The Utilization of Machine Learning Algorithms in Network Traffic Intrusion Detection

Kavitha Gowda

Received September 17, 2024
Accepted January 13, 2025
Electronic access January 31, 2025

The increasing prevalence of cyber-attacks in our current digital era has underscored the importance of cybersecurity research. Within this context, the integration of artificial intelligence (AI) has emerged as a focal point for research, particularly in the realm of enhancing threat detection mechanisms. This study seeks to explore the effectiveness of different machine learning models for classifying malicious and normal traffic data. Before model construction, the dataset underwent preprocessing where non-numerical values were converted into numerical form and any rows with missing values were dropped. Four models were evaluated: a logistic regression, neural network, decision tree, and random forest. Among these, the neural network noticeably outperformed the others across all three measured metrics, exhibiting a precision of 84.6%, a recall of 95.7%, an F1 score of 0.898, and an overall accuracy of 90.6%. A model yielding such outcomes could be implemented into modern frameworks, empowering users – ranging from large corporations to small businesses and individual users – to preemptively identify impending threats, thereby allowing for the timely implementation of necessary protective measures.

Keywords: machine learning, cybersecurity, network traffic, classification

Introduction

As Artificial Intelligence (AI) becomes increasingly prevalent in modern infrastructure, understanding its implications on cybersecurity and exploring its capacity to create effective protection mechanisms has become a crucial area of research.

In today's digital age, cybersecurity has become an essential area of focus. It aims to protect the systems, applications, sensitive data, and financial assets of individuals and organizations from sophisticated ransomware attacks, viruses, and other cyberattacks[1]. As more information is being digitized and stored online, these cyberattacks are growing increasingly pervasive. In fact, over the past year, there has been a 75% increase in intrusions into Cloud environments*. Without adequate precautions, individuals risk having critical personal information such as credit cards, social security numbers, and other important documents stolen, resulting in financial loss, restricted access to or deletion of personal files, and identity theft[2]. Furthermore, cyberattacks have the potential to disrupt, damage, or even destroy businesses, leading to significant financial consequences for the victims. For instance, IBM's Cost of a Data Breach 2023 report indicates that the average cost of a data breach in 2023 was 4.45 million USD, marking a 15% increase over the past three years[1].

* National University. (n.d.). 101 Cybersecurity Statistics and Trends for 2024. Retrieved September 9, 2024, from https://www.nu.edu/blog/cybersecurity-statistics/

However, cybersecurity presents a few challenges, particularly in balancing detection speed and accuracy. Real-time threat detection often requires rapid, in-the-moment responses to mitigate the damage done by attacks, but achieving this speed can come at the cost of reduced accuracy, leading to either missed threats or excessive false alarms[3]. To safeguard against such cyber threats, it is crucial to be able to identify malicious software effectively. AI holds great promise in this regard, as it can process vast amounts of data in a fraction of the time it would its human counterparts and can also recognize patterns that may be more difficult for the human eye to spot, leading to better accuracy in detecting malicious activity. However, despite increased efficiency, utilizing AI still poses real time constraints with processing large volumes of data at low latency, as the goal is to have high computational efficiency without sacrificing precision[3]. Additionally, with the dynamic nature of cyber threats and hackers constantly adapting to protective measures, models will have to be frequently updated and retrained to maintain efficacy[4]. But these challenges will be overcome as the field of cybersecurity grows and evolves - regardless, utilizing machine learning in intrusion detection will undoubtedly yield better results, preventing a countless amount of cyberattacks. Specifically, machine learning models can be trained on network traffic data to efficiently recognize abnormal patterns indicative of security threats, such as malware infections, intrusion attempts, and unauthorized access[5].

This paper aims to explore the efficacy of various machine

learning models and their combinations that yield the highest accuracy in differentiating between malicious and normal network traffic. These models include a neural network, logistic regression, decision tree, random forest classifier, and a final combined algorithm that incorporates the outcomes of all tested models. While this study does not directly tackle zero-day attacks or real-time processing constraints, it contributes by testing models on a controlled synthetic dataset, which ensures that anomalies and malicious patterns are adequately represented. This addresses challenges posed by limited or unbalanced real-world data – an approach fairly unique from previously conducted studies.

The classification of network traffic into normal and malicious categories has been a central focus in cybersecurity research, with various machine learning models applied to enhance detection accuracy. The models we implemented in our study are among the most commonly utilized in this domain. For instance, a study by Hammad et al. (2021) evaluated the performance of random forests in network intrusion detection using the UNSW-NB15 dataset, highlighting the model's ability to handle complex datasets. The research demonstrated that random forests could achieve high accuracy in detecting network intrusions with an accuracy of 90.14%, aligning with the 81.1% accuracy observed in our study's random forest[6].

In another study by De Carvalho Bertoli et al. (2021), the decision tree, random forest, and logistic regression models were evaluated on a dataset consisting of the MAWILAB dataset combined with an Attack Dataset the researchers had created for a Local Area Network (LAN) environment. Their research found that the Random Forest and Decision Tree models outperformed the Logistic Regression which mostly supports our findings of more complex models (Random Forest) outperforming the simpler models (Logistic Regression) with the exception of the results of the Decision Tree[7].

Similarly in a study by Disha and Waheed (2022) using the UNSW-NB15 dataset and a Gini Impurity-based Weighted Random Forest (GIWRF) feature selection technique, they found that the Decision Tree outperformed all other models with an accuracy score of 93.01%. The Neural Network also proved effective with an accuracy score of 87.26%. Though, their findings on the Decision Tree did not align with our results, the accuracy/efficiency of the neural network is mirrored in our study where the Neural Network obtained an accuracy of 90.6%[8].

Method

The Dataset

The dataset utilized for this project is from Kaggle and named "Network Traffic Data for Intrusion Detection."† The set is composed of 2000 instances of synthetic network traffic designed to emulate real-world network scenarios. Its primary purpose is to facilitate the development, training, and evaluation of intrusion detection systems (IDS) and other cybersecurity models[9]. The dataset encompasses a range of network traffic properties, including the connection duration (in seconds), protocol type (TCP, UDP, ICMP), source and destination IP addresses, source and destination port numbers, number of packets in the connection, number of bytes transferred, and a label indicating whether the traffic was normal or an attack. Additionally, it is important to note that 51% of the data is comprised of attacks and the other 49% of normal traffic flows. This dataset was chosen for our research since synthetic data provides a more controlled environment for experimenting and may not be as complex as real-world data, allowing for simpler and more efficient models. The goal was that our results would provide the baseline models/algorithms that work well in network traffic classification, which could then by further improved when dealing with larger real-world datasets.

Exploratory Data Analysis

Each feature or variable in the dataset has a varying degree of significance in the framework of a machine learning model when determining the target variable: whether the network traffic flow is an attack or normal. Performing an exploratory data analysis (EDA) allows us to detect outliers, reveal relationships and underlying structures among variables, and help us better understand patterns within the data[10]. An example of an EDA graphic is a correlation matrix, otherwise known as a heat map, as demonstrated in Fig.1 below.

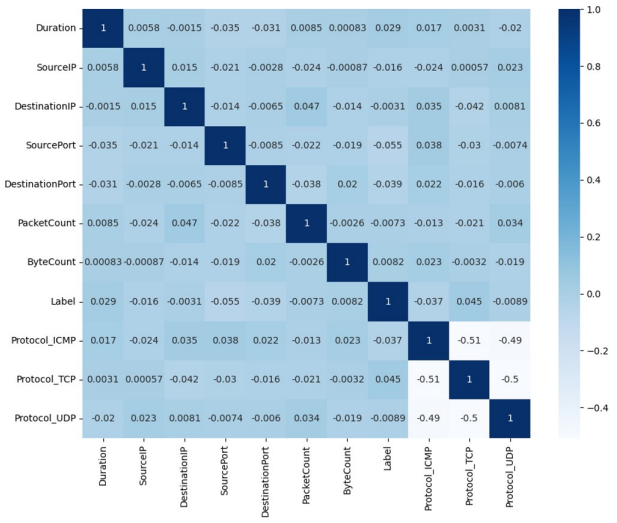

Fig. 1 Correlation Matrix for Network Traffic Data

In a correlation matrix, values closer to +1 indicate a strong positive correlation between the two variables, values closer to -1 indicate a strong negative correlation, and those closer to 0

† https://www.kaggle.com/datasets/mohdzia356/network-traffic-data-for-intrusion-detection

show little to no linear correlation. The diagonal line across the middle is the correlation of the variables to themselves which is why they all have values of 1[10]. These relationships are also visualized by a color map, with darker colors signifying a stronger correlation and lighter colors suggesting a weaker correlation between the two variables, as demonstrated by the color bar to the right of the matrix. As can be observed in the row marked 'label' of the correlation plot in Fig. 1, no variable had a high correlation (>0.5) with determining network traffic status. In fact, the greatest value of correlation was 0.045 with Protocol TCP as the feature, indicating that there was not a strong linear association between any one feature and a traffic flow's label. Given these results, with no variable having a strong correlation with the target label, it would be wise to apply feature engineering and/or dimensionality reduction to the data before building the models – one of which we implemented much later on that proved to yield better results.

Preprocessing

The original dataset was composed of 9 columns with three different variable types: integer, string, and decimal. In order for the models to function, it is imperative that each input and output variable is numerical. So, to meet this criterion, it was necessary to modify our four String variables to be only in integer or decimal form.

Initially, we addressed the two IP address inputs – Source IP and Destination IP – by converting them into integer values. This was simple as IP addresses are numerical with periods separating parts of the address. Thus, removing the periods in each address, and then converting the data type from string to integer form, accomplished the necessary conversion.

Subsequently, we focused on the 'Protocol' input which describes the protocol used for a traffic flow. This was a categorical variable – a variable that belongs to a group based on some qualitative property – as the three protocols within the variable were TCP, UDP, and ICMP. To convert this input into integer form we used a technique called 'One Hot Encoding.' This method separates each category into its own column and gives each traffic flow a value of either a 1 or 0, indicating the Boolean values of true or false. For example, the first traffic flow in the dataset follows the TCP protocol, so after One Hot Encoding, the TCP column has a value of 1, representing that it is true, while the other two columns have a value of 0, displaying that it is false (the traffic did not follow those protocols). After One-Hot-Encoding, we dropped the original column from the dataset and added the three new ones, successfully fitting the 'Protocol' input to our models.

Lastly, the final string variable, 'Label,' was our output, signaling whether the traffic was classified as an 'attack' or 'normal.' As this was a binary categorization, assigning the categories with values of 1 or 0 was a straightforward process. With the primary objective of identifying how accurately our models can detect attacks in mind, we assigned the 'attack' category a value of 1 and the 'normal' category a value of 0.

Upon completing the process of converting every string variable into numerical form, the revised data frame appears as shown in Table 1.

Data Split

Our data was split into a training and testing set using an 80/20 split. This means that we fitted our model on 80% of the data to train it, while the other 20% was used to test the accuracy of the model. A Train/Test Split, such as the one we utilized, is essential because it helps assess how well a machine learning model will perform on new, unseen data. Therefore, in our case, it can help predict how well a model can classify new network traffic data into malicious and normal.

Models

The first model we built was a simple logistic regression model, or logit model, with no added parameters. Logistic regression is used for binary classification tasks; specifically, it simulates the likelihood of an instance falling into a specific class or category, producing results between 0 and 1[11].

The second model we tested was a neural network, A neural network works like a human brain: it consists of many 'nodes' (artificial neurons) that are each responsible for a simple computation. Then these nodes are connected with varying strengths to form a network that produces an output[12]. The model learns during initial training, where the neural network modifies the strengths of each input/connection to lower overall loss in accuracy (a process called 'Backpropagation'). This includes giving inputs that have a greater contribution to producing the right answer a higher weight, and vice versa with variables of less importance to the output. The amount of adjustment to each weight is determined by calculus, where the model takes the derivative (gradient) to see which changes minimize the loss function[13]. Furthermore, neural networks are usually structured with multiple layers or tiers. The first layer – the input layer – receives raw input information, and then the data is passed through the succeeding layers (known as "hidden layers") where it is transformed in complex ways before finally arriving at the output layer, producing the final result of all the data processing done by the neural network[14]. For this model, we initially started with no added parameters, but after testing, we set the 'hidden layer sizes" to (100,100) and the maximum number of iterations to 350.

The third model we created was a decision tree with a max depth of 5. A decision tree is used to categorize or make predictions based on how a previous set of questions was answered. It has a hierarchical, tree-like structure, consisting of a root

	Duration	SourceIP	DestinationIP	SourcePort	DestinationPort	PacketCount	ByteCount	Label	Protocol_ICMP	Protocol_TCP	Protocol_UDP
0	24.077749	1921681239	1921681234	8055	1	827	198244	1	0	1	0
1	97.252384	1921681176	192168182	63174	687	673	1202973	0	1	0	0
2	85.842654	1921681120	1921681113	30873	570	319	984671	1	0	1	0
3	79.196991	1921681212	1921681140	4410	683	375	36284	0	0	0	1
4	34.928018	192168117	1921681223	4702	989	465	817463	0	0	1	0

Table 1 First 5 Data Points of Revised Data Frame

node, branches, internal nodes, and leaf nodes. The base of the tree is the root node. From there flows a series of decision nodes/internal nodes that each represent a question or a decision to be made. Each node conducts evaluations to form binary subsets based on the available features fed to the model. Then at last come the leaf nodes which represent all the possible consequences or outcomes of these evaluations [15]. Simply put, decision trees are algorithms that only contain conditional statements (if-statements) resulting in a tree-like structure. A visual example of a decision tree is demonstrated in Fig. 2, which shows our decision tree model trained on network traffic data with a max depth of 4.

Our last model was a random forest classifier with a max depth of 10. A random forest classifier works by building many decision trees during the training phase with each tree constructed using a random subset of the data set to introduce variability and improve overall prediction performance. When predicting, the algorithm collects the results of all the trees and provides a final classification based on the voting majority, a process often referred to as 'bagging'[‡]. For this model, after testing, we added the 'n_estimators' parameter which set the number of trees in the forest to 100.

After creating 4 standard models, we built one final algorithm that combined all 4 models to produce results based on a majority outcome. To do this we first created a list with the cumulative predicted values from every model for each traffic flow. Recalling that an output from a model could be a 1 or 0 representing malicious or normal traffic, the potential value for each traffic flow ranged between 0 and 4. Next, we iterated through the combined list to determine whether the sum of each instance was equal to at least 3, which is considered the majority in a group of 4. If the traffic flow sum had a value of 3 or greater, then it could be classified as an attack, otherwise, the traffic could be classified as normal. As a result, this algorithm produced a new list based on the majority vote from all the models combined, which theoretically should produce the best accuracy.

‡ https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

Processing

Originally, every model exhibited an accuracy of approximately 50%. We investigated to determine if the reason for this low accuracy was overfit models. Overfitting occurs when a model is overly trained on the training data, leading to poor performance on new, unseen data. Our examination revealed that the decision tree model, as well as our second random forest model, had training accuracies of 60% and 98%, respectively, with the latter being significantly higher than its testing accuracy. To address this overfitting issue, we adjusted the models' parameters, specifically the max depth of each model. Max depth, as the name suggests, refers to the maximum depth of a tree, with greater depth signifying more complexity. By reducing these depths from 5 and 10 to 3 for the decision tree and random forest, respectively, we ensured that the training and testing accuracies aligned more and that the models were not overfitting as much. These adjustments helped the overfitting, yet the models' performances on the testing data remained the same, yielding accuracies of around 50%.

Our next approach was to evaluate for any bias the models may have had towards either class. To achieve this, we delved into more specific metrics by calculating the testing accuracies for all instances of 0s and all instances of 1s. This approach enabled us to assess the performance of each model with respect to each class. Upon examination, we discovered that the accuracy count for any one class significantly surpassed the other constantly, even reaching as high as a 100% to 0% accuracy ratio, indicating that there was indeed heavy bias. The root cause of this bias may have stemmed from the starting random initialization of input weights by the models, followed by subsequent weight adjustments based on the correctness of the predictions. Consequently, if our testing data predominantly included 1s initially and then transitioned to 0s, all the training conducted in the initial half would undergo continual readjustment in the later phase, leading to high accuracy in identifying 0s. This same pattern would occur if the scenario were reversed, resulting in overfitting to the 1s. Considering these findings, two potential courses of action emerged: first, normalizing the data, and second, balancing the two classes within our dataset.

Thus, our final approach included feature scaling, dimension-

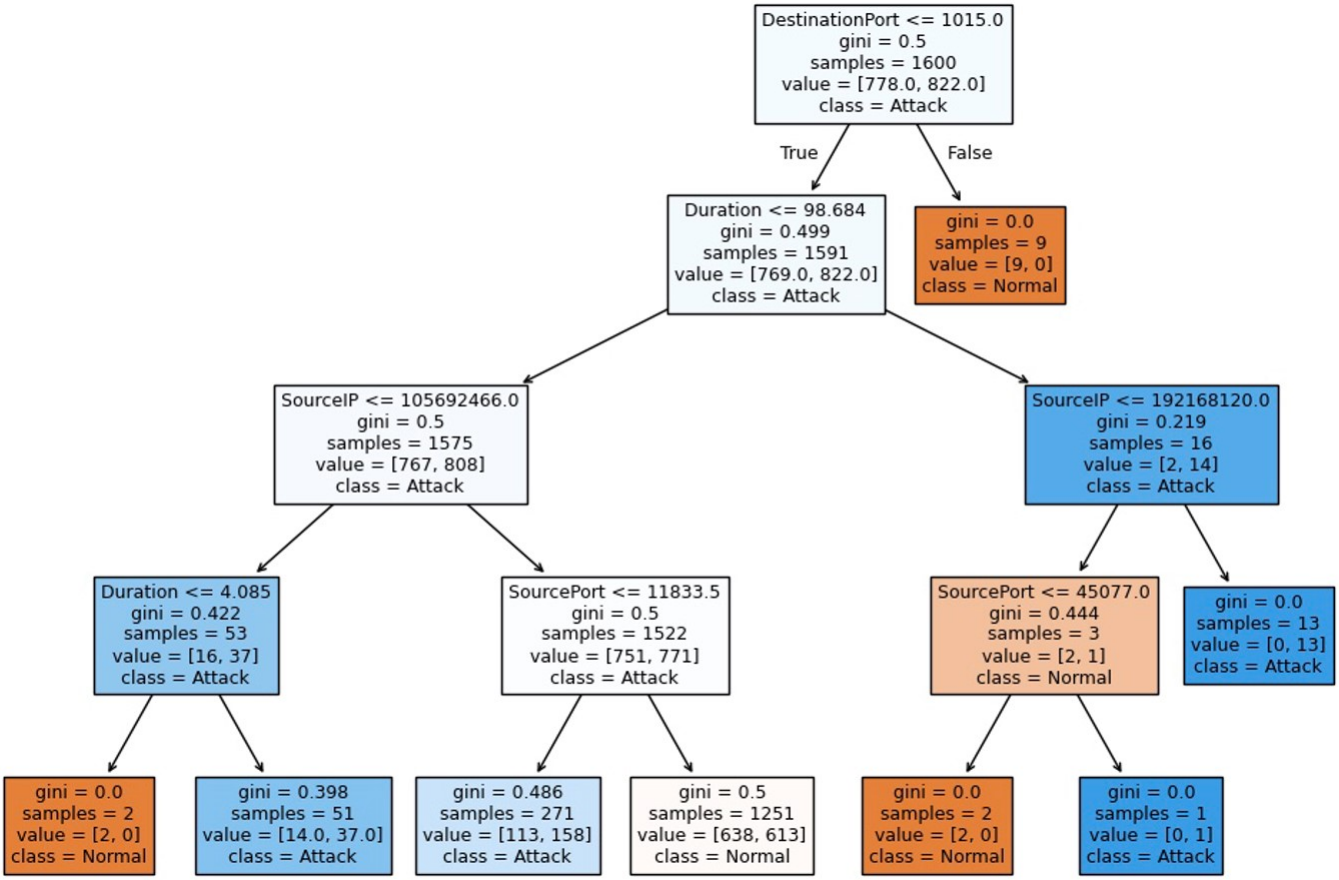

Fig. 2 Visualized Decision Tree for Network Traffic Data

ality reduction, and class balancing. Feature scaling involved using the StandardScaler method from Scikit-learn to transform the range of each feature's values in the dataset to fit the same scale. This data normalization process allows every variable to be considered equally and reduces any bias towards certain features, ultimately prompting better model performance, especially in logistic regression and neural network machine learning models[16].

Next, dimensionality reduction describes a kind of feature extraction that seeks to minimize the number of input features while preserving as much of the original information as possible. To achieve this, we used Principal Component Analysis (PCA), an unsupervised learning technique that reduces the dimensionality of a dataset by examining interrelations among variables and preserving the most important relationships while discarding others. Using PCA often improves model performance as it simplifies the data and removes noise (unimportant inputs) leading to the retention of most of the original variability/information[17].

Lastly, we addressed the class imbalance by using SMOTE-ENN, a hybrid sampling algorithm. SMOTE (Synthetic Minority Oversampling Technique) is an over-sampling technique that randomly increases minority class examples by duplicating them, while ENN (Edited Nearest Neighbors) is an under-sampling technique that reduces the number of samples in the majority class by removing samples that differ from their neighbors in the majority class. By combining the two in a hybrid sampling algorithm, the number of minority samples (the 'normal' class) increases, and the number of majority samples (the 'attack' class) decreases so that the sample imbalance is mitigated[18]. Though this method is usually only implemented with largely imbalanced datasets, and our dataset had an almost even number of samples from each class, (51% attacks, 49% normal), it was still very effective in improving our models' accuracies and reducing the overfitting to one class. Additionally, we readjusted the parameters of certain models, such as the decision tree and random forest models, by resetting their max depths from 3 back to their original values of 5 and 10, as they performed better this way.

Model Type	Precision	Recall	F1-Score	Accuracy
Logistic Regression	0.533	0.696	0.604	0.604
Neural Network	0.846	0.957	0.898	0.906
Decision Tree	0.607	0.739	0.667	0.679
Random Forest	0.724	0.913	0.808	0.811
All Models Combined				0.849

Table 2 Final Testing Results

Results

Table 2 displays a classification report outlining the precision, recall, and F1-score for the classification of 1s (attacks), as well as the individual accuracies of each model and the combined model accuracy. Precision denotes the percentage of correct positive predictions relative to total positive predictions. This is calculated by dividing the number of true positives (the cases were positive and predicted positive) by the sum of false positives (the cases were negative but predicted positive) and true positives. The recall represents the percentage of correct positive predictions relative to total actual positives and is calculated by dividing the true positives by the sum of the false negatives (the cases were positive but predicted negative) and the true positives. And F1-score is a combination of both precision and recall, making it similar to accuracy with a value closer to 1.0 signifying a better model.

As shown in Table 2, the most accurate model was the neural network displaying a precision of 84.6%, a recall of 95.7%, and an overall accuracy of 90.6%. Subsequently, the random forest model demonstrated the second-highest performance with a precision of 72.4%, a recall of 91.3%, and an overall accuracy of 81.1%. The decision tree followed with a precision, recall, and accuracy of 60.7%, 73.9%, and 67.9% respectively. And lastly, our worst-performing model was the logit model with a precision, recall, and accuracy of 53.3%, 69.6%, and 60.4% respectively. Synthesized together in the combined model, however, these models produced a collective accuracy of 84.9% making it the next best predictor after the neural network.

Fig. 3 denotes the ROC curves for all of our models. An ROC (Receiver Operating Characteristic) curve illustrates the trade off between the True Positive Rate (TPR) and False Positive Rate (FPR) for each model as the decision threshold varies. Since we did not modify the threshold, the default threshold of 0.5 was used for classification in all models, meaning predictions with probabilities greater than or equal to 0.5 were classified as attacks. The AUC (Area Under the Curve) quantifies the overall performance of each model, with a value closer to 1.0 indicating better classification ability. The dashed red line represents random guessing (AUC = 0.5), providing a baseline for models to be compared with. Also, models with curves closer to the top-let corner of the plot are better at balancing sensitivity and specificity. Sensitivity is the true positive rate or recall (y-axis), which measures the proportion of actual positive cases that the model correctly identifies as positive while specificity is the true negative rate, which measures the proportion of actual negative cases that the model correctly identifies as negative. The false positive rate (x-axis), which is the rate of negative instances that were incorrectly predicted to be positive, can be found by subtracting the specificity from 1, as the two are complements.[§]

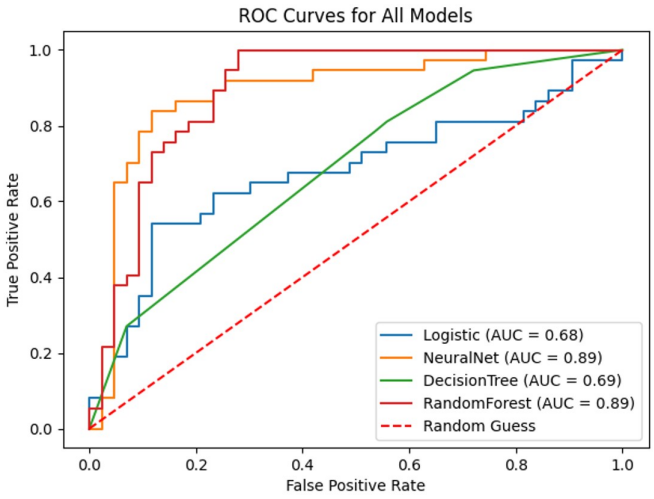

Fig. 3 AUC-ROC Curves for All Models

In this plot, the Logistic Regression, with an AUC of 0.68, has a relatively high FPR compared to the others indicating that it generates more false alarms, while its TPR is not very high, meaning that it doesn't classify attacks as effectively as the other models. Similarly, the Decision Tree, with an AUC of 0.69, also has a higher FPR and moderate TPR, displaying how these two models are not ideal for our task. A high FPR poses large financial implications as it can overwhelm security teams with false alarms and waste financial resources. And with a moderate TPR, the model could miss critical attacks, reducing its reliability and putting systems at risk. On the contrary, the Neural Network and Random Forest, both with an AUC of 0.89, maintain a low FPR and high TPR. Thus, these two models are our best choices as they minimize false alarms (reducing wasted resources) while still catching most attacks.

Discussion

Upon reviewing all our models, an interesting observation arises regarding the performance metrics. It is noteworthy that the recall for each model was consistently and significantly higher than its precision, with this trend being most evident in the random forest model. In the context of our data, this indicates

§ https://www.geeksforgeeks.org/auc-roc-curve/

Study	Dataset	Model Type	Precision	Recall	F1-Score	Accuracy
Disha and Waheed (2022)[7]	UNSW-NB15	Neural Network	0.8202	0.9844	0.8948	0.8726
		Decision Tree	0.9269	0.9476	0.9372	0.9301
De Carvalho Bertoli et al. (2021)[8]	MAWILab + LAN Attack Dataset	Decision Tree	X	X	0.96	X
		Random Forest	X	X	0.96	X
		Logistic Regression	X	X	0.70	X

Table 3 Final Testing Results of Previous Studies with Similar Models

that out of all the attack instances in the network traffic, the random forest predicted this outcome correctly 91.3% of the time – a whole 18.9% greater than the model's precision, which shows that out of all the traffic instances that the model predicted were attacks, only 72.4% actually were. Simply put, every model tended to predict 1 ('attack') more frequently than it did 0 ('Normal') leading to better accuracy in predicting attacks over normal traffic flows. However, given the nature of our data, it is safer for the models to misclassify more normal traffic as attacks rather than malicious traffic flows as normal (a consequence of higher precision than recall). This is due to the fact that a higher number of false negatives has more serious implications in our scenario and raises greater security concerns while having more false positives allows us to be more cautious and safer. Therefore, in our specific context, a higher recall holds greater significance and is preferable.

Limitations

In general, a limitation produced by our dataset (which contains synthetic network traffic) is that it may not capture all the complexities of actual network environments despite being generated using random data generation techniques. Thus, our models may perform very differently on real network traffic data and will likely need to be modified when performing on larger, real-world datasets. One such dataset, with over 2 million records, that could be used when conducting more advanced future research is the UNSW-NB15 dataset, which is the most commonly used in this field. Additionally, the models will need to be further adapted and complicated when applying to real-world situations, where the ability to have instantaneous detection and detect zero-day attacks is crucial – an important next step for research in this field.

Future Improvements

To potentially improve our outcomes, one should consider using skicit-learn's 'PolynomialFeatures' tool prior to implementing PCA. 'PolynomialFeatures' increases dimensionality by producing each feature's interactions with each other. It achieves this by raising each existing feature to a power, which can help uncover more complex, non-linear relationships between the features and target variable. By utilizing this method before PCA, which reduces the number of inputs to the most important ones, our results may be improved. Furthermore, employing a type of hyperparameter tuning, such as GridSearchCV, will help find the best parameters for each model. Instead of setting parameters based on numerous trials, as we did, hyperparameter tuning would unquestionably optimize our models' performances and improve our results. Additionally, deeper analysis could be conducted on our inputs prior to model development, especially on the three different internet protocols (TCP, UDP, ICMP) to explore whether any of them were more indicative of malicious behavior. This analysis could then influence our model/algorithm design. Also, conducting analysis and experimenting with different thresholds for the models instead of keeping the default of 0.5 would likely improve results as well. Lastly, incorporating a slight bias towards identifying attacks in the training set would ensure that there is a higher recall every time the model is run, which is necessary due to the nature of our situation.

Conclusion

The goal of this project was to build multiple models on network traffic data and find the ones that performed the best across all three measured metrics: precision, recall, and accuracy. We built four standard models, in addition to a combined model which took the output of each model and merged them to produce a classification based on majority vote. To prepare the data for processing, we converted all non-numerical variables into numerical form and applied preprocessing methods such as StandardScaler, PCA, and SMOTE-ENN. Following this, we split the data, allocating 80% for training and 20% for testing the models. Each model was then trained on the training data, and hand-tuned through hundreds of trials. The final step involved testing each model on the testing data and analyzing the results. The neural network proved to be the most accurate model in classifying data with a precision of 84.6%, a recall of 95.7%, and an overall accuracy of 90.6%. It was followed by the combined model, which produced an accuracy of 84.9%, and then the random forest with a precision, recall, and accuracy of 72.4%, 91.3%, and 81.1% respectively. Despite being trained

on synthetic network traffic data, these models can be adapted and applied to real network traffic. Further fine-tuning, such as utilizing GridSearchCV to identify the optimal parameters for each model and implementing Polynomial Features, can potentially enhance these models to reliably detect attacks. Ultimately, they have the potential to enhance digital infrastructure security by alerting users of incoming attacks and can even be further improved to autonomously neutralize such threats. These types of AI detection tools are currently gaining more traction, yet there remains significant potential for their advancement within the cybersecurity realm. Therefore, future AI research should prioritize enhancing existing tools and exploring new, innovative methods of addressing the growing threat of cyber-attacks.

References

1 G. Lindemulder and M. Kosinski, *What is Cybersecurity?*, https://www.ibm.com/topics/cybersecurity, 2024, Retrieved September 9, 2024.

2 Individual, Community and P. D. (ICPD), *Cyberattack — Impact*, https://community.fema.gov/ProtectiveActions/s/article/Cyberattack-Impact, Retrieved September 9, 2024.

3 I. Sarker, A. Kayes, S. Badsha *et al.*, *Cybersecurity data science: an overview from machine learning perspective*, 2020.

4 R. Sommer and V. Paxson, *Outside the Closed World: On Using Machine Learning for Network Intrusion Detection*, 2010.

5 I. A. Alwhbi, C. C. Zou and R. N. Alharbi, *Encrypted Network Traffic Analysis and Classification Utilizing Machine Learning*, 2024.

6 M. Hammad, N. Hewahi and W. Elmedany, *T-SNERF: A novel high accuracy machine learning approach for intrusion detection systems*, 2021.

7 G. De Carvalho Bertoli *et al.*, *An End-to-End Framework for Machine Learning-Based Network Intrusion Detection System*, 2021.

8 R. Disha and S. Waheed, *Performance analysis of machine learning models for intrusion detection system using Gini Impurity-based Weighted Random Forest (GIWRF) feature selection technique*, 2022.

9 M. zia, *Network Traffic Data for Intrusion Detection*, https://doi.org/10.34740/KAGGLE/DSV/8661173, 2024.

10 S. K. Mukhiya and U. Ahmed, *Hands-on exploratory data analysis with Python: perform EDA techniques to understand, summarize, and investigate your data*, 2020.

11 T. Rymarczyk, E. Kozłowski, G. Kłosowski and K. Niderla, *Logistic Regression for Machine Learning in Process Tomography*, 2019.

12 N. N. C. for Data Services (NCDS), *Neural Networks*, https://www.nnlm.gov/guides/data-glossary/neural-networks, Retrieved September 9, 2024.

13 J. McGonagle, G. Shaikouski, C. Williams, A. Hsu, J. Khim and A. Miller, *Backpropagation*, https://brilliant.org/wiki/backpropagation/, Retrieved September 9, 2024.

14 L. Hardesty, *Explained: Neural Networks*, https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414, 2017, Retrieved September 9, 2024.

15 IBM, *What is a Decision Tree?*, https://www.ibm.com/topics/decision-trees, Retrieved September 9, 2024.

16 L. Quang, B. Baek, W. Yoon, S. Kim and I. Park, *Comparison of Normalization Techniques for Radiomics Features From Magnetic Resonance Imaging in Predicting Histologic Grade of Meningiomas*, 2024.

17 A. Macintosh, R. Ellis and T. Allen, *The Effect of Principal Component Analysis on Machine Learning Accuracy with High Dimensional Spectral Data*, 2006.

18 Z. Xu, D. Shen, T. Nie and Y. Kou, *A hybrid sampling algorithm combining M-SMOTE and ENN based on Random forest for medical imbalanced data*, 2020.