ShiftShield: Classification for Network Intrusion

CS 584 Machine Learning

Prof. Yan Yan

Aanal Patel (A20528001)
College of Computing
Illinois Institute of Technology
E-mail GitHub

Abstract

SwiftShield, a sophisticated Network Intrusion Detection System (NIDS), addresses the escalating threat landscape by leveraging machine learning. It develops a multi-class classifier to accurately classify diverse network intrusions, mitigating risks posed by evolving cyber threats. Through meticulous data preprocessing and modeling, SwiftShield offers robust defenses against network-based attacks, ensuring enhanced network security.

1 Introduction

Traditional network intrusion detection systems (NIDS) primarily rely on signature-based detection methods to identify known patterns of malicious activity or unauthorized access. However, these systems are inherently limited in their ability to detect novel or previously unseen threats, leaving networks vulnerable to emerging cyber threats that evade signature-based detection.

Furthermore, the prevalence of false positives and the inability to adapt to evolving attack vectors further exacerbate the challenges faced by existing NIDS solutions. As cyber threats continue to evolve in complexity and diversity, there is a pressing need for more dynamic and accurate intrusion detection mechanisms capable of effectively safeguarding digital assets and preserving network integrity.

1.1 Previous Work[1][2]

In the field of network intrusion detection, signature-based detection systems (SBDS) have long been the cornerstone of defense mechanisms. These systems excel at swiftly identifying known patterns in network traffic that indicate malicious activity or unauthorized access. However, their effectiveness is hampered by inherent limitations, notably their inability to detect patterns of new threats not already cataloged in their signature databases.

To overcome this challenge, researchers have explored alternative approaches, among which anomaly-based detection systems (ABDS) have emerged as a promising solution. Unlike SBDS, ABDS focus on deviations from established norms or standard behaviors within network traffic, rather than predefined signatures. By leveraging statistical models and machine learning algorithms, ABDS can identify anomalies indicative of potential intrusions or novel threats.

1.2 Methods

SwiftShield employs a Machine Learning approach to address the complexities of network intrusion detection. This approach entails training multiple Machine Learning algorithms on the UNSW-NB15 dataset to discern whether network transactions represent attacks and to classify the nature of such attacks. This distinguishing feature sets SwiftShield apart from conventional NIDS solutions.

Before utilizing the dataset for model training, rigorous preprocessing steps were undertaken to ensure data quality and model effectiveness. The preprocessing pipeline encompassed several key stages:

1. **Data Cleaning**: Removal of erroneous or incomplete data entries to enhance dataset integrity.
2. **Data Preprocessing**: This phase included:
    - **One-Hot Encoding**: Conversion of categorical variables into numerical format to facilitate model training.
    - **Normalization**: Scaling of numerical features to a standard range to mitigate disparities in feature magnitudes.
    - **Log Transformation**: Transformation of skewed or non-normally distributed data to achieve a more symmetrical distribution.
    - **Outlier Detection**: Identification of anomalous data points through statistical methods.
    - **Outlier Replacement**: Substitution of outlier values with median values to mitigate their influence on model performance.
3. **Feature Selection**: Utilization of embedded feature selection methods to identify and retain the most informative features for model training.
4. **Stratified Sampling**: Partitioning of the dataset into training and testing subsets while preserving the distribution of target classes to ensure representative sampling.
5. **Modeling**: Implementation of Machine Learning models on the stratified training dataset to learn and generalize patterns from the data.

By meticulously adhering to these preprocessing steps, SwiftShield prepares the dataset for robust model training, thereby laying the groundwork for effective network intrusion detection.

1.3 Results

Following the stratified sampling process, the preprocessed dataset was subjected to various Machine Learning (ML) algorithms, including K-Nearest Neighbors (KNN), Logistic Regression, Decision Trees, Random Forest, Gradient Boosting, and XGBoost.

Among these algorithms, XGBoost exhibited outstanding performance on the dataset, demonstrating exceptional precision and recall across all classification categories, indicating its efficacy in accurately identifying and classifying network intrusion instances.

The success of the XGBoost algorithm can be attributed to its unique ensemble learning approach, which combines the strengths of multiple decision trees through gradient boosting. XGBoost employs a regularization term in its objective function to control model complexity, preventing overfitting and enhancing generalization performance. Additionally, its parallel and distributed computing capabilities enable efficient training on large datasets, making it well-suited for handling the complexities of network intrusion detection tasks.

2 Problem Description

In today's rapidly evolving digital landscape, the proliferation of sophisticated cyber threats

poses a significant challenge to organizations worldwide. Traditional Network Intrusion Detection Systems (NIDS) are struggling to keep pace with the dynamic and complex nature of these threats, often exhibiting limitations in terms of both accuracy and adaptability.

SwiftShield seeks to address these critical challenges by engineering a NIDS with heightened dynamism and accuracy. The primary objective is to develop a sophisticated Machine Learning model or Multi-class Classifier capable of effectively classifying diverse network intrusions alongside normal traffic. SwiftShield's approach is rooted in the recognition that traditional signature-based detection methods are inadequate in detecting novel or previously unseen threats. By leveraging advanced Machine Learning techniques, SwiftShield aims to overcome the limitations of traditional NIDS solutions and enhance network security posture.

Following are the Implementations in theory and practical:

1. **Exploraing the Data**:
    - Examined for any missing values.
    - Substituted the rows containing '-' with NaN (Not a Number).
    - Examined for duplicate values. In the dataset, it's possible for one device to establish multiple connections with another device. Consequently, these entries in the network data cannot be categorized as duplicate data. Other than this scenario, there were no instances of duplicate transactions in the dataset.
    - The cleaning procedure maintained the original size and structure of the dataset intact. The dataset's shape was verified to remain unaltered subsequent to the replacement of missing values.
2. **Data Visualization**: This phase included exploring the dataset through graphs:

- **Distribution of Network Traffic**:

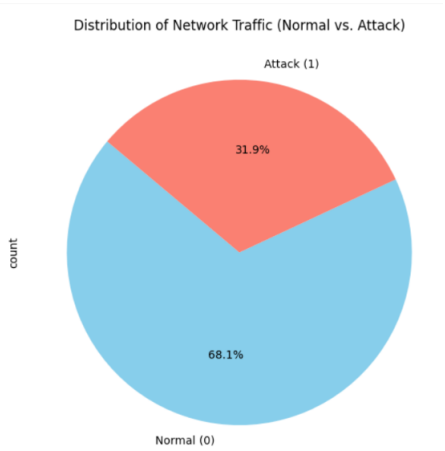


Figure 1: Pie Chart of Distribution of Data

This pie chart illustrates the overall distribution of network traffic between 'Normal' and 'Attack' classifications. The chart is divided into two segments, with 'Normal' traffic comprising 68.1% and 'Attack' traffic comprising 31.9% of the network traffic.

- **Bar Graph distribution of Multiple Class Labels**:
  This bar graph presents the frequency distribution of different categories of network traffic, classified as either normal or various types of attacks. The x-axis lists the categories, including 'Normal', 'Generic', 'Exploits', 'Fuzzers', 'DoS' (Denial of Service), 'Reconnaissance', 'Analysis', 'Backdoor', 'Shellcode', 'Worms'. The y-axis indicates the frequency of each category. The 'Normal' traffic has the highest frequency, followed by 'Generic' and 'Exploits' with significantly fewer instances, and the other categories have progressively lower frequencies.

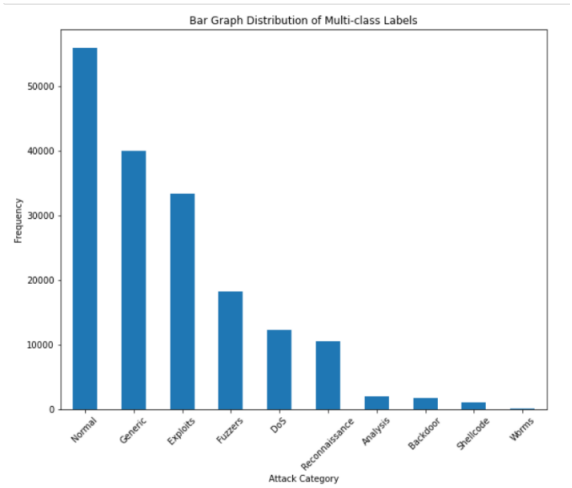


Figure 2: Bar Graph of Distribution of Attacks

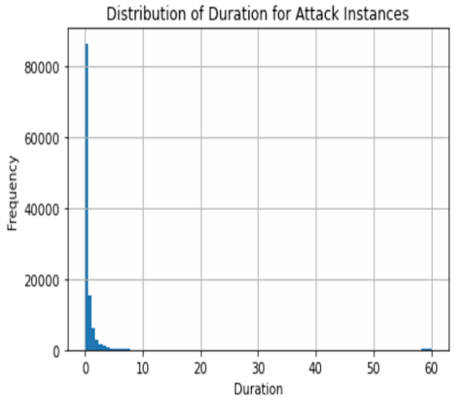


Figure 3: Time Duration of the attacks

- **Distribution of Duration for Attack Instances**: This histogram shows the distribution of the duration of attack instances. The x-axis indicates the duration, and the y-axis shows the frequency. Most of the attack instances seem to be of very short duration, as indicated by the high frequency at the lower end of the duration scale. There is a rapid drop-off in frequency as the duration increases, suggesting that longer attacks are less common.

3. **Data-Preprocessing**:

- **One-Hot Encoding**: It is a technique used to convert categorical variables into numerical format, which is required for training machine learning models. In this process, each category is represented as a binary vector, where each element in the vector corresponds to a category and is set to 1 if the instance belongs to that category and 0 otherwise. This ensures that

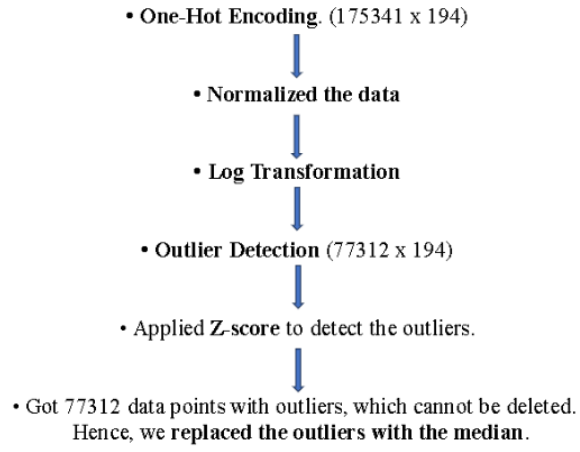


Figure 4: Methodology of Data Preprocessing

the model can effectively interpret and learn from categorical features during training.

Practical Implications on the data: Before One Hot encoding, there were 45 features, and after One Hot encoding, it increased to 194 features. Hence, the dimension of the data became: 175341 x 194.

- **Normalization**: It is a technique used to scale numerical features to a standard range, typically between 0 and 1. Normalization ensures that all features contribute equally to the model's training process and prevents features with larger magnitudes from dominating the learning process.

- **Log Transformation**:

	dur	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	...
0	0.121478	6	4	258	172	74.087490	252	254	1.415894e+04	8495.365234	...
1	0.649902	14	38	734	42014	78.473372	62	252	8.395112e+03	503571.312500	...
2	1.623129	8	16	364	13186	14.170161	62	252	1.572272e+03	60929.230470	...
3	1.681642	12	12	628	770	13.677108	62	252	2.740179e+03	3358.622070	...
4	0.449454	10	6	534	268	33.373826	254	252	8.561499e+03	3987.059814	...
...	...	...	...	...	...	...	...	...	...	...	...
175336	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	0.000000	...
175337	0.505762	10	8	620	354	33.612649	254	252	8.826286e+03	4903.492188	...
175338	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	0.000000	...
175339	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	0.000000	...
175340	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	0.000000	...

Figure 5: Result of Data before Log transformation

After the data was visualized, majority of the columns were right skewed, hence to handle the skewness, log transformation was applied. It is a technique that is used to transform skewed or non-normally distributed data into a more symmetrical distribution or gaussian distribution. By taking the logarithm of the data values, the transformation compresses large values and expands small values, effectively reducing the skewness of the distribution. This helps improve the linearity of relationships between variables and stabilizes variance, making the data more suitable for certain types of machine learning algorithms.

	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	...
0	2.022587e-03	0.618456	0.693147	0.223144	0.000520	0.000364	0.000018	0.000012	0.000074	0.687247	...
1	1.077346e-02	0.618456	0.693147	0.223144	0.001351	0.003457	0.000054	0.002863	0.000078	0.217638	...
2	2.669271e-02	0.618456	0.693147	0.223144	0.000728	0.001457	0.000026	0.000899	0.000014	0.217638	...
3	2.764179e-02	0.618456	0.154151	0.223144	0.001143	0.001093	0.000048	0.000053	0.000014	0.217638	...
4	7.462984e-03	0.618456	0.693147	0.223144	0.000936	0.000547	0.000039	0.000018	0.000033	0.691184	...
...	...	...	...	...	...	...	...	...	...	...	...
175336	1.500000e-07	0.642651	0.080043	0.318454	0.000104	0.000000	0.000007	0.000000	0.105361	0.691184	...
175337	8.394039e-03	0.618456	0.693147	0.223144	0.000936	0.000729	0.000046	0.000024	0.000034	0.691184	...
175338	1.500000e-07	0.642651	0.080043	0.318454	0.000104	0.000000	0.000007	0.000000	0.105361	0.691184	...
175339	1.500000e-07	0.642651	0.080043	0.318454	0.000104	0.000000	0.000007	0.000000	0.105361	0.691184	...
175340	1.500000e-07	0.642651	0.080043	0.318454	0.000104	0.000000	0.000007	0.000000	0.105361	0.691184	...

Figure 6: Result of Data after Log transformation

- **Outlier Detection**: Outliers can skew statistical analyses and affect the performance of machine learning models.

$$Z = \frac{x - \mu}{\sigma}$$

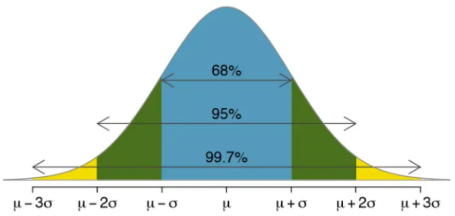


Figure 7: Z-Score[3]

a) 68% of the data points lie between +/- 1 standard deviation.

b) 95% of the data points lie between +/- 2 standard deviation

c) 99.7% of the data points lie between +/- 3 standard deviation

The number of outliers that were obtained 64097, which is almost 1/3rd of the data, hence instead of removing all, it was replaced by the median.

4. **Feature Selection**:

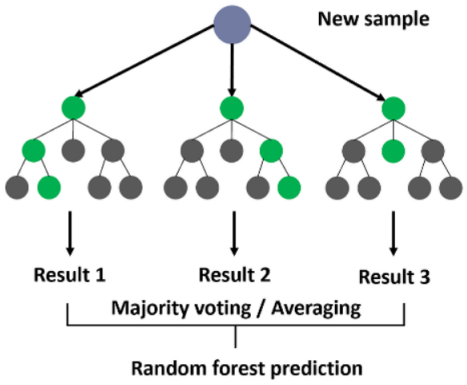


Figure 8: Random Forest Working[4]

The embedded feature selection method employs a Random Forest classifier to assign importance scores to each feature. By setting the threshold as the mean of these scores, features above the threshold are deemed significant and retained, while those below are discarded. This process ensures the selection of the most influential features for the classification task. Finally, the selected features are utilized for modeling, enhancing the model's predictive performance and efficiency by focusing on the most informative attributes.

5. **Stratified Sampling**:

Stratified sampling involves dividing a dataset into homogeneous subgroups based on certain characteristics, then randomly

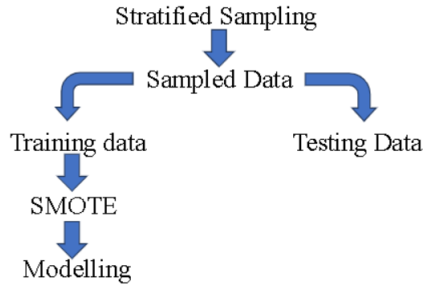


Figure 9: Stratified Sampling

selecting samples from each subgroup. This ensures representation of all groups in the sample. Implemented this sampling technique to address the issue of modeling complexity caused by the large dataset size while maintaining data integrity and opted for 50% of the original data.

Thereafter the dataset was divided into Training and Testing.

6. **SMOTE**:

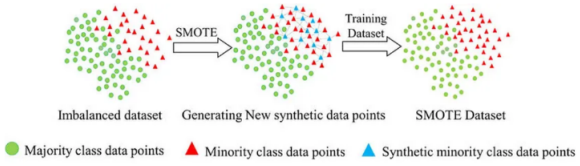


Figure 10: SMOTE[5]

SMOTE (Synthetic Minority Oversampling Technique) is an oversampling method used to address class imbalance in datasets. It generates synthetic samples for minority classes by interpolating between existing minority class samples. This helps balance class distribution and prevents the model from being biased towards majority classes. Implemented to counteract imbalanced datasets, SMOTE ensures equal representation of all classes,

enhancing the model's ability to learn from minority class instances and improve overall performance.

7. **Modeling**[6][7] : *Appliedthefollowingmodels* :

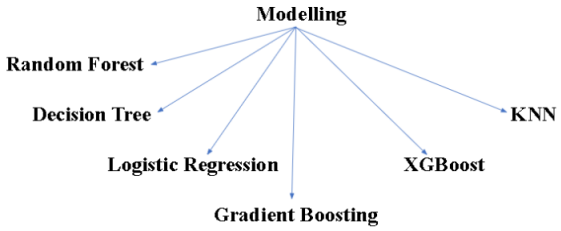


Figure 11: Machine Learning Models

The evaluation matrices that were considered:

1. Confusion Matrix
2. Classification Report

- **Precision:** Precision is the ratio of correctly predicted positive observations to the total predicted positives.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- **Accuracy:** Accuracy is the ratio of the number of correct predictions to the total number of instances.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ instances}$$

- **Recall:** Recall is the ratio of correctly predicted positive observations to the all observations in actual class.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- **F1-Score:** F1-Score is the weighted average of Precision and Recall.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- **Support:** Support is the number of actual occurrences of the class in the specified dataset.

Models and their results:

- **Logistics Regression**: Logistic Regression is a popular algorithm that is used for classification problems in Machine Learning. The algorithm works by modeling the relationship between a set of independent variables and a dependent variable.

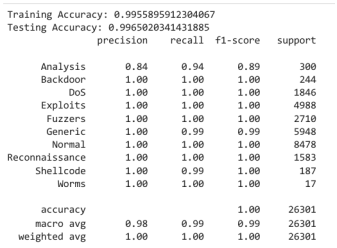
$$p(y = 1|\mathbf{x}; \theta) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)}} = \sigma(\theta^T \mathbf{x})$$

where σ is the logistic function, $\mathbf{x}$ is the feature vector, θ is the parameter vector, and y is the binary target variable.

	precision	recall	f1-score	support
Analysis	0.41	0.98	0.58	300
Backdoor	1.00	1.00	1.00	244
DoS	1.00	1.00	1.00	1846
Exploits	1.00	0.95	0.97	4988
Fuzzers	0.91	0.88	0.89	2710
Generic	1.00	0.98	0.99	5948
Normal	1.00	1.00	1.00	8478
Reconnaissance	1.00	0.96	0.98	1583
Shellcode	0.89	0.98	0.94	187
Worms	0.67	0.94	0.78	17
accuracy			0.97	26301
macro avg	0.89	0.97	0.91	26301
weighted avg	0.98	0.97	0.97	26301

Figure 12: Result of Logistic Regression Model

- **K Nearest Neighbour (KNN)**: kNN is a non-parametric learning algorithm. Contrary to the other ML algorithms that allow discarding the training data after the model is built, kNN keeps all the training examples in memory.
- **Decision Tree**: Decision trees are powerful models that learn decision

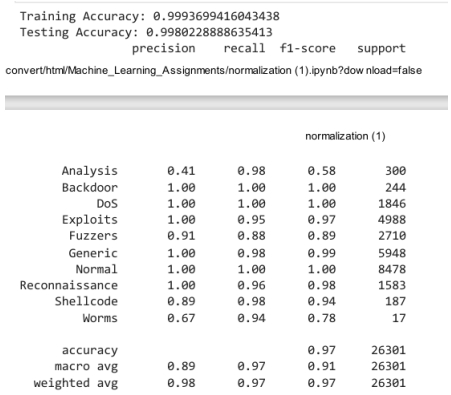


Training Accuracy: 0.9955895912304067
Testing Accuracy: 0.9965020341431885

	precision	recall	f1-score	support
Analysis	0.84	0.94	0.89	300
Backdoor	1.00	1.00	1.00	244
DoS	1.00	1.00	1.00	1846
Exploits	1.00	1.00	1.00	4988
Fuzzers	1.00	1.00	1.00	2710
Generic	1.00	0.99	0.99	5948
Normal	1.00	1.00	1.00	8478
Reconnaissance	1.00	1.00	1.00	1583
Shellcode	1.00	0.99	1.00	187
Worms	1.00	1.00	1.00	17
accuracy			1.00	26301
macro avg	0.98	0.99	0.99	26301
weighted avg	1.00	1.00	1.00	26301

Figure 13: Result of Logistic kNN Model

rules from the data. It recursively divides the data based on different attributes to create a tree structure of decision nodes and leaf nodes.



Training Accuracy: 0.9993699416043438
Testing Accuracy: 0.9980228888635413

	precision	recall	f1-score	support

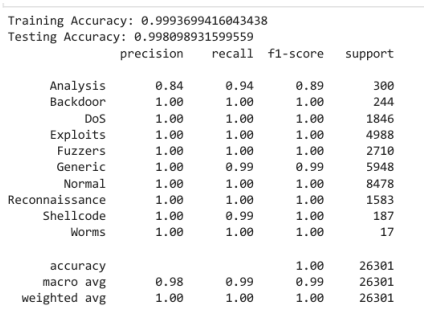
convert/html/Machine_Learning_Assignments/normalization (1).ipynb?dow nload=false

normalization (1)

Analysis	0.41	0.98	0.58	300
Backdoor	1.00	1.00	1.00	244
DoS	1.00	1.00	1.00	1846
Exploits	1.00	0.95	0.97	4988
Fuzzers	0.91	0.88	0.89	2710
Generic	1.00	0.98	0.99	5948
Normal	1.00	1.00	1.00	8478
Reconnaissance	1.00	0.96	0.98	1583
Shellcode	0.89	0.98	0.94	187
Worms	0.67	0.94	0.78	17
accuracy			0.97	26301
macro avg	0.89	0.97	0.91	26301
weighted avg	0.98	0.97	0.97	26301

Figure 14: Result of Decision Tree

- **Random Forest**: Random Forests are powerful ensemble learning algorithms that combine multiple decision trees to create a robust and accurate prediction model. It works well especially when there are complex interactions and non-linear relationships between the entities and the target variable.
- **Gradient Boosting**: It is a powerful ensemble learning technique that combines multiple weak predictive models to create strong predictive

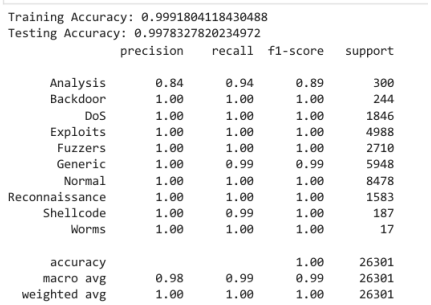


Training Accuracy: 0.9993699416043438
Testing Accuracy: 0.998098931599559

	precision	recall	f1-score	support
Analysis	0.84	0.94	0.89	300
Backdoor	1.00	1.00	1.00	244
DoS	1.00	1.00	1.00	1846
Exploits	1.00	1.00	1.00	4988
Fuzzers	1.00	1.00	1.00	2710
Generic	1.00	0.99	0.99	5948
Normal	1.00	1.00	1.00	8478
Reconnaissance	1.00	1.00	1.00	1583
Shellcode	1.00	0.99	1.00	187
Worms	1.00	1.00	1.00	17
accuracy			1.00	26301
macro avg	0.98	0.99	0.99	26301
weighted avg	1.00	1.00	1.00	26301

Figure 15: Result of Random Forest

model. It is a powerful machine learning algorithm that is known for its high performance and the ability to handle various types of data. It works by iteratively training a sequence of patterns, where each subsequent pattern is built to correct the errors of the previous pattern.



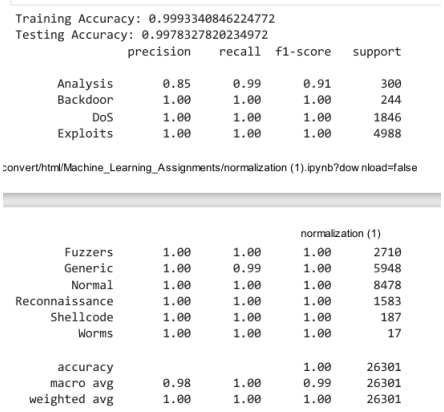
Training Accuracy: 0.9991804118430488
Testing Accuracy: 0.9978327820234972

	precision	recall	f1-score	support
Analysis	0.84	0.94	0.89	300
Backdoor	1.00	1.00	1.00	244
DoS	1.00	1.00	1.00	1846
Exploits	1.00	1.00	1.00	4988
Fuzzers	1.00	1.00	1.00	2710
Generic	1.00	0.99	0.99	5948
Normal	1.00	1.00	1.00	8478
Reconnaissance	1.00	1.00	1.00	1583
Shellcode	1.00	0.99	1.00	187
Worms	1.00	1.00	1.00	17
accuracy			1.00	26301
macro avg	0.98	0.99	0.99	26301
weighted avg	1.00	1.00	1.00	26301

Figure 16: Result of Gradient Boosting

Random Forest and Gradient Boosting algorithms gave similar results.

- **XGBoost**: XGBoost (Extreme Gradient Boosting) operates on the principle of boosting, seeking to iteratively improve the weak learners and construct a robust ensemble model. Its core lies in the concept of gradient boosting, where subsequent weak learners are trained to correct the errors made by the previous learners. XGBoost amplifies this concept with its refined approach, leveraging the power of gradient information to enhance the accuracy and efficiency.



Training Accuracy: 0.9993340846224772
Testing Accuracy: 0.9978327820234972

	precision	recall	f1-score	support
Analysis	0.85	0.99	0.91	300
Backdoor	1.00	1.00	1.00	244
DoS	1.00	1.00	1.00	1846
Exploits	1.00	1.00	1.00	4988

:onvert/html/Machine_Learning_Assignments/normalization (1).ipynb?dow nload=false

normalization (1)

Fuzzers	1.00	1.00	1.00	2710
Generic	1.00	0.99	1.00	5948
Normal	1.00	1.00	1.00	8478
Reconnaissance	1.00	1.00	1.00	1583
Shellcode	1.00	1.00	1.00	187
Worms	1.00	1.00	1.00	17
accuracy			1.00	26301
macro avg	0.98	1.00	0.99	26301
weighted avg	1.00	1.00	1.00	26301

Figure 17: Result of XGBoost

3 Results:

In the following table, the performance of all the models is concluded, and analyzed the results individually:

Model Name	Precision	Recall	F1-Score	Accuracy
Logistic Regression	0.89	0.97	0.91	0.971
KNN	0.98	0.99	0.99	0.996
Decision Tree	0.89	0.97	0.97	0.998
Gradient Boosting	0.89	0.97	0.91	0.997
Random Forest	0.98	0.99	0.99	0.998
XGBoost	0.98	1	0.99	0.999

Figure 18: Classification Report Summary of all models

Model Name	True Positives	False Positives	False Negatives
Logistic Regression	25558	743	706
KNN	26209	87	92
Decision Tree	26249	52	52
Random Forest	26251	50	50
Gradient Boosting	26253	47	47
XGBoost	26253	47	47

Figure 19: Confusion Matrix Summary of all models

8. **Modeling Analyzes:**

- **Logistic Regression**: Starting with the baseline, Logistic Regression showed commendable performance with a high accuracy of 97.1%. It's a great starting point due to its simplicity and interpretability. However, it lagged in precision and recall when compared to the more complex models. With the highest false positives and negatives among our contenders, it's best suited for datasets and problems where the relationship between features is linear and the complexity of the data is lower.
- **K-Nearest Neighbors (KNN)**: KNN improved upon Logistic Regression, with a higher accuracy of 99.6% and better precision and recall. It's a non-parametric method that excels when the dataset is not too large, and the feature space isn't overly complex. KNN showed a marked reduction in false negatives and positives, indicating its capability to handle more nuanced data patterns. Its reliance on the feature space's geometry makes it an excellent choice for when the distance between data points is meaningful.
- **Decision Tree**: The Decision Tree model was a step up, showcasing a robust accuracy of 99.8%. Its hierarchical, tree-structured decision-making process provided clear and interpretable results. With low false positives and negatives, it proved to be reliable, but as a single tree, it might still be prone to overfitting compared to its ensemble counterparts.
- **Random Forest**: Moving to ensemble techniques, Random Forest achieved an impressive accuracy of 99.8%, indicating its strength in dealing with various data types and distributions. Its ensemble nature, leveraging multiple decision trees, significantly reduces the risk of overfitting while improving the model's generalization capabilities. With very low false positives and negatives, Random Forest stands out as a versatile and powerful model for a broad range of classification tasks.
- **Gradient Boosting**: Gradient Boosting is another ensemble heavyweight with an accuracy of 99.7%. It builds trees sequentially, with each one correcting the errors of the previous, which led to excellent performance metrics across the board. Similar to Random Forest, it showed very few misclassifications, indicating strong predictive power, especially in scenarios where predictive performance is paramount.
- **XGBoost**: Finally, XGBoost — the pinnacle of our model performance with a perfect accuracy of 99.9%. It not only delivered flawless precision and recall but also maintained the lowest counts of false positives and negatives. As an optimized gradient boosting algorithm, it provides a scalable and efficient implementation that has proven its worth in various machine learning competitions. XGBoost is the model of choice for this dataset, promising unparalleled performance in both predictive power and operational efficiency.

9. **The Process:**

   After understanding the concept of how the data was preprocessed and observing the results of Machine Learning algorithm, the following is the flowchart of how the attacks in the Network can be identified, where NIDS is nothing but referred to SwiftShield.

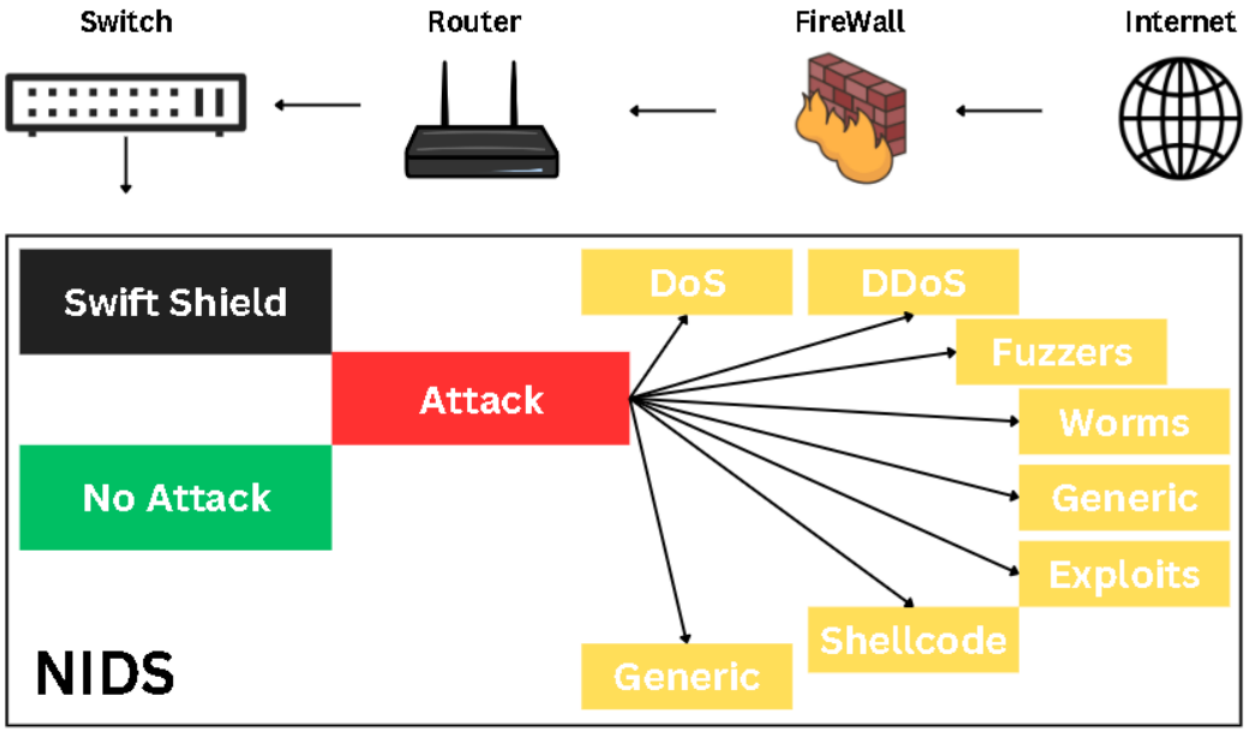


Figure 20: The Process of Network Intrusion Detection

4 Conclusion and Future Work

In summary, while simpler models like Logistic Regression and KNN offer speed and interpretability, they fall short in terms of predictive accuracy compared to their more complex counterparts. But from the boosting algorithm that we implemented, XGBoost exceled in performance, minimizing errors across the board and ensuring robustness and reliability in predictions for complex datasets.

4.1 Future Work:

Examine more intricate or diverse machine learning models, such as deep neural networks, to reduce time complexity and make quick predictions on bigger data sets. On the basis of the model, create a real-time intrusion detection system that can be incorporated into network architecture.

4.2 Scope of the project:

Network Intrusion Detection Systems (NIDS): By integrating the classifier into an NIDS, an organization can improve its security posture against a range of attack vectors by real-time detecting and classifying network intrusions. Anomaly detection involves using the classifier to find unusual patterns in network data that could be signs of fresh attack techniques or zero-day vulnerabilities. Automation of Incident Response: When a specific kind of attack is identified, the classifier can be integrated with automated response systems to take quick action, including limiting traffic or isolating compromised systems.

5 References

[1] Kumar, Satish, et al. 'Research Trends in Network-Based Intrusion Detection Systems: A Review'. IEEE Access, vol. 9, 2021, pp. 157761–79. IEEE Xplore, https://doi.org/10.1109/ACCESS.2021.3129775.

[2] Khraisat, Ansam, et al. 'Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges'. Cybersecurity, vol. 2, no. 1, July 2019, p. 20. BioMed Central, https://doi.org/10.1186/s42400-019-0038-7.

[3] Prabhakaran, Selva. 'How to Detect Outliers with Z-Score'. Machine Learning Plus, 5 Aug. 2023, https://www.machinelearningplus.com/machine-learning/how-to-detect-outliers-with-z-score/.

[4] Yiu, Tony. 'Understanding Random Forest'. Medium, 29 Sept. 2021, https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

[5] PhD, Everton Gomede. 'Synthetic Minority Over-Sampling Technique (SMOTE): Empowering AI through Imbalanced Data Handling'. Medium, 11 Mar. 2024, https://pub.aimind.so/synthetic-minority-over-sampling-technique-smote-empowering-ai-through-imbalanced-data-handling-d86f4de32ea3.

[6] 'Aman Kharwal'. Medium, https://amankharwal.medium.com. Accessed 28 Apr. 2024.

[7] 'Papers-Literature-ML-DL-RL-AI/General-Machine-Learning/The Hundred-Page Machine Learning Book by Andriy Burkov/Links to Read the Chapters Online.Md at Master · Tirthajyoti/Papers-Literature-ML-DL-RL-AI'. GitHub.