Network Intrusion Detection Using Machine Learning

Abstract

This paper presents a comparative analysis of machine learning algorithms for network intrusion detection using the NSL-KDD dataset. I implemented and evaluated three different models: Naive Bayes, K-Nearest Neighbors (KNN), and a two-layer Neural Network. My preprocessing approach included feature type-specific transformations to optimize model performance. Despite class imbalance challenges, the models achieved promising results, with the Neural Network performing best at 77.9% accuracy.

1. Introduction

For my ML capstone, I wanted to take on the challenge of picking something that I am not familiar with or have no knowledge of. In the past, I've done projects on semantic analysis, question answering, and fine-tuning LLMs like CLIP to do certain tasks. I'm currently taking applied cryptography, and so far, it is very fascinating to me. So I wanted to use my capstone as an opportunity to explore how ML can be used in applied cryptography. Based on my research, ML is increasingly vital in cryptography and network security, particularly in intrusion detection systems (IDS). Machine learning algorithms can analyze network traffic patterns, identify anomalies, and detect potential security breaches in real-time. This intersection of ML and cryptography is particularly relevant as cyber threats become more sophisticated, requiring advanced detection methods beyond traditional rule-based systems.

The primary prediction task of this project is to develop a multi-class classification model that can identify the different types of network attacks from the NSL-KDD dataset. The model will distinguish five types of network flows: normal flows and four types of attacks—DoS, Probe, R2L, and U2R. There are three important reasons for this. First, as time goes by, cyber attackers are likely to become more sophisticated than ever before, with traditional signature-based detection methods missing a lot of attacks. Machine learning helps to find those subtle patterns and anomalies that might be indicators of new means to carry out an attack. Second, with the growing volume of network traffic across new systems, it is practically impossible to perform manual monitoring of such traffic. Last but not least, false positives in intrusion detection can cost a fortune, hence the acute need for highly accurate decision-making models that can separate an actual threat from a benign anomaly.

There are several significant challenges posed to the project. One of the big challenges is the inherent class imbalance, where normal traffic outnumbers some attack types, particularly U2R

and R2L. This creates a biased model where performance is particularly poor on under-represented attack types. Another challenge is feature selection and engineering. The dataset has 41 features that are highly correlated or may potentially be irrelevant for modern attack detection. Additionally, network attacks and patterns change with time, so it is critical to create models that generalize well to novel/unseen attack patterns. Finally, in real-world applications, predictions need to be made in real-time, which introduces the challenge of balancing model complexity against computational efficiency.

Through this project, I aim to address the following research questions:

1. How effectively can different machine learning algorithms classify network intrusion attempts?
2. What preprocessing techniques are most appropriate for network traffic data?
3. How do these models perform when dealing with class imbalance?

2. Dataset

2.1 NSL-KDD Dataset

The NSL-KDD dataset is an enhanced version of the original KDD Cup 1999 dataset, specifically designed to address some of the inherent problems in the original dataset. It consists of approximately 125,973 training records and 22,544 test records, with each record containing 41 features plus a class label. The features can be categorized into four main groups: basic features (9 features) derived from packet headers, content features (13 features) from packet payload analysis, time-based traffic features (9 features), and host-based traffic features (10 features).

This dataset was chosen for several reasons:

- It eliminates redundant records from the original KDD dataset, reducing the bias toward frequent attacks
- It includes a reasonable number of records, making it practical for experiments
- It provides a diverse range of attack types that represent real-world scenarios

2.2 Attack Categories

The dataset categorizes attacks into four main types:

Denial of Service (DoS): These attacks aim to make resources unavailable to legitimate users by flooding the target with excessive traffic or connection requests. Examples include "neptune," "smurf," and "back" attacks. For instance, in a neptune attack, an attacker sends SYN packets to a server but never completes the TCP handshake, exhausting server resources.

Probe: These attacks involve scanning networks to gather information for potential future attacks. Examples include "portsweep," "ipsweep," and "nmap." A typical scenario involves an attacker using nmap to scan open ports on target systems to identify vulnerabilities.

Remote to Local (R2L): These attacks involve unauthorized access from a remote machine. Examples include "guess_passwd," "ftp_write," and "multihop." In a guess_passwd attack, an attacker attempts to gain access by repeatedly trying different passwords.

User to Root (U2R): These attacks involve legitimate users attempting to gain administrator privileges. Examples include "buffer_overflow," "rootkit," and "loadmodule." In a buffer_overflow attack, the attacker exploits a programming vulnerability to execute malicious code with elevated privileges.

2.3 Data Exploration

Training set (after mapping to attack types):

- DoS: 45,927 samples (67.7%)
- Normal: 13,449 samples (19.8%)
- Probe: 7,458 samples (11.0%)
- R2L: 995 samples (1.5%)
- U2R: 52 samples (0.1%)

Testing set (after mapping to attack types):

- DoS: 7,458 samples (36.6%)
- Normal: 9,711 samples (47.7%)
- Probe: 2,421 samples (11.9%)
- R2L: 2,754 samples (13.5%)
- U2R: 67 samples (0.3%)

This imbalance poses a significant challenge, particularly for minority classes like U2R, which represents only 0.1% of the training data. I considered this imbalance during model evaluation by using weighted metrics rather than relying solely on accuracy.

3. Methodology

3.1 Preprocessing

One of the key challenges was preprocessing the diverse feature types in the dataset. I identified three categories of features:

Categorical features: protocol_type, service, flag
Binary features: land, logged_in, root_shell, su_attempted, is_hot_login, is_guest_login
Numerical features: The remaining 32 features

I applied type-specific preprocessing techniques:

1. **Categorical features:** One-hot encoding using sklearn's OneHotEncoder
2. **Binary features:** Kept unchanged as they were already in binary format
3. **Numerical features:** Standardized using sklearn's StandardScaler to have zero mean and unit variance

For the Naive Bayes model, which requires binary input for the Bernoulli variant, I binarized all features using a threshold of 0.3 (converting values > 0.3 to 1, and ≤ 0.3 to 0).

For the target variable, I first mapped each specific attack to its corresponding attack type (DoS, Probe, R2L, U2R, or normal) and then encoded these categories using sklearn's LabelEncoder.

3.2 Model Implementation

I implemented three different models:

Naive Bayes: I implemented a Bernoulli Naive Bayes model from scratch. The implementation includes:

- Maximum likelihood estimation for training
- Laplace smoothing to handle zero probabilities
- Log-probabilities for numerical stability

K-Nearest Neighbors: I used sklearn's KNeighborsClassifier with k=7 neighbors.

Neural Network: I implemented a two-layer neural network with:

- Input layer with size matching the feature dimension
- Hidden layer with 128 neurons and sigmoid activation
- Output layer with 5 neurons (one per class) and softmax activation
- Cross-entropy loss function
- Mini-batch gradient descent with a batch size of 128
- Learning rate of 0.001 and 50 training epochs

4. Challenges and Solutions

Throughout this project, I encountered several challenges:

Challenge 1: Feature preprocessing Initially, I was uncertain whether to use sklearn's OneHotEncoder or LabelEncoder for categorical features. I experimented with both and found that one-hot encoding provided better model performance, especially for the neural network.

Solution: I implemented a comprehensive preprocessing pipeline that handled each feature type appropriately, using one-hot encoding for categorical features, standardization for numerical features, and maintaining binary features unchanged.

Challenge 2: Class imbalance The severe class imbalance in the dataset posed a significant challenge, especially for minority classes like U2R.

Solution: I used weighted evaluation metrics (precision, recall, and F1-score) to better assess model performance across all classes.

Challenge 3: Binarizing continuous features for Naive Bayes My Naive Bayes implementation was designed for binary features, but many features in the dataset were continuous.

Solution: I implemented a thresholding function to convert continuous features to binary, setting values above 0.3 to 1 and values below to 0.

Challenge 4: Adapting the Neural Network from Regression to Classification

Initially, my neural network was set up for regression tasks, similar to those in Homework 10, so I had to modify it to handle classification instead from regression to classification

Solution: I modified the output layer to use softmax activation and implemented cross-entropy loss instead of mean squared error. I also adjusted the backward propagation to handle multi-class classification.

5. Results

I evaluated all three models using accuracy, precision, recall, and F1-score metrics:

Naive Bayes:

- Accuracy: 69.3%
- Precision: 74.5%
- Recall: 69.3%
- F1-score: 67.9%

K-Nearest Neighbors:

- Accuracy: 77.8%
- Precision: 82.3%
- Recall: 77.8%
- F1-score: 73.6%

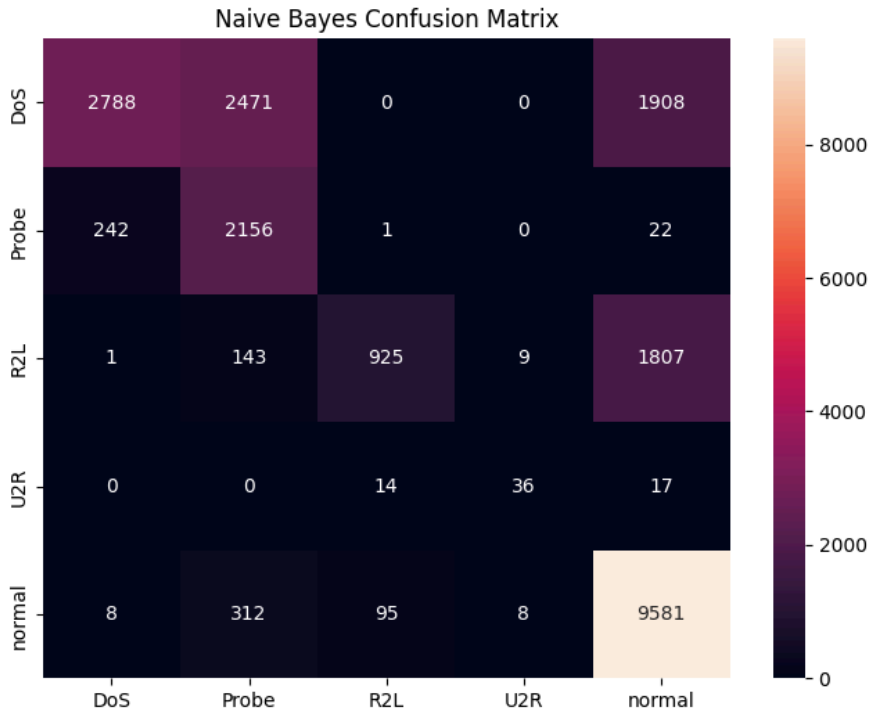
Neural Network:

- Accuracy: 77.9%
- Precision: 78.8%
- Recall: 77.9%
- F1-score: 76.4%

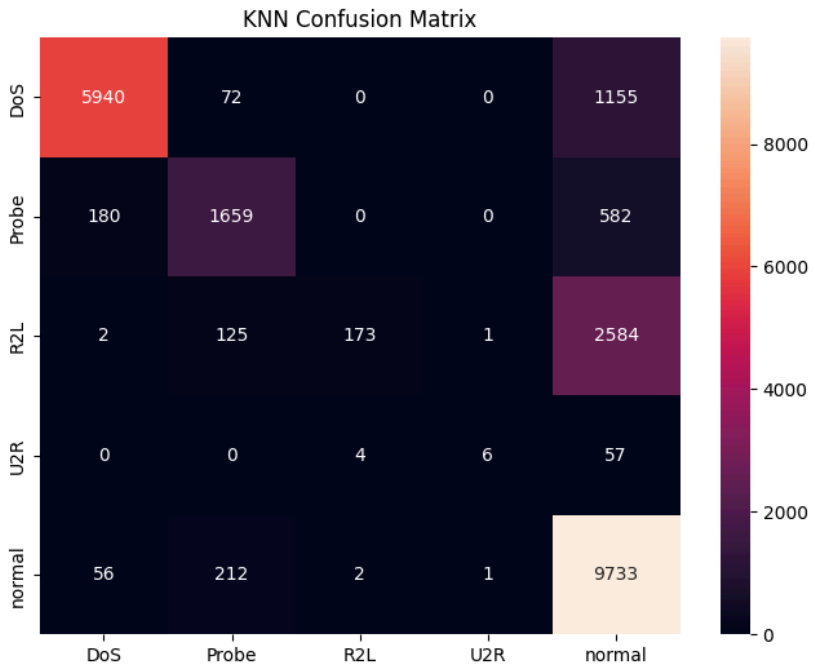
The neural network achieved the best overall performance, followed closely by KNN. The Naive Bayes model performed adequately but was less effective at handling the complexity of the data.

Confusion matrix analysis revealed that all models struggled with the minority classes (U2R and R2L), often misclassifying them as normal traffic. This highlights the challenge of detecting rare but potentially severe attacks.

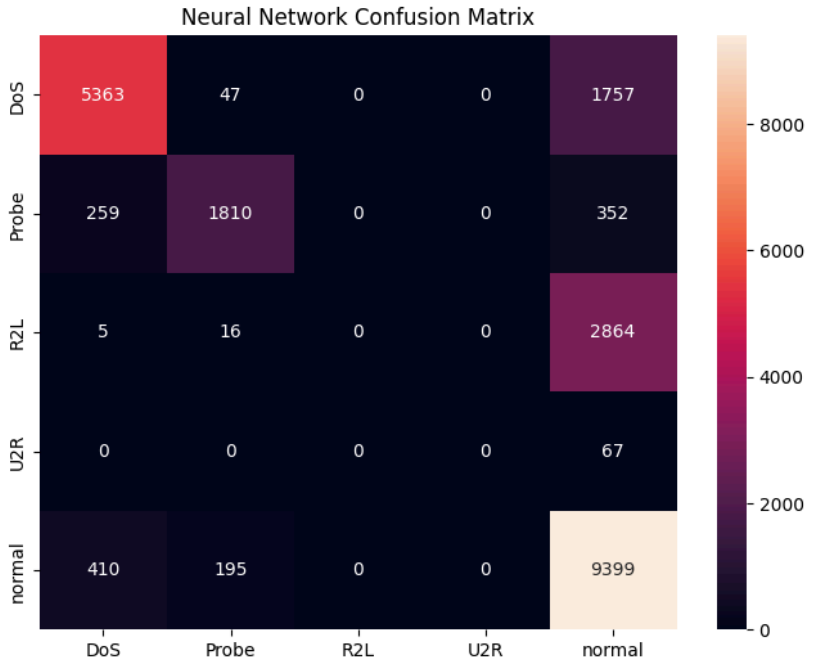
Naive_Bayes Confusion Matrix:



KNN confusion Matrix:



Neural Network confusion matrix:



6. Future Improvements

If continuing this work, several improvements could be made:

1. **Address class imbalance:**Somehow address the class imbalance

2. **Feature engineering:** Perform more in-depth feature selection and creation to better capture attack patterns, possibly using techniques like Principal Component Analysis.
3. **Model enhancements:**

    - Maybe experiment with other models to yield better results.

7. Conclusion

I evaluated three machine learning approaches for network intrusion detection. The neural network achieved the best performance, demonstrating the potential of deep learning for cybersecurity applications. However, all models struggled with the significant class imbalance in the dataset, particularly with rare attack types.

Future work should focus on addressing class imbalance, feature engineering, and exploring ensemble and deep learning models. The findings suggest that machine learning can effectively contribute to network security but requires careful consideration of data characteristics and model selection.

References

1. *A Study on NSL-KDD Dataset for Intrusion Detection ...*, e-tarjome.com/storage/btn_uploaded/2019-07-13/1563006133_9702-etarjome-English.pdf. Accessed 13 May 2025.
2.