Botnet Classification using Machine Learning and DNN Algorithms NIDS

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

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, which often enables them to bypass traditional detection methods. Comparing and contrasting 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, offering a more efficient and automated alternative to rule-based systems commonly used in existing NIDS. We will also develop an interface to display all of these algorithms and also allow for the potential use of different datasets.

Introduction Page

Scope:

- Modular Network Intrusion Detection System (NIDS) for detecting botnet traffic
- Utilizes multiple machine learning and deep neural network algorithms
- Python-based GUI includes:
  - ➢ Model selection options
  - ➢ Visualization of results
  - ➢ Support for testing with various datasets (dataset-dependent)
- Designed to be flexible, testable, and accurate
- Primary goal: evaluate and improve botnet detection strategies

Relevance:

- Botnets pose a major threat by enabling attackers to:
  - ➢ Control large networks of infected devices
  - ➢ Launch coordinated attacks
  - ➢ Steal sensitive data
  - ➢ Disrupt services
- Botnets:
  - ➢ Operate stealthily, making detection difficult
  - ➢ Constantly evolve, bypassing traditional detection methods
  - ➢ Modern detection systems must adapt quickly to stay effective

Legacy Systems

IDS (Intrusion Detection Systems) have evolved significantly due to increasingly sophisticated cyber attacks
The evolution of IDS mirrors an arms race between defenders and attackers
Legacy IDS primarily used **signature-based detection**:

- Effective against known threats
- Ineffective against zero-day attacks
- Required constant updates with new attack signatures

Modern IDS often combine:

- **Signature-based detection** for known threats
- **Anomaly-based detection** for identifying unknown or zero-day attacks

Anomaly-based methods establish a baseline of normal behavior to detect deviations
Many modern IDS, including this project, use **machine learning algorithms** for detecting malicious traffic

CTU-13 Dataset

CTU-13 dataset:
- Released in **2011**
- Widely used as a **benchmark** for testing botnet detection algorithms

Contains **13 distinct scenarios** with a mix of:
- **Normal traffic**
- **Botnet traffic**

Includes various botnet behaviors such as:
- Click fraud
- Spam sending
- Port scanning
- Distributed Denial-of-Service (DDoS)
- Command-and-Control (C&C) communication

Composed of **59 features** representing network behavior
Used to **train machine learning models** for botnet detection

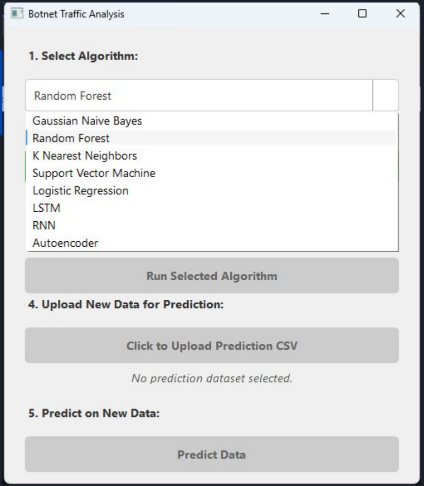
Scen.	Total Flows	Botnet Flows	Normal Flows	C&C Flows	Background Flows
1	2,824,636	39,933(1.41%)	30,387(1.07%)	1,026(0.03%)	2,753,290(97.47%)
2	1,808,122	18,839(1.04%)	9,120(0.5%)	2,102(0.11%)	1,778,061(98.33%)
3	4,710,638	26,759(0.56%)	116,887(2.48%)	63(0.001%)	4,566,929(96.94%)
4	1,121,076	1,719(0.15%)	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%)

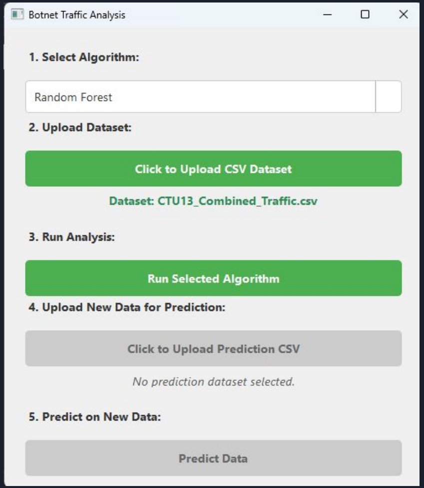
Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	2,824,637	52GB	Neris	1
2	4.21	71,851,300	1,808,123	60GB	Neris	1
3	66.85	167,730,395	4,710,639	121GB	Rbot	1
4	4.21	62,089,135	1,121,077	53GB	Rbot	1
5	11.63	4,481,167	129,833	37.6GB	Virut	1
6	2.18	38,764,357	558,920	30GB	Menti	1
7	0.38	7,467,139	114,078	5.8GB	Sogou	1
8	19.5	155,207,799	2,954,231	123GB	Murlo	1
9	5.18	115,415,321	2,753,885	94GB	Neris	10
10	4.75	90,389,782	1,309,792	73GB	Rbot	10
11	0.26	6,337,202	107,252	5.2GB	Rbot	3
12	1.21	13,212,268	325,472	8.3GB	NSIS.ay	3
13	16.36	50,888,256	1,925,150	34GB	Virut	1

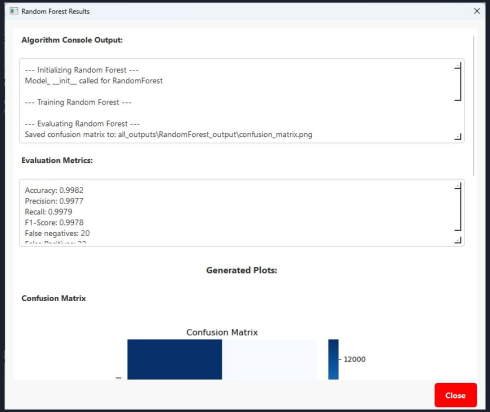
Table 2 – Characteristics of the botnet scenarios. (CF: ClickFraud, PS: Port Scan, FF: FastFlux, US: Compiled and controlled by us.)

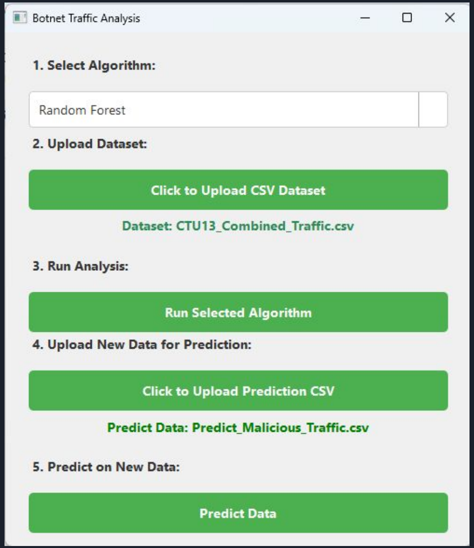
Id	IRC	SPAM	CF	PS	DDoS	FF	P2P	US	HTTP	Note
1	√	√	√							
2	√	√	√							
3	√			√				√		
4	√				√			√		UDP and ICMP DDoS.
5		√		√					√	Scan web proxies.
6				√						Proprietary C&C. RDP.
7									√	Chinese hosts.
8				√						Proprietary C&C. Net-BIOS, STUN.
9	√	√	√	√						
10	√				√			√		UDP DDoS.
11	√				√			√		ICMP DDoS.
12							√			Synchronization.
13		√		√					√	Captcha. Web mail.

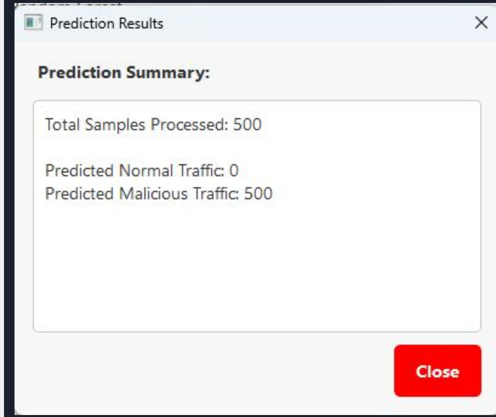
User Interface











Metrics Used

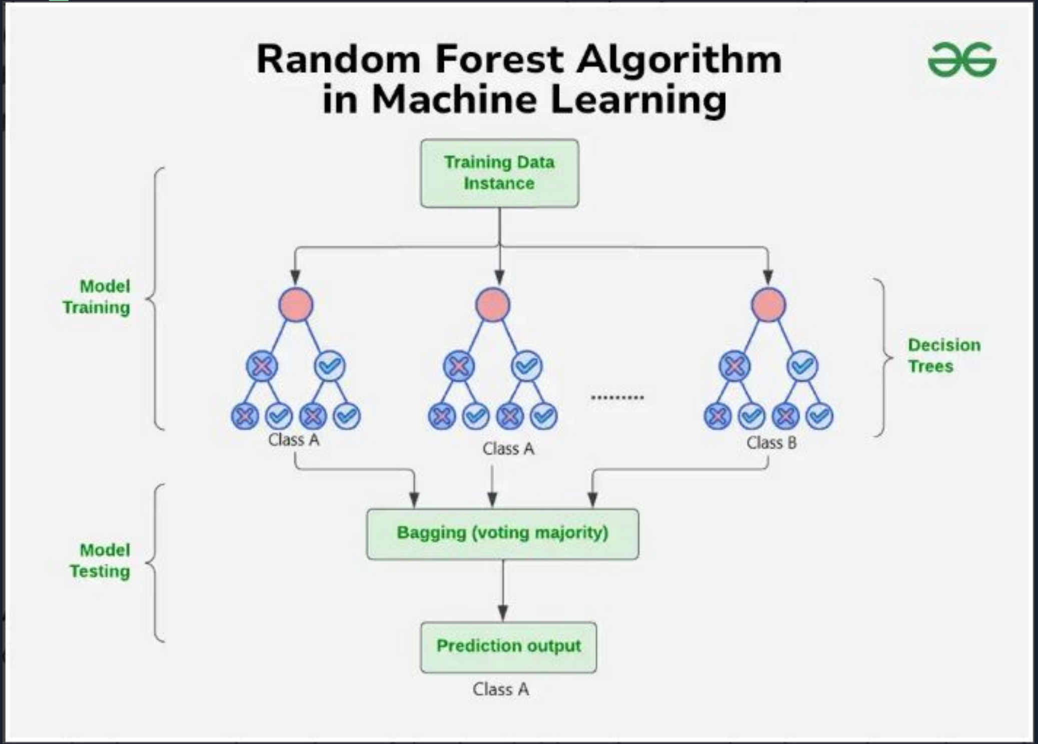
$$\text{Precision} = \frac{TP}{TP + FP}$$

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

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

- The metrics that we used to evaluate each model are precision, recall, and F1-Score.
- Precision measures the accuracy of positive predictions.
- Recall measures how many of the true positives were predicted as positive.
- F1-Score is the harmonic mean of precision and recall and indicates the balance between the two.

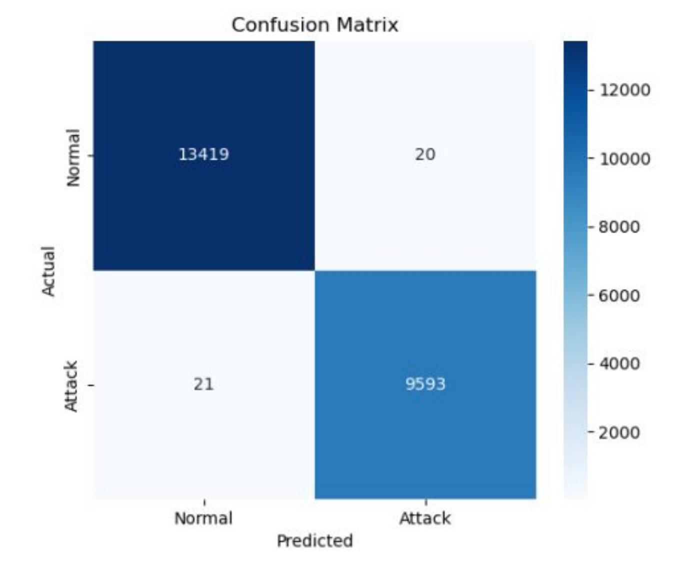
Random Forest Explained

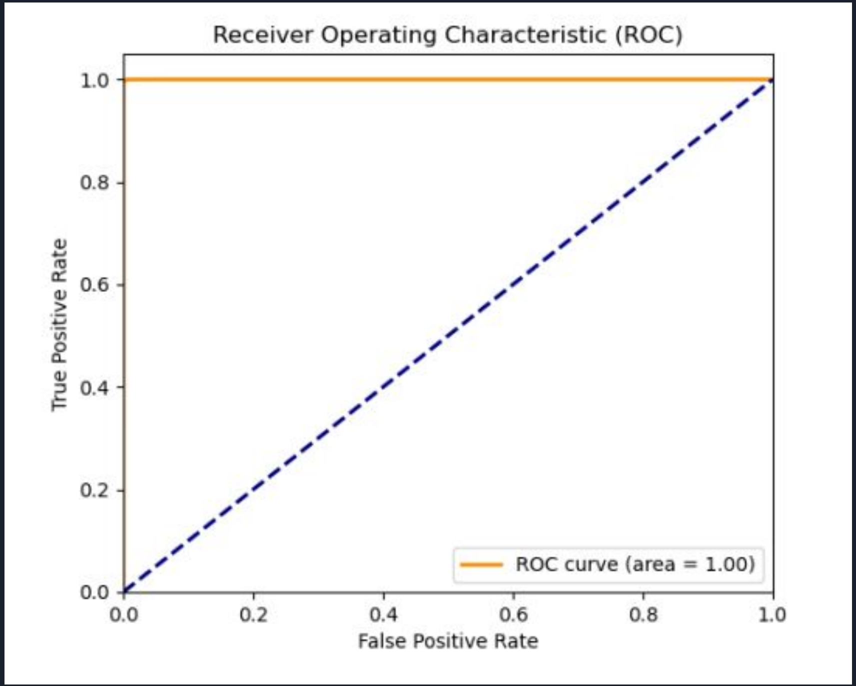


- Random forest is a machine learning algorithm that combines multiple decision trees to classify data.
- Random forest is an extremely common machine learning algorithm used in IDSs because of its high accuracy in this type of classification.

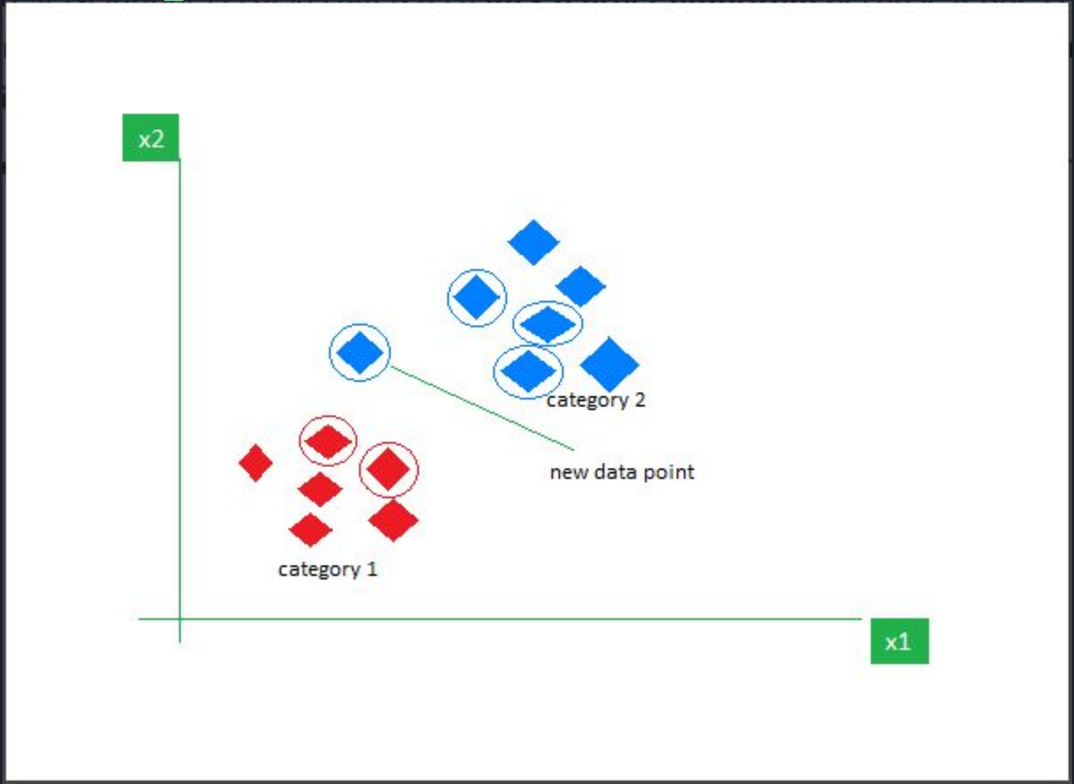
Random Forest Results

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





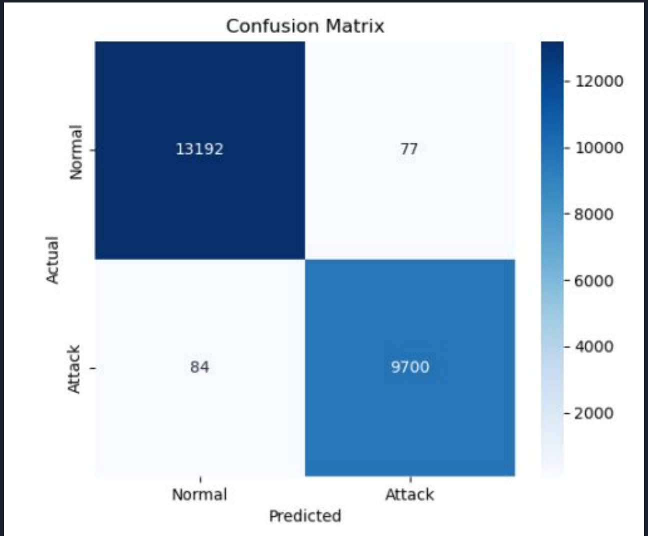
KNN Explained

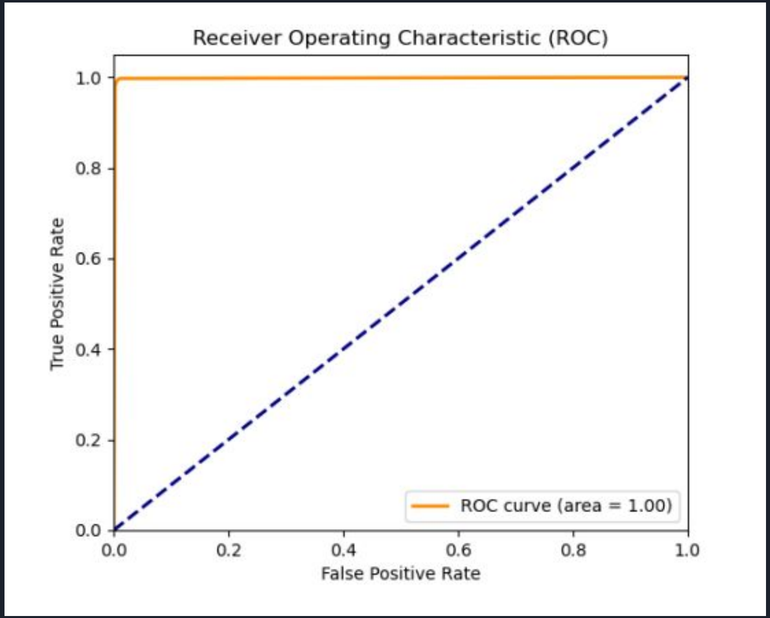


- KNN is a supervised machine learning algorithm that uses distances to classify points.
- KNN takes the euclidean distances of the nearest neighbors.
- In our case KNN takes the 5 closest points to our new data point
- The class that appears the most in those 5 will be the predicted class of our new data point.

KNN Results

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





Naive Bayes Explained

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

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

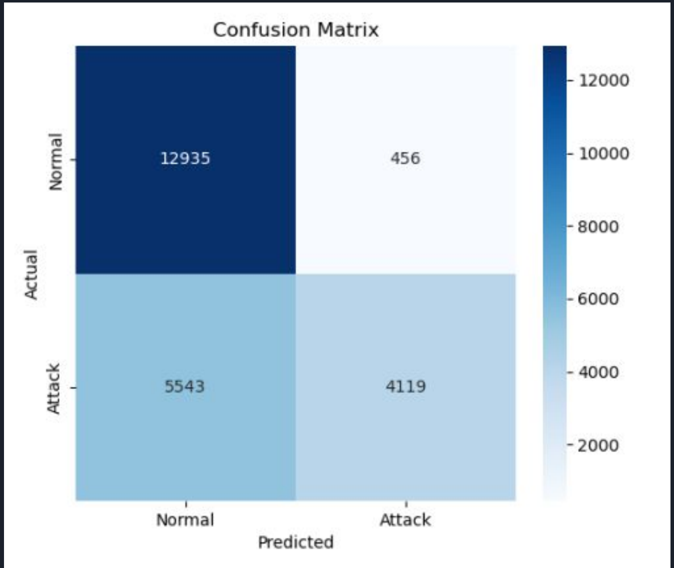
	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

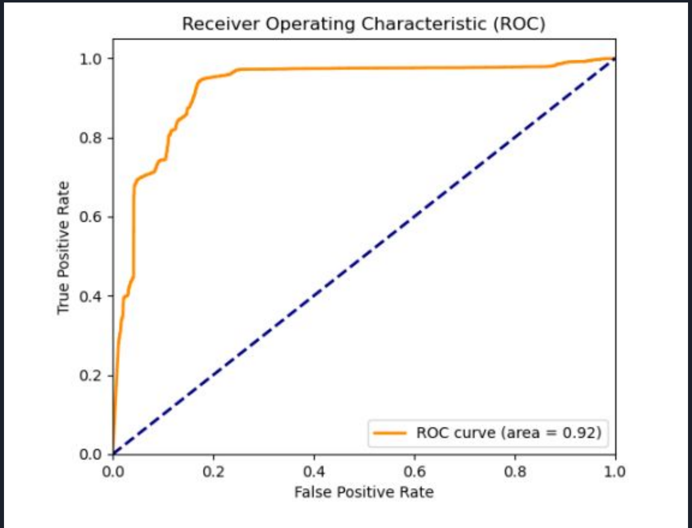
Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

- Naive Bayes is a supervised machine learning algorithm that uses probabilities to classify where new data points belong.
- The algorithm applies Bayes' theorem to calculate the new probability based on the prior probability.
- Once the computations are complete the highest probability is chosen.
- This example show how the probability for playing golf on a given day can be calculated using bayesian statistics
- Our implementation achieved a relatively low accuracy, which could be due to the fact that Naive bayes assumes feature independence, which is rarely true with botnet data.

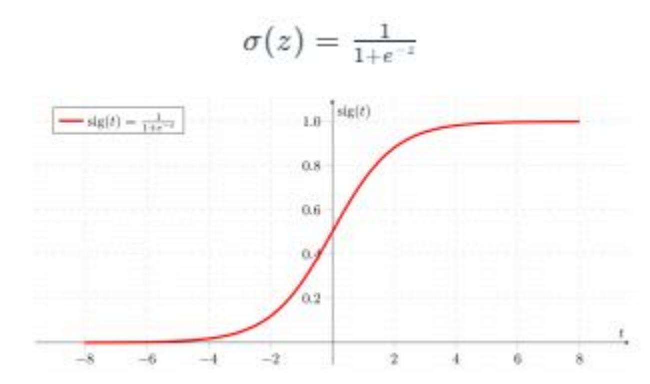
Naive Bayes Results

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





Logistic Regression Explained

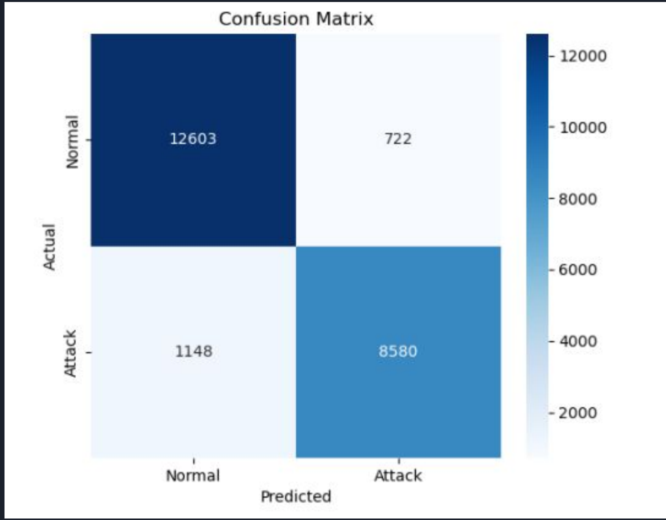


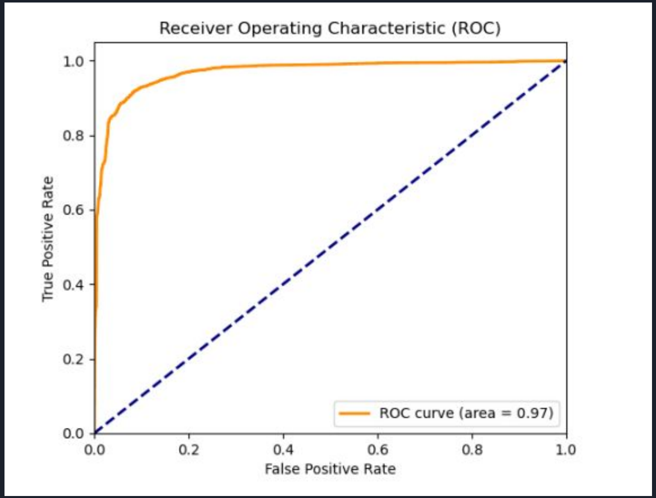
Sigmoid function

- Logistic Regression is a supervised machine learning algorithm that can be used for binary classification.
- Logistic Regression uses weights that are assigned to different features to classify the data.
- The weights define a decision boundary that separates the space into regions where different classes are more probable.
- It uses the sigmoid function to retrieve either 0 or 1 for classification.

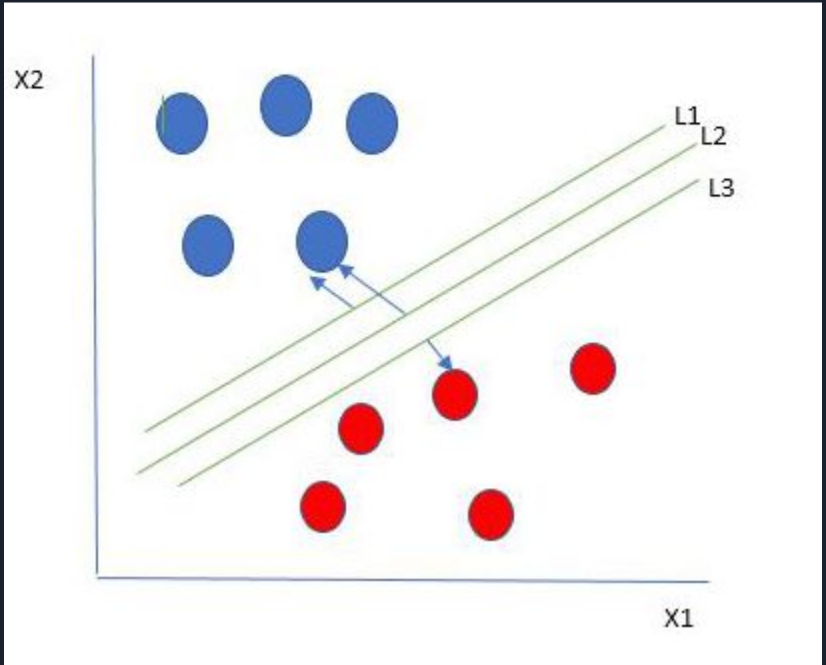
Logistic Regression Results

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





SVM Explained



- SVM is a supervised machine learning algorithm used for classification by finding the optimal hyperplane that separates different classes.

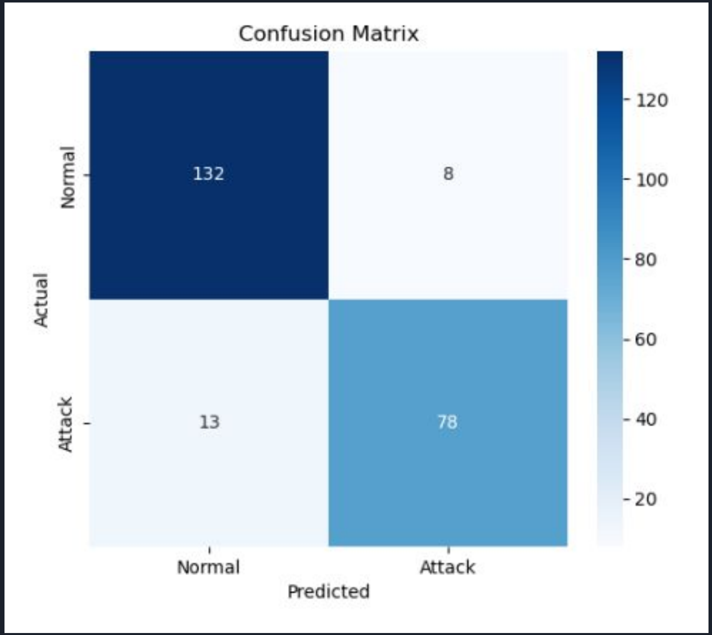
- It works by maximizing the margin between the closest points of each class, known as support vectors.

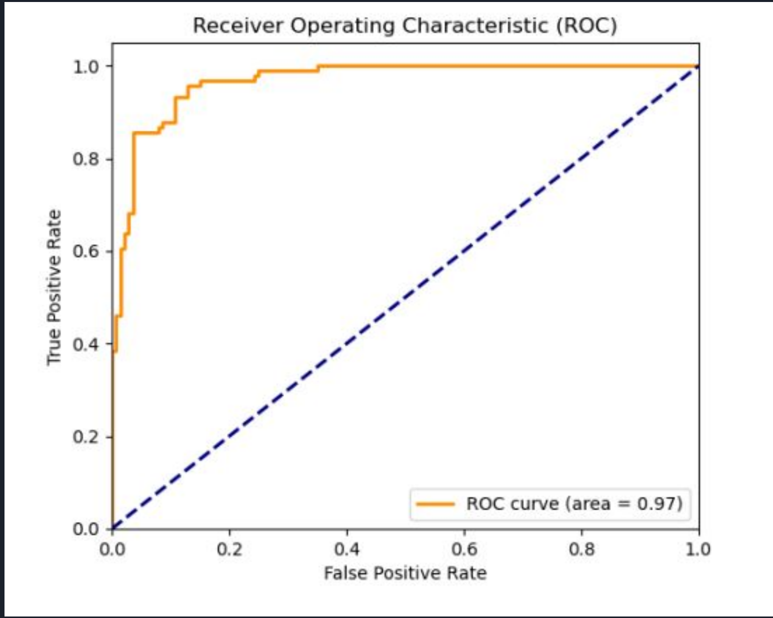
- In our case, we used an RBF kernel to handle non-linear relationships in the data.

- Points are classified based on which side of the hyperplane they lie

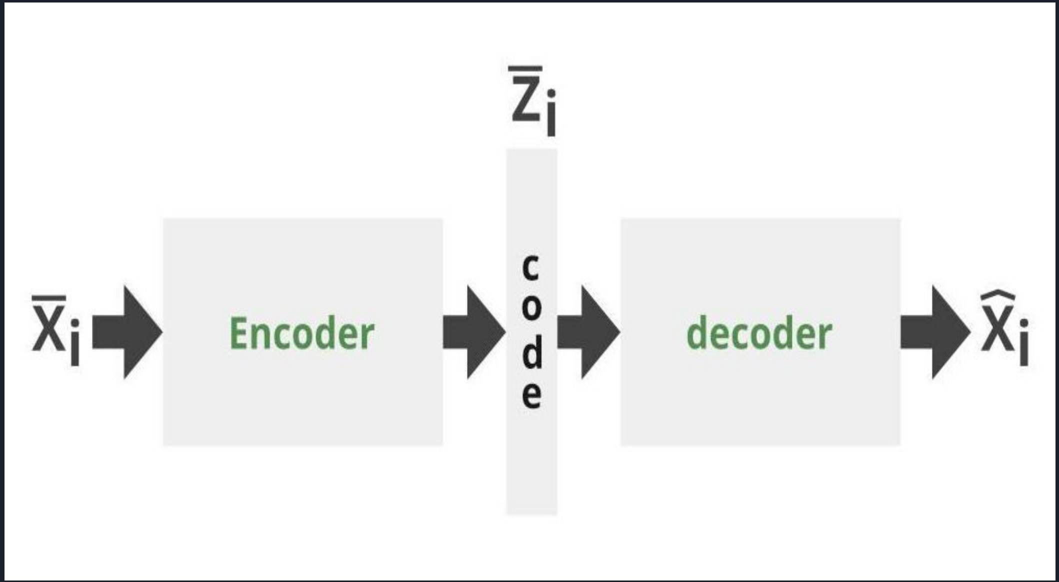
SVM Results

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



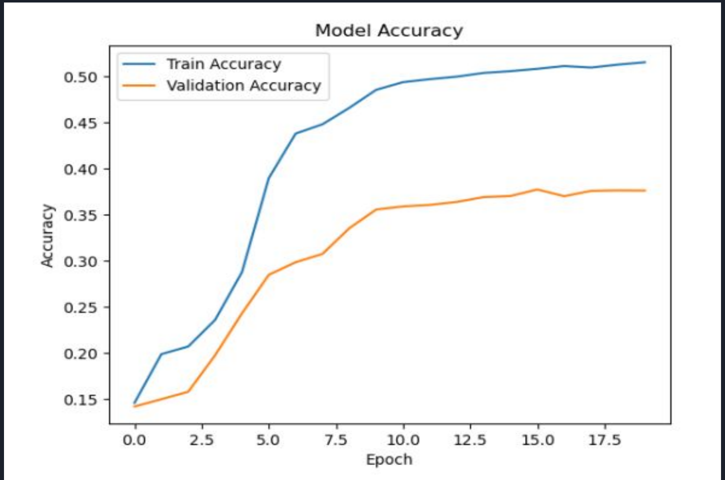


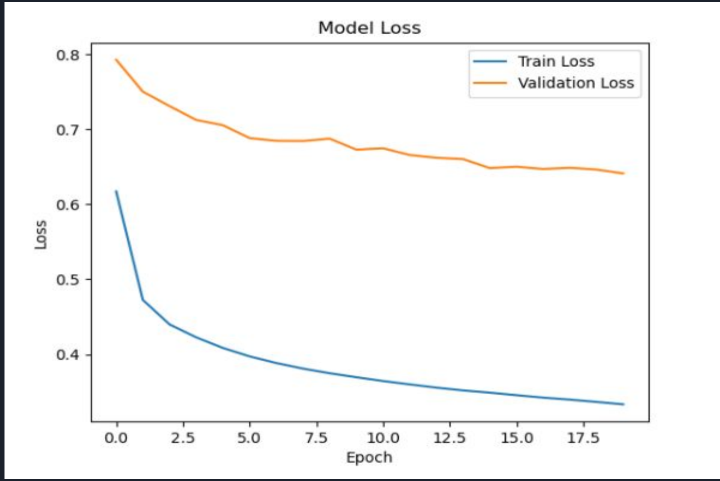
Autoencoder Explained



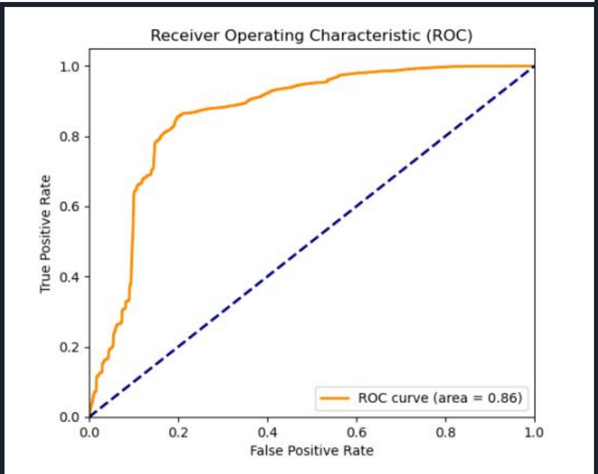
- An autoencoder is a type of neural network that attempts to encode input data and reconstruct it to make its predictions.
- Our model was trained on only normal data, which established a baseline for anomaly detection
- Autoencoders typically do not reconstruct anomalies well, which indicates that the traffic could be malicious.
- In our implementation, the model had a relatively low accuracy score. This is likely due to the fact that botnet traffic is very similar in many ways to normal traffic, which means the encoder was able to reconstruct malicious traffic easily and did not allow for anomaly detection.

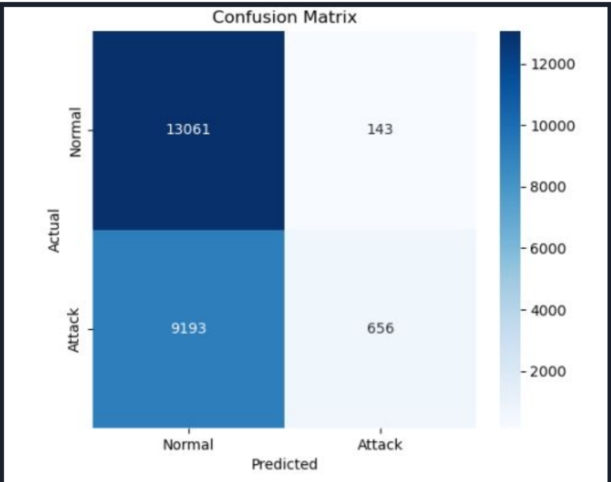
Autoencoder Results





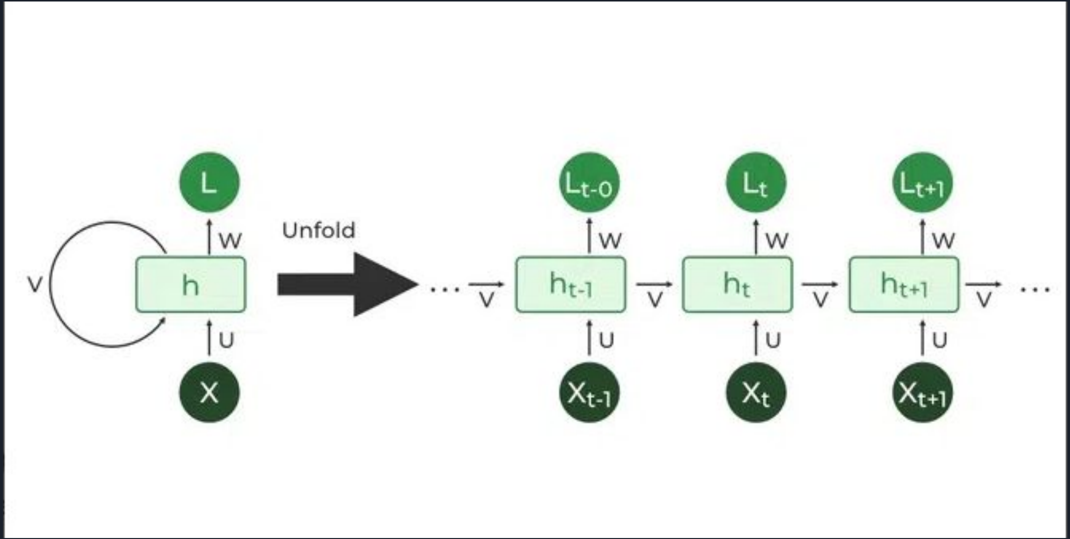
Accuracy: 0.5950
Precision: 0.8210
Recall: 0.0666
F1-Score: 0.1232
False negatives: 9193
False Positives: 143





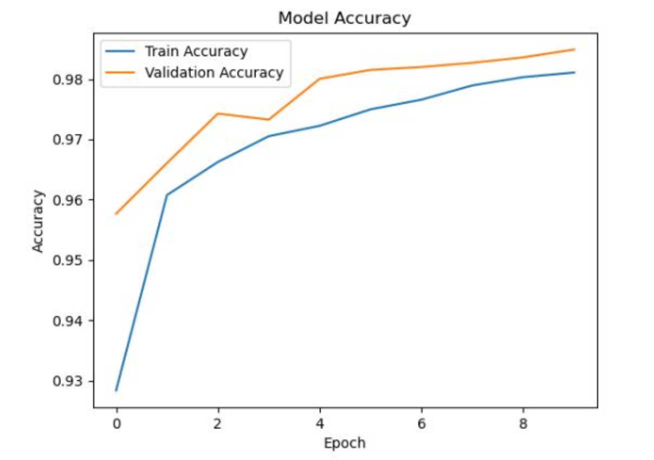
RNN Explained

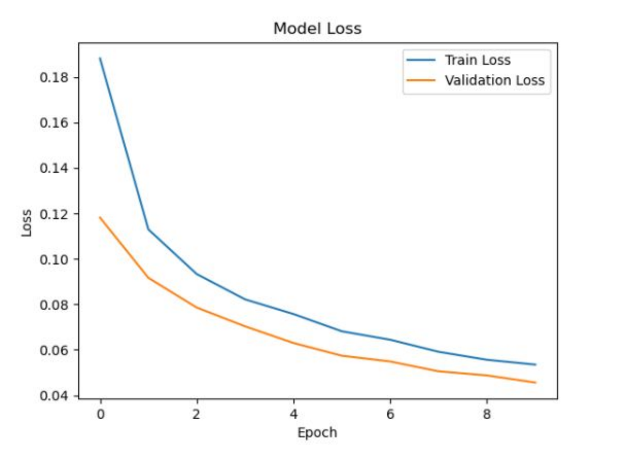
- RNN stands for Recurrent Neural Network.
- RNNs perform well with sequential data making it an ideal neural network for our IDS for botnet attacks.
- RNNs are designed to remember previous input and make predictions based on current and previous input by saving the previous data in something called 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 back propagated through many time steps. As a result, the weights stop updating effectively, and the network fails to learn dependencies from earlier in the sequence.



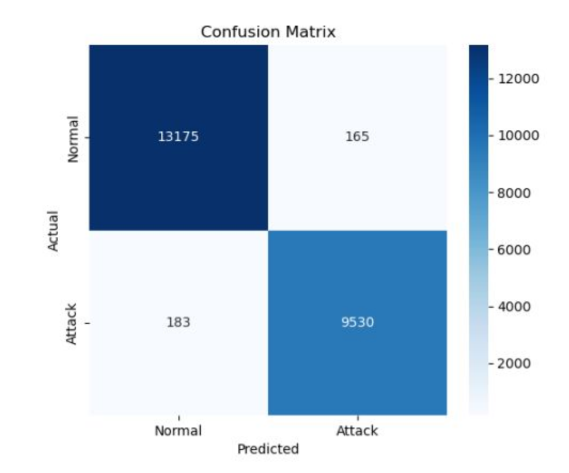
X - inputs
H - hidden states
U, W - Weights
L - Loss
V - Vector being used

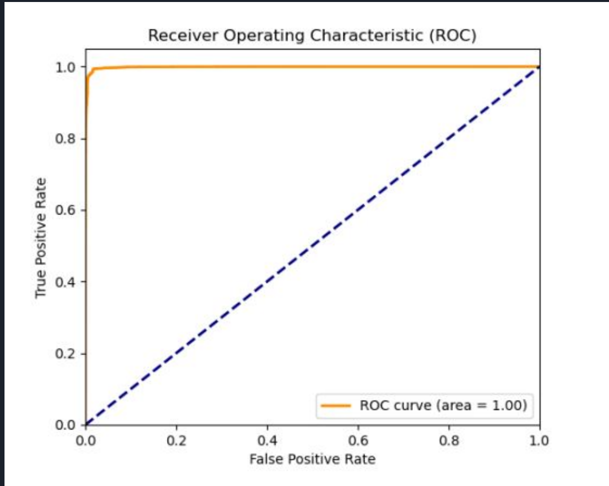
RNN Results



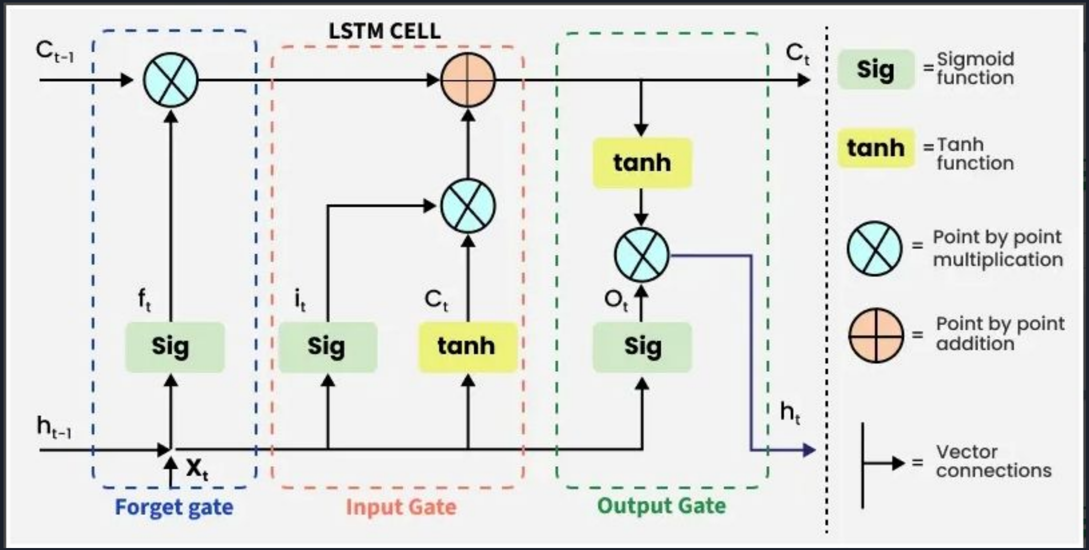


Accuracy: 0.9849
Precision: 0.9830
Recall: 0.9812
F1-Score: 0.9821
False negatives: 183
False Positives: 165





LSTM Explained

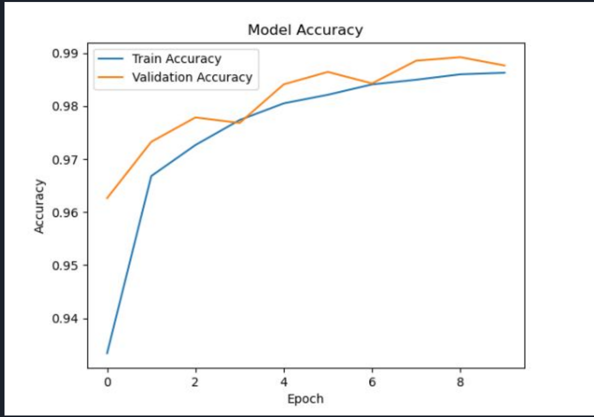


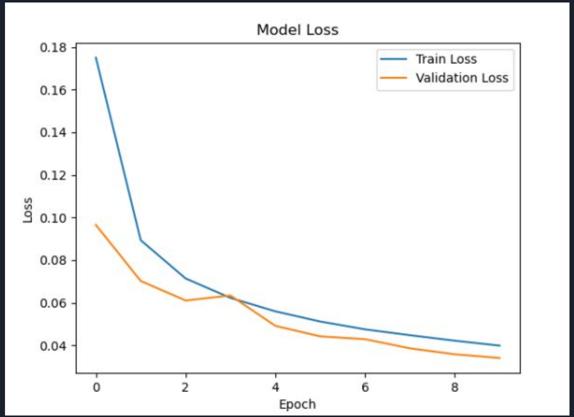
X - inputs
h - Hidden States
C - Cell State

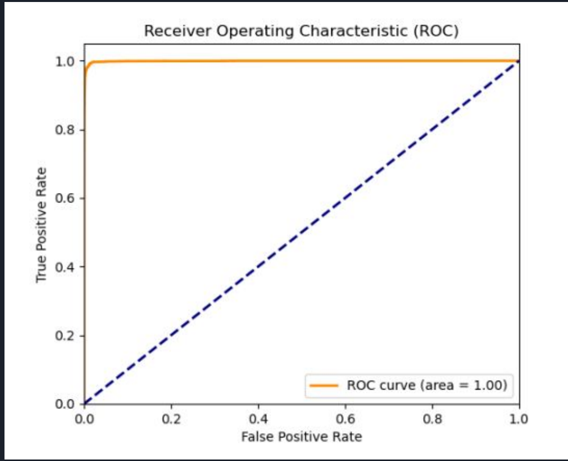
- LSTM stands for long short-term memory and is a type of RNN.
- LSTM is often used for sequence modeling tasks and seeks to overcome some of the issues brought up by RNNs - namely the vanishing gradient problem.
- Three gates are used within an LSTM memory cell. The input gate, forget gate, and output gate control what information is added, removed, and output by the cell respectively.
- 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.

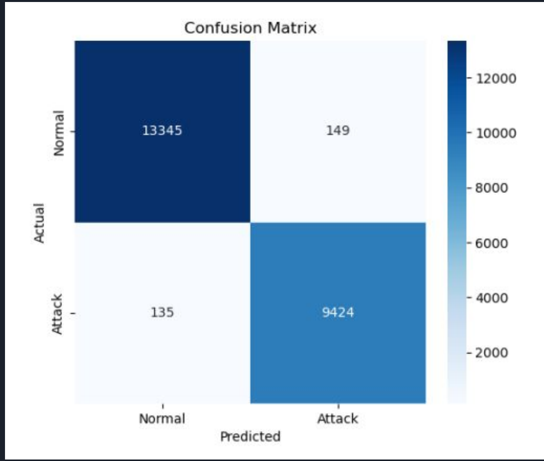
LSTM Results

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









Results Summarized

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

Algorithms aimed to classify traffic as **normal** or **malicious**
Performance varied across models due to differing classification approaches
Random Forest emerged as the most effective algorithm for:

- Classifying botnet data in this specific IDS
- Delivering strong performance with the **CTU-13 dataset**
- Aligning with its common use in other IDS implementations

Neural Network Results:

- LSTM and RNN performed well, **better than most ML models**
- However, neither outperformed **Random Forest**
- Neural Networks designed to handle time series data performed the best amongst the neural network implementations

Key Takeaways:

- Simpler machine learning models can be **highly effective**
- Random Forest may be preferred over more **computationally expensive** neural networks in many IDS contexts

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.