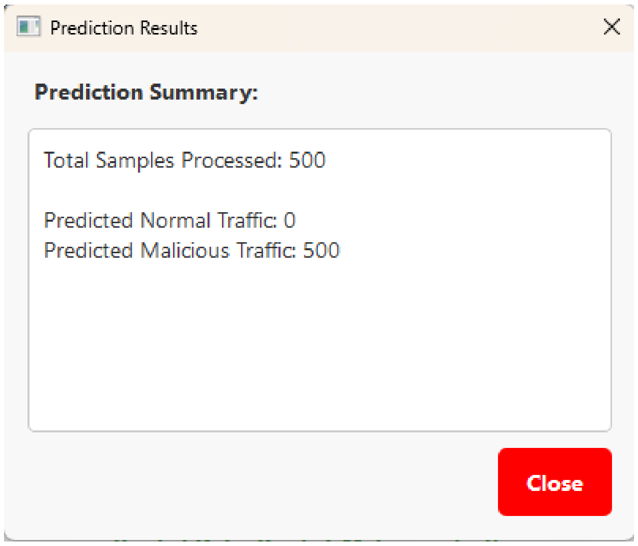
3. Run Analysis:

4. Upload New Data for Prediction:

Predict Data: Predict_Malicious_Traffic.csv

5. Predict on New Data:

Finally, after clicking the "Predict Data" button, another window appears showing the results of the predictions:

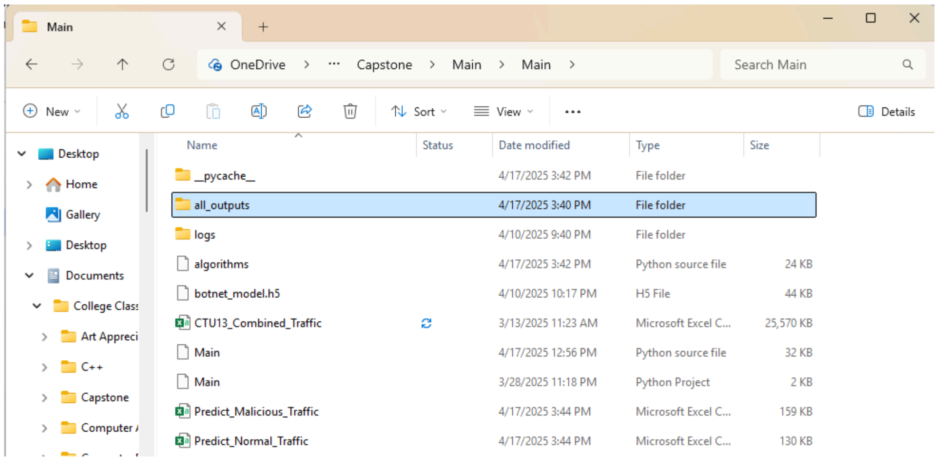

Prediction Results

Prediction Summary:

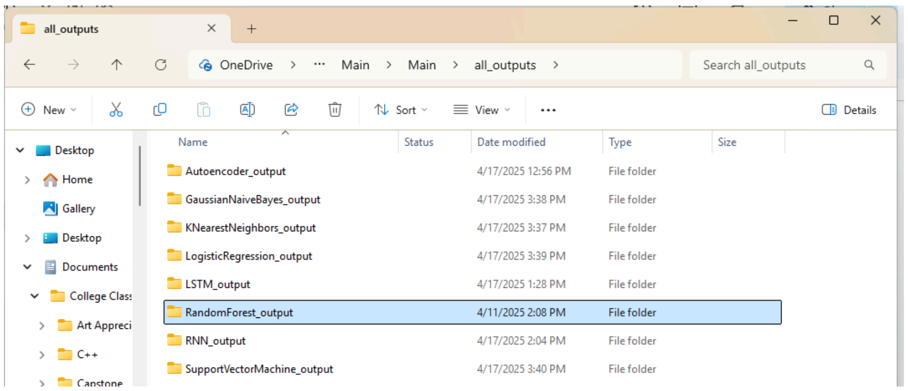
```
Total Samples Processed: 500

Predicted Normal Traffic: 0
Predicted Malicious Traffic: 500
```

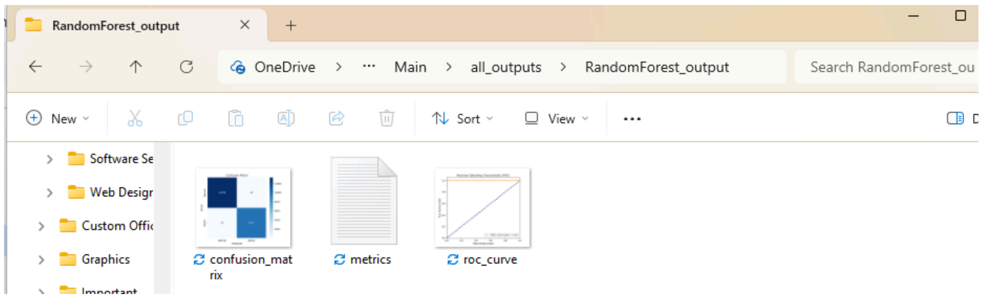
Additionally, more files can be uploaded to be predicted, different models can be trained, and different datasets can be uploaded. After the user is finished running all of their desired models, all of the result metrics and graphs are saved in an output directory for each model:



Name	Status	Date modified	Type	Size
__pycache__		4/17/2025 3:42 PM	File folder	
all_outputs		4/17/2025 3:40 PM	File folder	
logs		4/10/2025 9:40 PM	File folder	
algorithms		4/17/2025 3:42 PM	Python source file	24 KB
botnet_model.h5		4/10/2025 10:17 PM	H5 File	44 KB
CTU13_Combined_Traffic		3/13/2025 11:23 AM	Microsoft Excel C...	25,570 KB
Main		4/17/2025 12:56 PM	Python source file	32 KB
Main		3/28/2025 11:18 PM	Python Project	2 KB
Predict_Malicious_Traffic		4/17/2025 3:44 PM	Microsoft Excel C...	159 KB
Predict_Normal_Traffic		4/17/2025 3:44 PM	Microsoft Excel C...	130 KB



Name	Status	Date modified	Type	Size
Autoencoder_output		4/17/2025 12:56 PM	File folder	
GaussianNaiveBayes_output		4/17/2025 3:38 PM	File folder	
KNearestNeighbors_output		4/17/2025 3:37 PM	File folder	
LogisticRegression_output		4/17/2025 3:39 PM	File folder	
LSTM_output		4/17/2025 1:28 PM	File folder	
RandomForest_output		4/11/2025 2:08 PM	File folder	
RNN_output		4/17/2025 2:04 PM	File folder	
SupportVectorMachine_output		4/17/2025 3:40 PM	File folder	

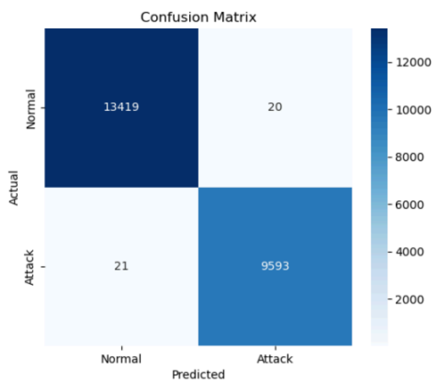


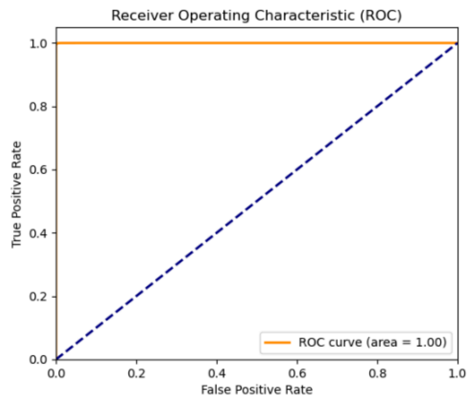
Results

After testing all of the models, some performed better than others. This is to be expected because of how differently each of the algorithms handles the data. Thus, there were a few algorithms that emerged as the best algorithms to use for this particular implementation and likely in similar real-world implementations. The results for each model include the previously mentioned accuracy, precision, recall, and F1-score values, along with the number of false negatives and false positives for each model. False negatives represent the number of test values predicted to be normal traffic that were actually malicious, and false positives represent the number of test values predicted to be malicious that were actually normal. Additionally, a confusion matrix is also shown, which visually shows the number of true and false positives and negatives. An ROC curve is also shown for each model. ROC stands for receiver operating characteristic and is a plot that shows the true positive rate vs the false positive rate for the model at different thresholds. The area under the curve (AUC) represents the model's ability to distinguish between the two classes. Values close to 1 indicate that the model is excellent at distinguishing between the positive and negative classes, meaning it ranks positive examples higher than negative ones with high accuracy. An AUC of 0.5 indicates a model that performs no better than random guessing, as shown by the diagonal line in the plot, which represents the performance of a random classifier. Finally, for the neural network models, learning and loss curves are also included, which show the gradual improvement in performance and loss, respectively, throughout the model's training. These graphs are also mentioned in the "Methodology" section of this report, but we have redefined them here for a more encompassing explanation of the results. The results of each algorithm are shown below:

Random Forest boasted extremely high accuracy as well as very good values in all of the other metrics:

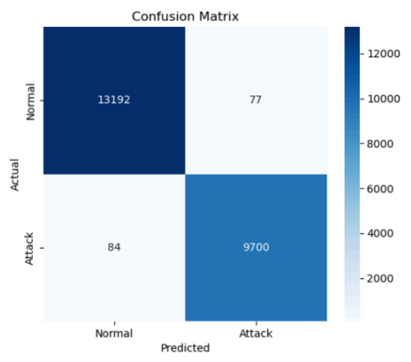
Accuracy: 0.9982
Precision: 0.9979
Recall: 0.9978
F1-Score: 0.9979
False negatives: 21
False Positives: 20

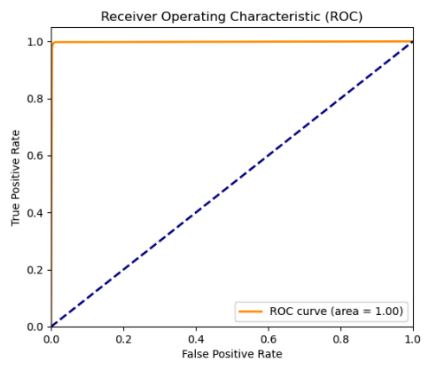




KNN also had very good results, but it still underperformed random forest:

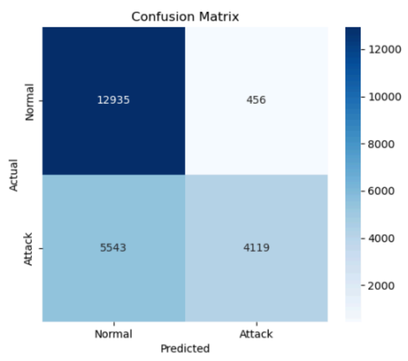
Accuracy: 0.9930
Precision: 0.9921
Recall: 0.9914
F1-Score: 0.9918
False negatives: 84
False Positives: 77

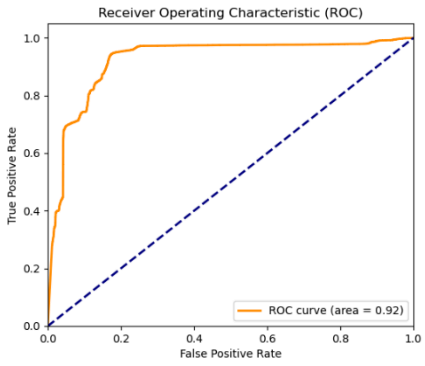




Naive Bayes had trouble classifying true positives and thus did not perform well. This could be due to the fact that naive bayes assumes feature independence, which is rarely true in botnet data:

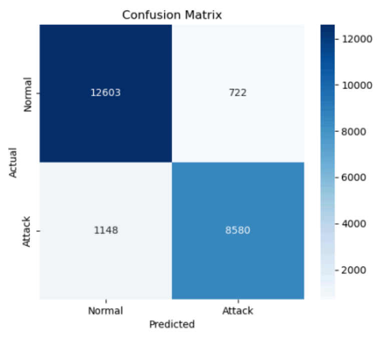
Accuracy: 0.7398
Precision: 0.9003
Recall: 0.4263
F1-Score: 0.5786
False negatives: 5543
False Positives: 456

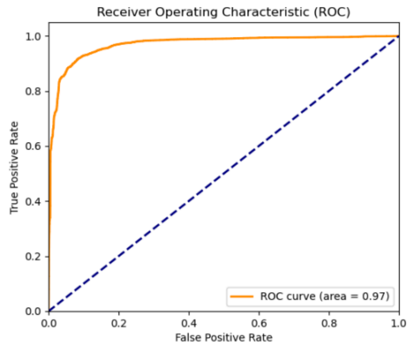




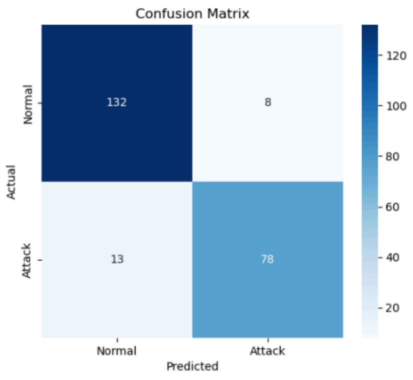
Logistic Regression performed moderately well, but not as well as some of the other models:

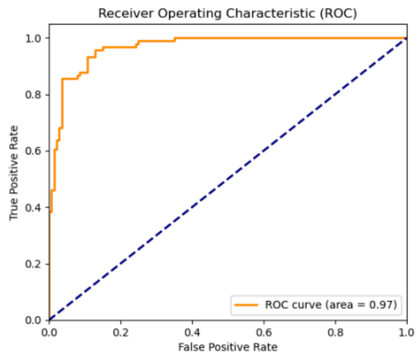
Accuracy: 0.9189
Precision: 0.9224
Recall: 0.8820
F1-Score: 0.9017
False negatives: 1148
False Positives: 722





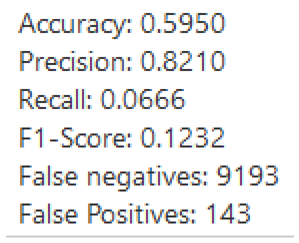
SVM performed similarly to logistic regression. In these results, the model was used with every 100th data point instead of the full dataset because of the model's extensive training time:

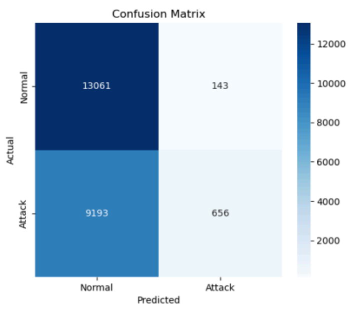


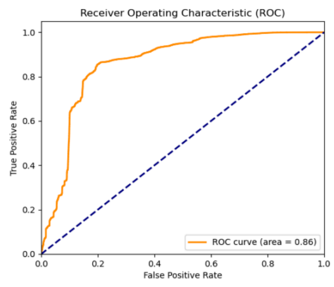


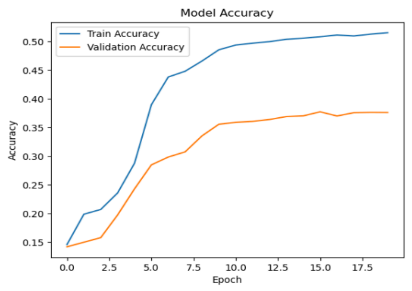
Accuracy: 0.9091
Precision: 0.9070
Recall: 0.8571
F1-Score: 0.8814
False negatives: 13
False Positives: 8

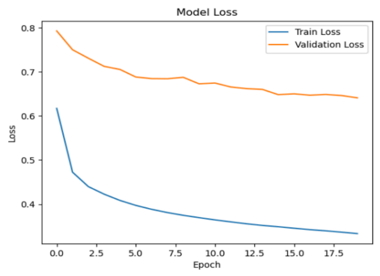
The Autoencoder performed very poorly, which is most likely due to how it makes its predictions. It relies on its inability to reconstruct malicious data as normal data after it has been encoded. Because botnet traffic is very similar to normal traffic in many cases, the autoencoder may have still been able to reconstruct the malicious data similarly to the normal data that it was trained on:


Accuracy: 0.5950
Precision: 0.8210
Recall: 0.0666
F1-Score: 0.1232
False negatives: 9193
False Positives: 143

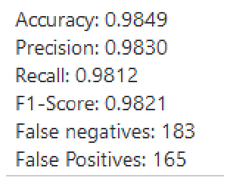


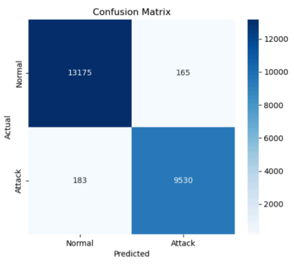


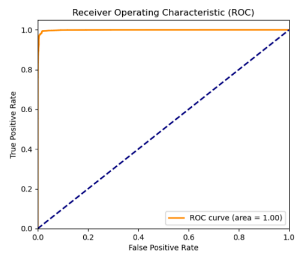


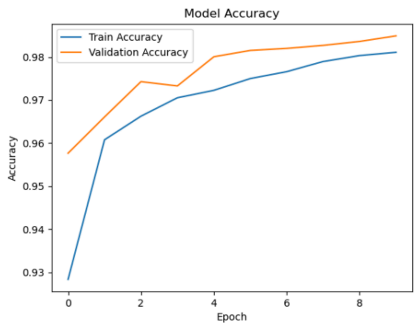


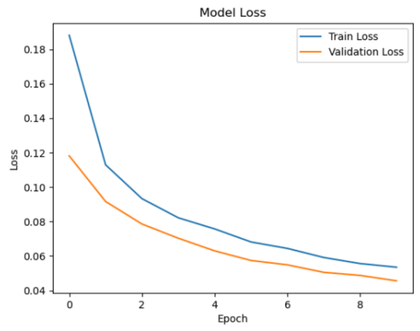
The RNN performed very well. This is likely due to its emphasis on handling time series data, which is common in botnet attacks:


Accuracy: 0.9849
Precision: 0.9830
Recall: 0.9812
F1-Score: 0.9821
False negatives: 183
False Positives: 165



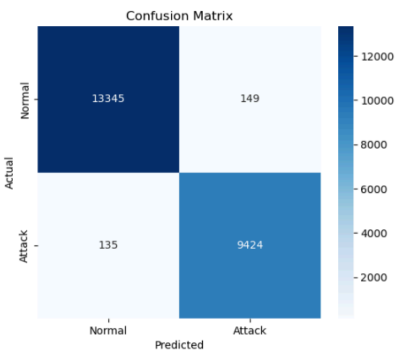


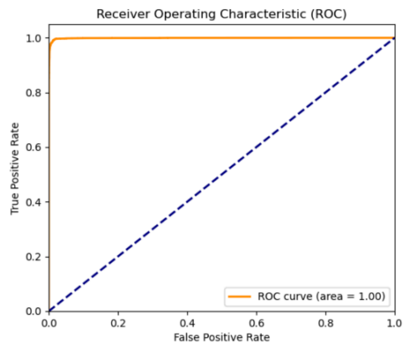


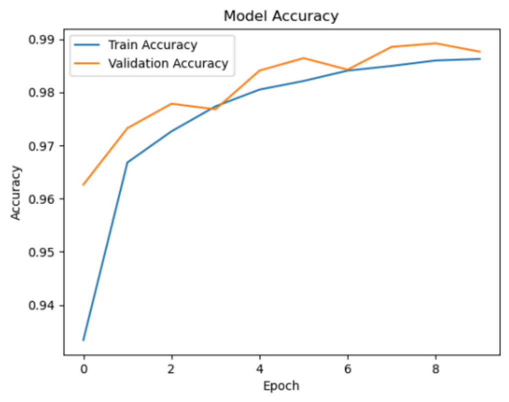


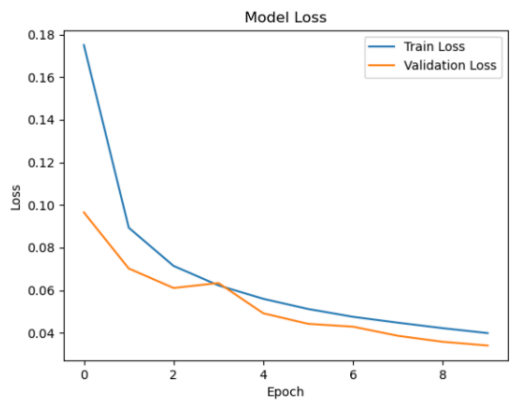
The LSTM model also did very well and slightly outperformed the RNN in most cases. This elevated performance is likely due to LSTM's improvements on classical RNN models, but it also seems like these improvements were not strictly necessary, as the performance was only slightly improved.:

```
Accuracy: 0.9877
Precision: 0.9844
Recall: 0.9859
F1-Score: 0.9852
False negatives: 135
False Positives: 149
```









<u>Results Summarized:</u>

Algorithm:	**Accuracy:**	**Precision:**	**Recall:**	**F1-Score:**
Random Forest	99.82	99.79	99.78	99.79
KNN	99.30	99.21	99.14	99.18
SVM	90.91	90.70	85.71	88.14
Logistic Regression	91.89	92.24	88.20	90.17
Naive Bayes	73.98	90.03	42.63	57.86
Autoencoder	59.50	82.10	0.07	0.12
RNN	98.49	98.30	98.12	98.21
LSTM	98.77	98.44	98.59	98.52

Conclusions

During this research project, we tested 5 different machine learning algorithms as well as 3 different neural networks. These algorithms use a wide variety of ways to classify the traffic as normal or attack. Because of this variety, some performed better than others. Overall, random forest emerged as the most well-suited algorithm for classifying botnet data in our IDS. This conclusion is specific to this dataset, but it is not a surprising discovery since random forest is often used in IDSs. Another interesting conclusion that can be drawn from our research is that the neural networks did not outperform the machine learning algorithms as well as some might have expected. Our implementation of LSTM and RNN did outperform the majority of the machine learning algorithms, but neither was as good as random forest. This highlights the fact that simpler machine learning algorithms can be highly effective for classifying network traffic as malicious. Algorithms such as random forest may be the preferred method over neural networks, which are typically much more computationally expensive.

References

GeeksforGeeks. (2024, December 18). Machine learning algorithms. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning-algorithms/

GeeksforGeeks. (2025, February 25). Types of neural networks. GeeksforGeeks. https://www.geeksforgeeks.org/types-of-neural-networks/

Malik, F. (n.d.). CTU13-CSV-dataset/readme.md at main · imfaisalmalik/CTU13-CSV-dataset. GitHub. https://github.com/imfaisalmalik/CTU13-CSV-Dataset/blob/main/README.md

Saraalhatem. (2023, November 19). Autoencoder-for-CCFD. Kaggle. https://www.kaggle.com/code/saraalhatem/autoencoder-for-ccfd

scikit-learn. scikit. (n.d.). https://scikit-learn.org/stable/

Tensorflow. TensorFlow. (n.d.). https://www.tensorflow.org/

V. R, P. C. A and V. M, "A Comprehensive Analysis of Intrusion Detection System using Machine Learning and Deep Learning Algorithms," 2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS), Hassan, India, 2024, pp. 1-5, doi: 10.1109/IACIS61494.2024.10721636.