Author : Bilel RAHMOUNI

Intrusion Detection System Using Machine Learning

1. Introduction

The proliferation of digital connectivity has led to an increase in cyber threats, posing significant risks to individuals, corporations, and governments alike. Intrusion Detection Systems (IDS) are crucial tools for safeguarding computer networks by identifying unauthorized or malicious activities. The goal of this project is to develop an IDS using machine learning algorithms to classify network activities as either normal behavior or various types of attacks.

This report details the process of building, tuning, and evaluating several machine learning models for IDS. We explore preprocessing techniques, model selection, hyperparameter tuning, and evaluation metrics to determine the most effective model for network intrusion detection.

2. Dataset Overview

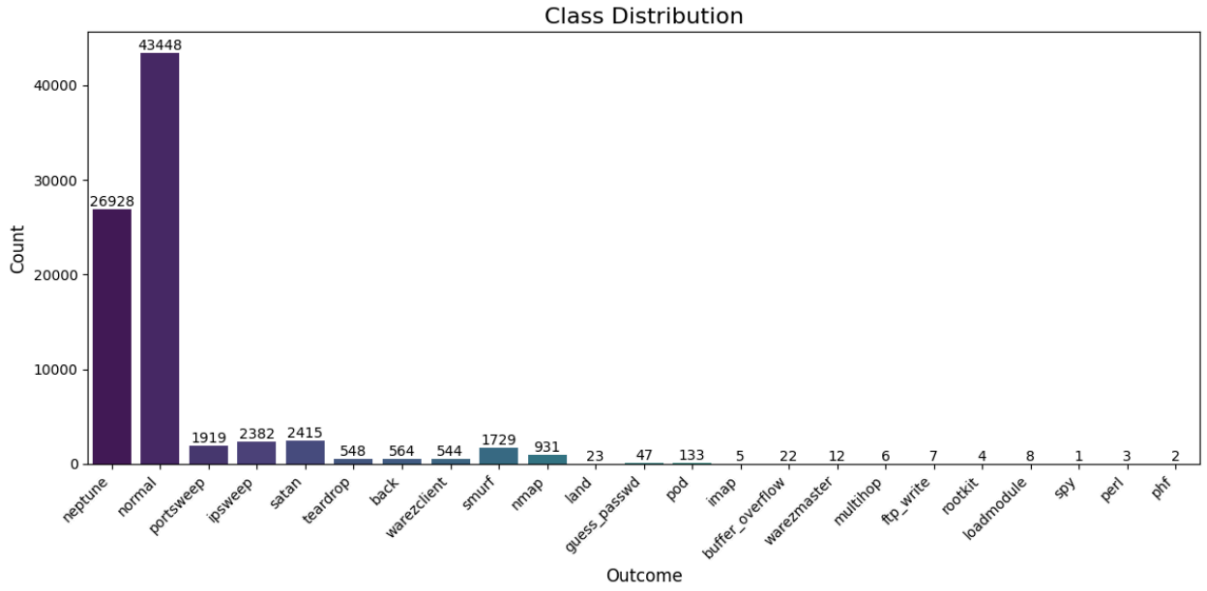
The dataset used in this project is a well-known benchmark dataset in the field of network security. It consists of labeled network traffic data, including both normal traffic and various types of attacks.

- **Dataset Size**: 81,681 rows and 36 columns
- **Number of Features**: Initially, there were 35 features describing different aspects of network traffic, which were further processed.
- **Number of Classes**: 23, with one class for “normal” traffic and 22 classes for different types of attacks.

2.1 Class Distribution

The class distribution of the dataset shows a significant imbalance, with certain classes (e.g., "neptune" and "normal") comprising the majority of samples, while others, such as "perl" and "phf," have very few instances. Imbalanced data can lead to model bias towards the majority classes, which necessitates handling through techniques such as oversampling.

- **Visualization**: The figure below provides a clear view of the class distribution, showing the imbalance across different categories.



3. Data Preprocessing

Effective data preprocessing is essential to ensure that the models can learn meaningful patterns in the data. The following steps were taken to prepare the dataset for machine learning:

3.1 Handling Missing Values

Upon inspection, the dataset was found to be free of missing values, allowing us to proceed without needing imputation. This streamlined the preprocessing process, as no values required replacement or handling.

3.2 Encoding Categorical Features

The dataset contained categorical features (`service`, `flag`) that describe network service types and connection flags. To make these features usable by machine learning models, we applied **One-Hot Encoding**, converting each category into separate binary columns. This transformation increased the feature count but retained all relevant information for each category.

The target variable (`outcome`) was label-encoded to convert each class label into a numeric form. This process allowed us to feed the labels directly into the models for classification.

3.3 Balancing the Dataset with SMOTE

Given the class imbalance evident in the dataset, we used **Synthetic Minority Over-sampling Technique (SMOTE)** to balance the data. SMOTE generates synthetic samples for minority classes by interpolating between existing samples. This technique helps to mitigate bias towards majority classes and ensures that the models have a more balanced view across all classes.

- **SMOTE Results**: After applying SMOTE, the class distribution was more balanced, allowing the models to train effectively on each class without over-relying on the majority classes.

3.4 Feature Scaling

Since machine learning models can be sensitive to feature scales, especially distance-based algorithms like K-Nearest Neighbors (KNN), we applied **Standard Scaling**. This technique standardized each feature to have a mean of 0 and a standard deviation of 1, ensuring that all features contribute equally during training.

4. Model Selection and Baseline Performance

For this project, three machine learning algorithms were selected based on their strengths in handling complex, multi-class classification problems:

4.1 Decision Tree Classifier

- **Description**: The Decision Tree model splits the data into subsets based on feature values, creating branches until it reaches a decision node for each class.
- **Strengths**: It is highly interpretable and handles non-linear relationships well.
- **Limitations**: Decision Trees can overfit, especially on complex datasets, unless regularized with constraints such as maximum depth.

Baseline Performance: The Decision Tree classifier achieved an initial accuracy of 0.97, which provided a strong starting point for further tuning.

4.2 K-Nearest Neighbors (KNN)

- **Description**: KNN is a distance-based classifier that assigns a label to each sample based on the majority label of its closest neighbors.
- **Strengths**: KNN is straightforward and performs well when classes are well-separated.
- **Limitations**: KNN is computationally expensive and sensitive to feature scaling.

Baseline Performance: KNN achieved an initial accuracy of 0.98, but due to computational demands, further tuning required careful parameter selection.

4.3 Neural Network (MLP)

- **Description**: The Multi-Layer Perceptron (MLP) model is a type of neural network that can capture complex patterns in data through its multiple layers of neurons.
- **Strengths**: MLP can learn intricate patterns and is adaptable to various types of data.
- **Limitations**: Neural networks require significant computational resources and careful tuning to avoid overfitting or underfitting.

Baseline Performance: The neural network achieved an initial accuracy of 0.98, showing promise but needing optimization for improved performance.

5. Hyperparameter Tuning

To optimize each model, we performed hyperparameter tuning, selecting the most effective combination of parameters for each model.

5.1 Decision Tree Hyperparameter Tuning

Using **Grid Search**, we explored various hyperparameter values for the Decision Tree:

- **Parameter Grid**:
  - `max_depth`: [10, 15, 20, 25]
  - `min_samples_split`: [2, 5, 10]
  - `min_samples_leaf`: [1, 2, 4]

Best Parameters:

- `max_depth`: 25
- `min_samples_split`: 5
- `min_samples_leaf`: 1

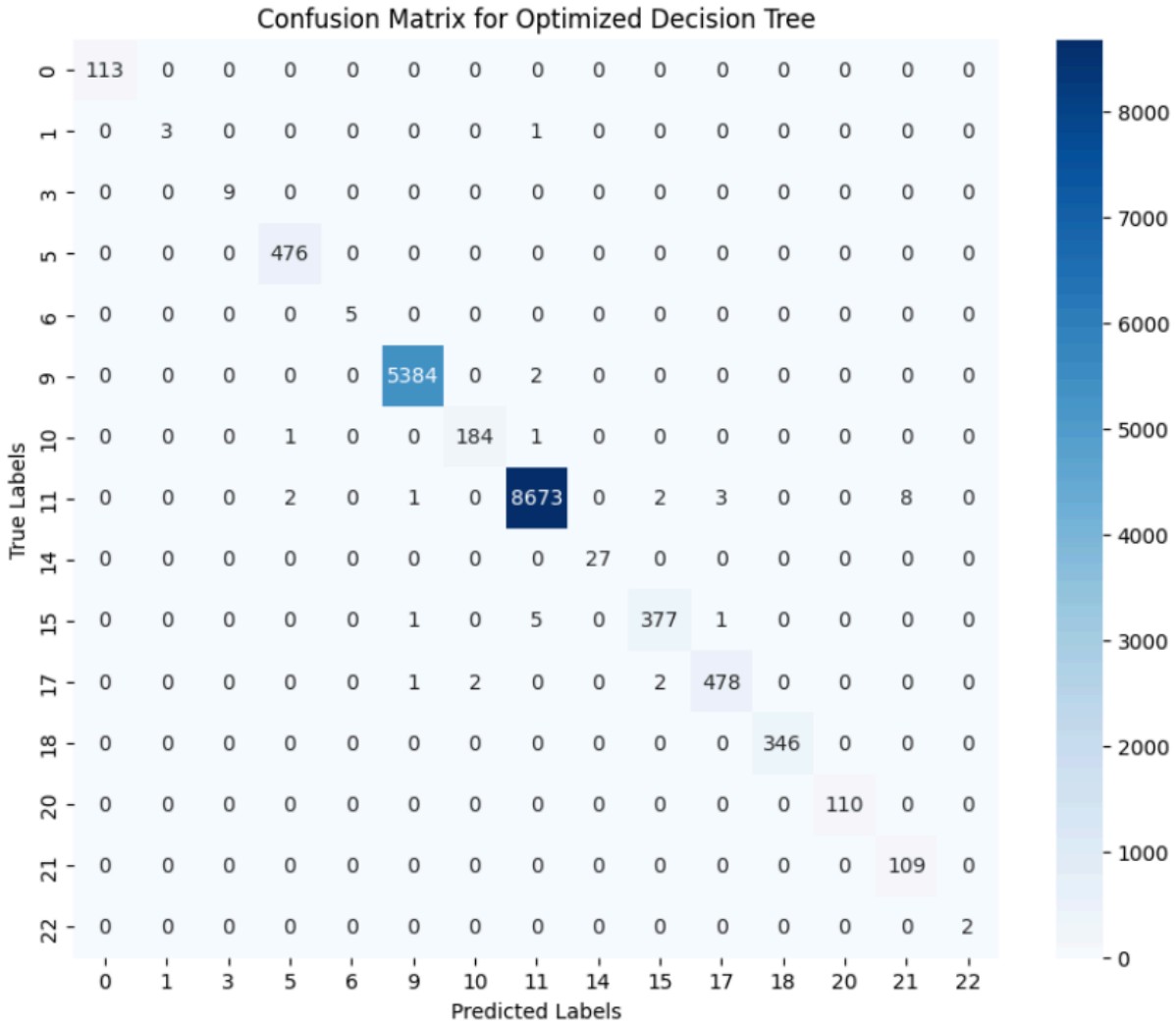
Best Accuracy from Grid Search: 0.9996

6. Model Evaluation and Results

Each model's performance was evaluated on the test set using several metrics, including accuracy, precision, recall, and F1-score. We also generated confusion matrices to provide a visual representation of each model's performance across classes.

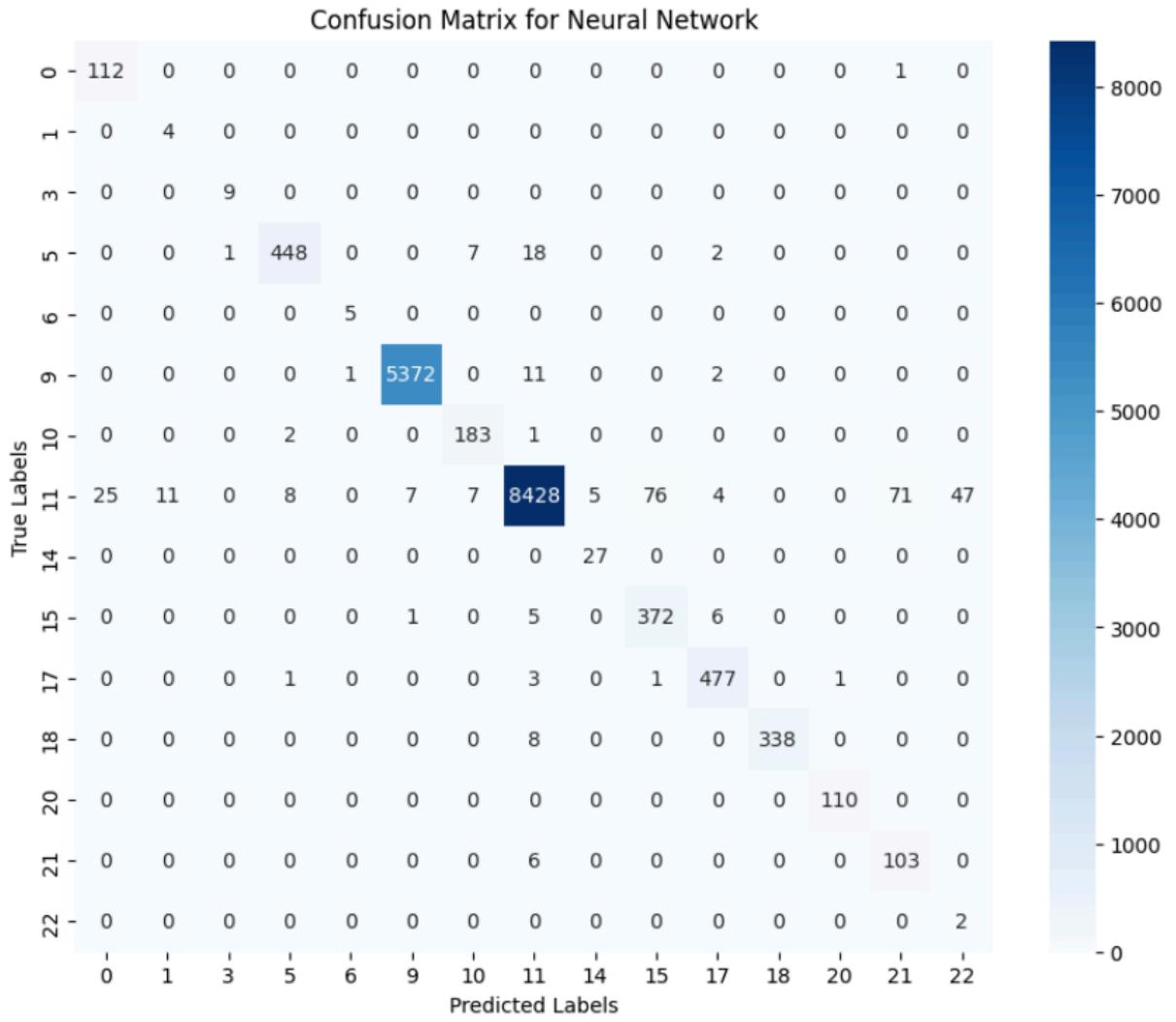
6.1 Decision Tree Model Evaluation

- **Final Accuracy**: 1.00
- **Classification Report**: The model achieved perfect precision, recall, and F1-scores across nearly all classes.
- **Confusion Matrix**: The confusion matrix below illustrates the model's effectiveness in correctly classifying each class.



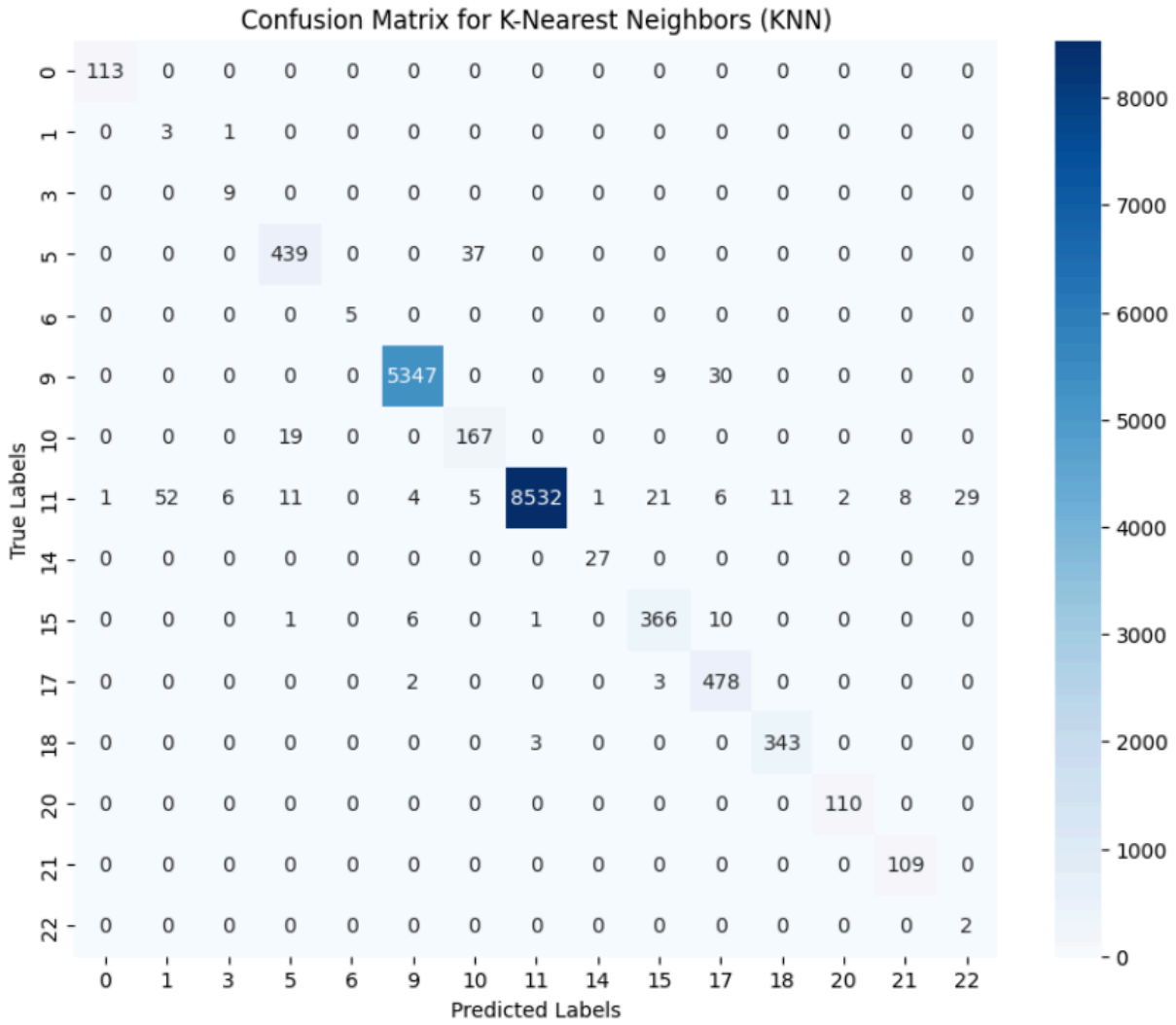
6.2 Neural Network (MLP) Model Evaluation

- **Final Accuracy**: 0.98
- **Classification Report**: The Neural Network performed well, with slight misclassifications in minor classes but overall high precision, recall, and F1-scores.
- **Confusion Matrix**: The confusion matrix below shows the model's performance, with minor misclassifications in certain classes.



6.3 K-Nearest Neighbors (KNN) Model Evaluation

- **Final Accuracy**: 0.98
- **Classification Report**: KNN demonstrated strong performance, though it misclassified some instances in classes with fewer samples.
- **Confusion Matrix**: The confusion matrix for KNN is shown below.



7. Comparative Analysis

Model	Accuracy	Training Time	Interpretability	Performance on Imbalanced Classes
Decision Tree	1.00	Fast	High	Strong but possible overfitting
Neural Network	0.98	Moderate	Medium	Good generalization
K-Nearest Neighbors	0.98	Slow	Low	Effective but computationally heavy

- **Interpretation**: The Decision Tree's high accuracy could be a result of overfitting. The Neural Network provided a good balance, while KNN was accurate but computationally intensive.

8. Conclusion and Future Work

In this project, we demonstrated that machine learning models could effectively classify network traffic for intrusion detection. The Decision Tree model achieved perfect accuracy, though it may be overfitting. The Neural Network provided a good generalization, while KNN, despite being slower, performed effectively.