Quantitative check on Machine Learning Algorithms for Intrusion Detection Systems.

Adeyemi Fagbade

Abstract:

Intrusion detection systems (IDS) are essential to securing modern networks, and recent advances in machine learning (ML) have enabled IDS to evolve from signature-based approaches to more adaptive, data-driven solutions. This report encapsulates a rigorous quantitative evaluation of advanced machine learning (ML) algorithms applied to intrusion detection systems (IDS). We focus on evaluating the long-term performance of ML-based IDS. By systematically benchmarking models including decision trees (DT), random forest (RF), support vector machines (SVM), ensemble methods, artificial neural network (ANN), and deep neural networks against standard IDS datasets (such as KDD Cup '99 and NSL-KDD). We assess key performance metrics like accuracy, precision, recall, and F1-score. Our study further investigates the effects of data imbalance and feature selection, offering a clear analysis of the trade-offs between computational efficiency and classification performance.

The experimental findings reveal that while many ML-based IDS demonstrate near-perfect detection rates under controlled test conditions, their performance can vary significantly in real-world scenarios characterized by diverse and evolving attack patterns. In particular, our computational simulation shows that RFM and XGB are most resistant to overfitting. Besides that, our experiment results also show that SVM linear and NB suffer the most from overfitting, although they perform well on the training dataset

The report not only highlights the statistical robustness and scalability of various algorithms but also identifies potential challenges in model generalization and overfitting. Ultimately, our work provides a comprehensive framework for benchmarking and deploying ML algorithms in IDS, thereby guiding practitioners in optimizing their systems for both precision and rapid response in today's dynamic cybersecurity landscape.

1.0 Introduction:

Intrusion Detection Systems (IDS) are pivotal in fortifying an organization's cybersecurity posture alongside traditional measures such as network access controls, antivirus software, and Virtual Private Networks (VPNs). IDS technologies monitor network or host activities to identify potential breaches by analyzing traffic patterns or system behavior, Liao et al. [1]. Depending on their deployment, IDS can be classified into network-based (NIDS), which scrutinizes network traffic, and host-based (HIDS), which focuses on system files. Moreover, IDS can operate based on predefined signatures or behavioral anomalies, thereby differentiating between known threats and deviations from normal activity patterns [2-3].

Signature-based IDS, often termed misuse detection systems, rely on databases of known threat signatures to flag malicious activities, offering the advantage of minimal false positives [4]. However, their dependency on previously identified attack patterns limits their efficacy against novel or modified threats. In contrast, anomaly-based IDS establish a baseline of normal network behavior and flag deviations as potential intrusions, which enhances their ability to detect zero-day and unseen attacks, albeit sometimes at the cost of higher false-positive rates [5]. This inherent trade-off underscores the necessity for more adaptive detection methodologies.

Recent advancements in machine learning (ML) have been instrumental in addressing these limitations by enabling IDS to learn and generalize from network data. Techniques such as Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), Artificial Neural Networks (ANN), and Deep Neural Networks (DNN) have demonstrated promising improvements in both accuracy and adaptability when distinguishing benign from malicious traffic [6-7]. In this report, I present a quantitative evaluation of these ML-based IDS approaches using datasets such as **NSL-KDD dataset** to simulate realistic network environments and benchmark performance across different algorithmic paradigms.

1.1 Literature Review:

The rapid expansion of the Internet has been accompanied by a notable increase in cyber-attacks. For instance, in the USA, in 2024, there was an average of roughly 4,000 to 5,000 cyber attacks per day, which could amount to over 1.5 million incidents annually for businesses. For government agencies, several sources indicate a significant increase in targeted attacks with some reports suggesting an annual rise of 30–40% compared to previous years [8]. This growing threat landscape has underscored the necessity for robust security solutions, among which

Intrusion Detection Systems (IDS) play a critical role in identifying and mitigating unauthorized activities.

IDS are designed to monitor network traffic or system behavior in real-time to detect malicious actions. They serve a similar purpose to firewalls by safeguarding the confidentiality, integrity, and availability of information systems [9]. Fundamentally, IDS implementations are categorized based on their detection methodology: signature-based systems, which rely on stored patterns of known attacks, and anomaly-based systems, which detect deviations from established normal behavior [10]. Each approach presents its own set of advantages and challenges.

Machine learning (ML) techniques have increasingly been integrated into IDS to overcome these limitations. ML's capability for pattern recognition and adaptive learning has not only revolutionized email spam filtering where approaches such as Artificial Neural Networks (ANN) have dramatically reduced unwanted messages [11-12] but also enhanced the detection of sophisticated cyber threats. A variety of ML algorithms, including Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), K-Nearest Neighbors (KNN), and various forms of Naïve Bayes, have been explored for their efficacy in distinguishing between benign and malicious network activity [13]. Recent studies indicate that combining these techniques in hybrid systems can further improve detection rates while minimizing false positives [14].

Building on these advancements, the present study evaluates six ML algorithms—five supervised and one unsupervised to determine their performance in both binary and multiclass classification tasks within IDS environments. By employing a comprehensive set of metrics such as accuracy, precision, recall, and F-measure, our analysis seeks to provide clear insights into each algorithm's strengths and limitations. This work not only offers a benchmark for the current state-of-the-art in ML-driven IDS but also serves as a valuable reference for researchers and practitioners striving to enhance intrusion detection capabilities in ever-evolving cyber landscapes.

1.3 Project Description (Software vs. Hardware)

This project will primarily be software-based. While hardware accelerators (e.g., GPUs, TPUs) can speed up ML computations, the core of the solution lies in creating and training classification or anomaly detection models. The focus is on the development, integration, and testing of ML algorithms within an existing or custom-built IDS framework. Hardware considerations may arise if high throughput or real-time analysis is required (especially in enterprise-scale networks), but the main thrust of research is on the software algorithms and their efficacy.

Tools, Datasets, and Requirements

1. **Traffic Analysis**: Tools like Wireshark or TShark for collecting and labeling real-world network traffic.
2. **Datasets**: Public repositories such as KDD Cup or NSL-KDD, for initial model training and benchmarking.
3. **Machine Learning Libraries**: Frameworks like scikit-learn, TensorFlow, or PyTorch for building, training, and tuning ML models.
4. **Event Management and Logging**: Platforms (e.g., SIEM solutions) for monitoring alerts and visualizing intrusion events.

By leveraging these tools, the research will demonstrate how machine learning can efficiently identify anomalies in real or simulated network environments. Ultimately, the study aims to validate ML's effectiveness in producing speedy, robust, and scalable intrusion detection to further reinforcing its indispensability in the evolving cybersecurity landscape.

2. Research Methodology:

2.1 Dataset

The availability of high-quality datasets is one of the biggest challenges in the domain of Intrusion Detection Systems (IDS). Due to privacy and security concerns, most organizations do not share their actual network traffic data. However, a reliable dataset is crucial for developing and evaluating anomaly-based IDS solutions.

In this report, **we adopt the NSL-KDD dataset [15]**. This dataset was published in 2009 as an updated version of the KDD99 dataset (originally from 1998/1999), in response to the issues highlighted by McHugh et al [15a]. NSL-KDD addresses several shortcomings found in KDD99, namely:

- **Removed redundant records** from the training set so that the classifiers will not be biased toward repeated records.

- **Removed duplicated records** from the test set, ensuring performance is not inflated by methods that simply memorize frequent samples.
- **Introduced record selection based on a difficulty measure** to emphasize the harder samples/classes. This leads to a more challenging and discriminative dataset.

For convenience and clarity, we manage the NSL-KDD dataset under different versions reflecting various preprocessing stages:

- **V1:** The base dataset in CSV format, as originally downloaded from the NSL-KDD repository.
- **V2:** A cleaning step that converts CSV files to Parquet format. (During cleaning, any anomalies would be addressed, but NSL-KDD is already known to have zero missing records.)
- **V3:** A reorganized version that saves storage space by retaining only the original CSV files in V1/V2. Redundant copies are removed.
- **V4:** An updated release that excludes certain contaminating features ([presentation] & [conference article]) not relevant to experimental classification. All data types are set correctly, and no records contain missing information.

Although, this new version of the KDD data set (BSL-KDD) still suffers from some of the problems discussed by McHugh and may not be a perfect representative of existing real networks, because of the lack of public data sets for network-based IDSs, I believe it still can be applied as an effective benchmark data set to help researchers compare different intrusion detection methods.

By applying straightforward baseline classifiers and initial treatments, the data already established an **AUROC score of 94.6%** on NSL-KDD, indicating that relatively simple models can perform well on this dataset.

The NSL-KDD dataset also provides the following advantages over the older KDD99 dataset:

1. **No redundant records** in the training set, so learning algorithms do not overfit to repeated patterns.
2. **No duplicate records** in the test sets, preventing artificially high accuracies for methods memorizing duplicates.
3. **Proportional coverage of difficulty levels:** record selection is inversely proportional to the percentage of records in the original dataset, introducing more challenging samples and broader difficulty variation.
4. **Manageable size:** the number of train/test records is sufficient to run experiments on the full dataset without random subsampling. Consequently, results from different research efforts on NSL-KDD are more consistent and comparable.

2.2 Machine Learning Models

Machine Learning (ML) has gained considerable attention for enhancing the capabilities of IDS [16], especially in detecting unknown or zero-day attacks. Below is a brief overview of six popular classification models often used for IDS.

2.2.1 Decision Tree

A Decision Tree (DT) classifies samples via a tree-like structure, where each internal node tests a feature and each branch corresponds to an outcome [17]. The final leaves represent class labels. DTs are typically trained with metrics such as Gini impurity or Entropy to decide which features best split the data. DTs are popular in IDS research because they balance good accuracy, efficiency, and interpretability. Fig. 1 characterizes how decision tree works and constructed. During each iteration of this iterative process, the data is split based on a chosen feature, resulting in leaf nodes that contain homogeneous subsets. The selection of each feature is guided by its ability to produce more uniform splits, typically assessed using metrics such as Gini or Entropy. In general, features nearer to the root node exhibit stronger correlations with the predicted outcome, which also aids in feature selection.

Once the tree is built, each test sample is routed from the root to a leaf node. The leaf node where a sample terminates designates its predicted class. Decision trees are widely used in Intrusion Detection Systems (IDS) because they are straightforward to train, provide high accuracy, and offer clear insights into how features influence classification.

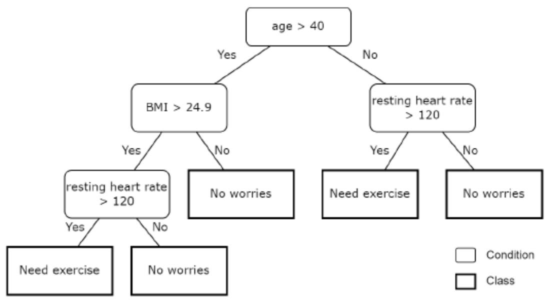


Figure 1: Decision tree creation process.

2.2.2 Random Forest

Random Forest (RF) is an ensemble of Decision Trees [18]. Each tree is trained on a bootstrapped subset of the training data, and the final classification is decided by majority vote as described in Fig 2. RFs tend to reduce overfitting, since each individual tree sees a slightly different subset of samples and features. This makes RFs robust to noise and an appealing choice for IDS.

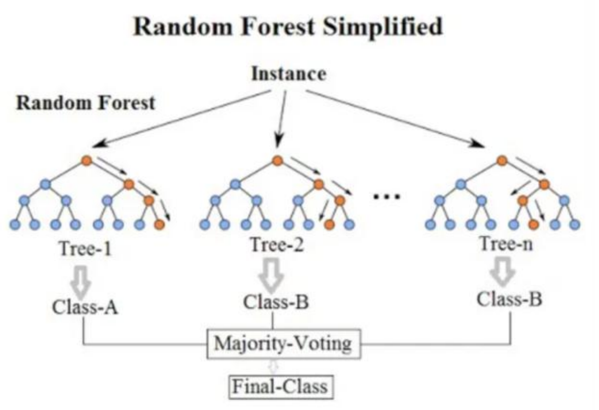


Fig.2 : Demystifying the Random Forest Algorithm process.

2.2.3 Support Vector Machine

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm used for classification and regression tasks. A SVM finds a hyperplane that separates classes with a maximal margin [19] (See Fig. 3). The dashed line characterizes a separator in a 2-dimensional space that separates the data of two different classes. By employing kernel functions (e.g., RBF kernel), SVMs can as well handle non-linear boundaries in a higher-dimensional feature space. Although they can be more computationally expensive, SVMs often exhibit strong generalization performance and have been widely adopted for intrusion detection tasks.

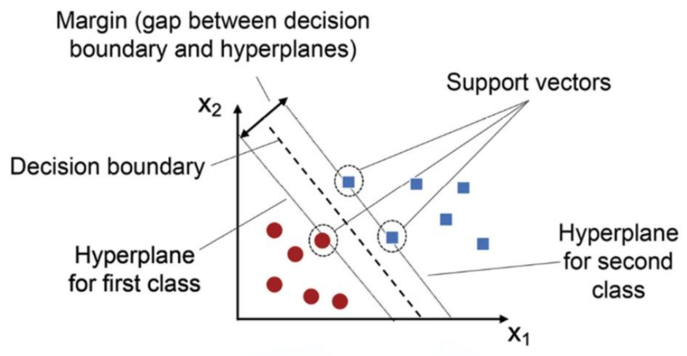

Margin (gap between decision boundary and hyperplanes)
Support vectors
X_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
X_1

Fig. 3: A simple SVM in 2D space.

2.2.4 Naïve Bayes

Naïve Bayes (NB) classifiers apply Bayes' theorem under a simplifying assumption of independence among features [20]. The general well-know equation of the Bayes theorem is given as:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Implying that the probability of A given B is equal to the probability of B given A times the probability of A and divided by the probability of B. In other words, we can calculate P(A | B) if P(B | A), P(A) and P(B) is known. Therefore, the theorem can calculate the probability of data being in each class based on the given information and classify the data. In the training stage, the necessary probabilities are computed using the training data. As probability calculation is a simple task, the model is very efficient. Although the model is "naïve" as it assumes that each feature is independent of the other, it sometimes achieves comparable accuracy to other models. Despite this naïve assumption, NB can yield competitive results in many classification problems and is typically fast to train. However, NB can underperform when the independence assumption is severely violated or when data distributions are very complex.

2.2.5 Artificial Neural Network

Artificial Neural Networks (ANNs) consist of interconnected layers of "neurons," each calculating a weighted sum of inputs that is then passed through a nonlinear activation function [21](See Fig 4.). A typical network includes three main types of layers:

1. Input Layer: Receives raw data.
2. Hidden Layer(s): Uses weights, biases, and activation functions to learn complex patterns.
3. Output Layer: Produces the network's final predictions.

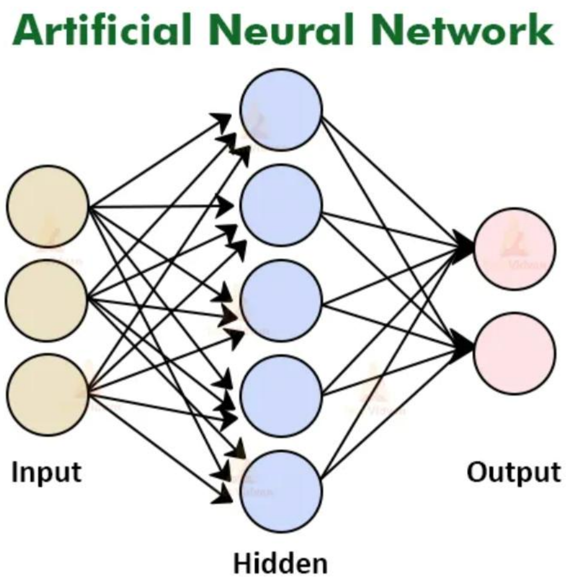
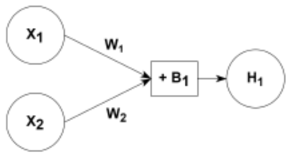


Fig. 4 : Illustration of ANN architecture.

The mapping illustrates below, which is represented with the equation:

$$h_i(x) = f(w_i^T x + b)$$



where x denotes the inputs to the next layer, wi represents the weights on the vertices, b denotes the biases, and f denotes the activation function.

During training, the goal is to optimize the weights and biases so that the network accurately maps inputs to outputs. Because ANNs often contain many parameters, they require significant computational power and large datasets to train effectively without overfitting.

A key advantage of ANNs is their ability to automatically extract features from raw data, reducing the need for hand-designed features. As the input travels through multiple layers (perceptrons), the network transforms the data into increasingly abstract representations, enabling it to capture intricate relationships that simpler methods (e.g., decision trees, naïve Bayes) may overlook. However, this flexibility comes at a higher computational cost, making ANNs more resource-intensive to train.

In an Intrusion Detection System (IDS) context, ANNs' capacity to learn complex patterns from raw network data can greatly reduce reliance on manually crafted features—provided sufficient data and computing resources are available.

2.2.6 Deep Neural Network

Deep Neural Networks (DNNs) build upon ANNs by stacking multiple hidden layers [21-24] (see Fig. 4a). This depth allows them to learn more complex functions and higher-level feature abstractions, often improving performance. However, DNNs can also be more prone to overfitting if the dataset is too small and may require extensive hyperparameter tuning and computational resources.

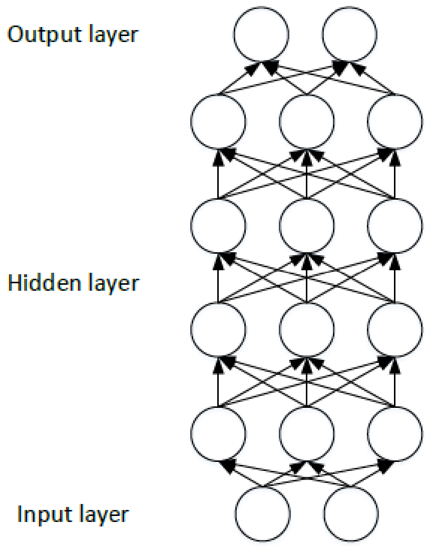


Fig. 4a: The DNN structure.

2.3 Feature Selection

Feature selection can significantly impact model performance. High-dimensional datasets can increase computational cost and risk overfitting. Common approaches include:

- Ranking feature importance via ensemble methods such as **Random Forest**, which can calculate how much each feature contributes to the classification.
- Manually removing features that might cause overfitting (e.g., IP addresses or timestamps unique to a specific environment).

By selecting only the most relevant features, one often reduces noise, shortens training time, and improves overall predictive accuracy.

3 Framework of the Experiment

This section describes the general experimental framework as shown in Fig. 5 and adapted to NSL-KDD, to reflect how datasets are processed, how models are trained and optimized, and how results are evaluated. The sequence of the steps are:

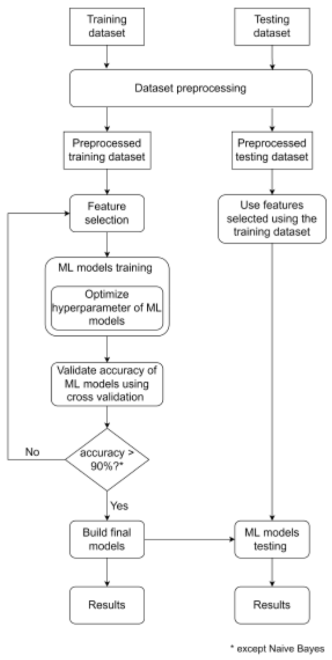

Training dataset
Testing dataset
Dataset preprocessing
Preprocessed training dataset
Preprocessed testing dataset
Feature selection
Use features selected using the training dataset
ML models training
Optimize hyperparameter of ML models
Validate accuracy of ML models using cross validation
No
accuracy > 90%?*
Yes
Build final models
ML models testing
Results
Results
* except Naive Bayes

Fig. 5: The Numerical experiment workflow.

A. **Data Pre-Processing**

  - **Cleaning:** Since NSL-KDD already excludes missing values, no additional imputation is needed. However, any inconsistent or unwanted records (e.g., if they arose during additional labeling steps) are removed.
  - **Format Conversion & Organization (V1–V4):**
    - V1: the original CSV files.
    - V2: cleaned Parquet files for efficient I/O.
    - V3: reorganized storage, retaining only essential CSVs.
    - V4: updated feature set, removing contaminating or irrelevant features.

B. **Feature Selection**

It is imperative to perform feature selection before training the model. Since the revised NSD-KDD dataset contain around 43 features, training the model without selective features will cost much more time. Besides, some features may include noise and hence reduce the accuracy of the model. The following approaches are applied to select features according to the set objectives:

  - **Random Forest Ranking:** A Random Forest algorithm (through RandomForestClassifier) can rank features by their predictive importance. In this case a random forest is trained with the training data and the top n features with the highest importance scores are selected for modeling.
  - **Refinement:** A brute force approach or domain knowledge can further refine the final feature subset by iteratively excluding less-relevant features (e.g., IP addresses or timestamps).

C. **Model Training**

Selective models are train using the training portion of the dataset with the selected features. In each training with each model, we optimize the hyperparameters through the use of grid-search algorithm called GridSearchCV function.

  - **Hyperparameter Tuning:** For each classifier (DT, RF, SVM, NB, ANN, DNN), a grid search (or other optimization techniques) is performed to identify parameters (e.g., depth of trees, number of hidden layers, kernel settings). Only a fraction of the training set may be used initially to expedite tuning.

D. Cross-Validation

The goal is to verify and validate the accuracy of the applied and selected models. The performance of these models can be measured through cross-validation algorithms, that include using:

- **K-Fold:** The training data is commonly split into *k* folds. Each model is iteratively trained on *k–1* folds and validated on the remaining fold. The average performance across folds helps detect overfitting or high variance.

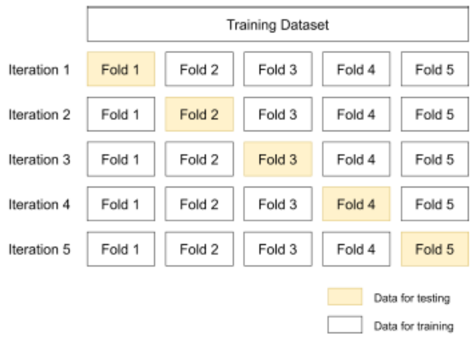


Fig. 6: k fold cross-validation with k =5.

E. Model Evaluation

Having verified the essential training for the model, the last phase is the model evaluation. This is done by testing the applied models with separate testing dataset. This phase includes:

- **Final Training & Testing:** Once hyperparameters are fixed, the models are retrained on the entire (or a large portion of) training set (V2–V4). The separate test set in NSL-KDD is then used for final evaluation.

- **Metrics:** Standard performance metrics as shown below—accuracy, precision, recall, F1-score, and especially AUROC (which can be as high as 94.6% with simple baselines)—can be used to demonstrate the IDS's performance of the model.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$
$$Precision = \frac{TP}{TP+FP}$$
$$Recall = \frac{TP}{TP+FN}$$
$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

In this formular, TP is true positive, and TN denotes true negative, that is, the number of samples that are correctly classified as positive and negative, respectively. False positive, (FP) and false negative (FN) characterize the number of incorrectly classified samples as positive and negative, respectively. In addition, confusion matrices offer insights into misclassifications (e.g., false positives or false negatives).

- **Time Complexity & Practicality:** In addition to accuracy, the computational cost of training and predicting is measured. For instance, ANNs/DNNs can be slower to train than DT or NB but may yield quicker classification once trained.

4. Implementation

This section describes the detailed implementation of the above ML-based Intrusion Detection System (IDS) using the NSL-KDD dataset in a Jupyter Notebook environment. The overall framework encompasses dataset preprocessing, feature selection, model training, and performance evaluation using standard classification metrics.

4.1 Environment Setup and Dataset Preparation

The experiments are conducted in a Python environment leveraging Jupyter Notebook. Essential libraries such as *pandas*, *numpy*, *scikit-learn*, and *matplotlib* are used for data manipulation, model implementation, and visualization. The NSL-KDD dataset, an improved version of the KDD Cup-99 dataset, was chosen because it addresses data redundancy issues and presents a more realistic challenge for IDS evaluation.

Data preprocessing involved several steps:

- **Loading the Data:** The dataset was imported in CSV format using *pandas*.
- **Cleaning and Formatting:** Although the NSL-KDD dataset has no missing values, preprocessing included converting categorical variables into numeric form using one-hot encoding and normalizing continuous features.
- **Train-Test Split:** The dataset was randomly divided into training (70%) and testing (30%) subsets to ensure robust evaluation. Additionally, k-fold cross-validation (with k=10) was applied during hyperparameter tuning to mitigate overfitting.

4.2 Feature Identification and Selection

Given the high-dimensional nature of network traffic data, feature selection is crucial to enhance both the computational efficiency and the predictive performance of the models. Inspired by approaches in [10-11], a Random Forest classifier was initially used to rank features based on their Gini importance. The top-ranked features were retained for further analysis. This step not only reduced the feature space but also helped in removing redundant or noisy features that could adversely affect model performance.

4.3 Model Implementation in Python

The following ML algorithms were implemented to classify network traffic into benign and attack classes:

- **Decision Tree (DT) and Random Forest (RF):**

  The DT and RF classifiers were built using the *scikit-learn* library. While the DT provides an interpretable model structure, RF—being an ensemble method—helps in reducing variance by averaging multiple decision trees. Grid search with cross-validation was employed to tune hyperparameters such as maximum depth and the number of estimators.

- **Support Vector Machine (SVM):**

  An SVM with a Radial Basis Function (RBF) kernel was used due to its robustness in handling non-linear relationships in the data. Hyperparameter tuning included optimizing the regularization parameter (C) and the kernel coefficient (gamma).

- **Naïve Bayes (NB):**

  Both Gaussian and Multinomial NB variants were evaluated. Despite its simplicity, NB often serves as a strong baseline due to its fast-training times and ease of interpretation.

- **Artificial Neural Network (ANN) and Deep Neural Network (DNN):**

  For ANN implementation, a simple feed-forward neural network was built using Keras with TensorFlow as the backend. The network architecture consisted of one input layer, one hidden layer (with 10 neurons as a starting point), and one output layer. The DNN model extended this architecture by stacking multiple hidden layers to capture higher-level abstractions. Early stopping and dropout were integrated to prevent overfitting.

Each model was trained on the preprocessed NSL-KDD dataset with standard classification targets. The use of Jupyter Notebook allowed for interactive visualization of training curves, confusion matrices, and ROC curves, thereby facilitating iterative improvements.

4.4 Evaluation Metrics and Results Visualization

Model performance is primarily assessed using accuracy, precision, recall, and F1-score. Additionally, confusion matrices are generated to provide insights into the number of true positives, false negatives, false positives, and true negatives.

Visualizations, such as ROC curves and confusion matrices, are plotted using *matplotlib* to compare the performance of different classifiers side by side. These plots helped in identifying the trade-offs between computational efficiency and classification performance, guiding further model optimization.

4.5 Implementation Challenges and Future Directions

While the implementation in Python provided flexibility and ease of experimentation, several challenges are encountered. Feature selection required careful consideration since excessive removal could lead to loss of critical information, whereas insufficient pruning increased computational complexity. Moreover, the inherent class imbalance in the NSL-KDD dataset necessitated the use of resampling techniques and cost-sensitive learning to improve detection of rare attack types. Future work will explore advanced deep learning architectures, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), to further enhance model generalization on real-world network traffic. Additionally, the integration of automated hyperparameter tuning frameworks and ensemble methods will be considered to refine detection performance and reduce false alarm rates.

5. Experiment and Result Discussion

Now, this section examines the results of the algorithmic experiment of the machine learning models with the KDD-based security detection data set. The base dataset has about 125,972 records of attack/normal with about 43 features (distinct characteristics). Among this large dataset, 80% were used for training the model, while the remaining 20% were used for testing with 10-fold cross-validation. The counts of each attribute in the whole dataset is illustrated in Fig.7. There appears to be no missing value in the dataset. But we noticed that normal intrusion accounts for about 53.5% of the dataset, followed by Neptune attack type, while the most attack form is through the TCP prototype as shown in Fig.8. That is, most of the attacks comes from tcp protocols and least come from ICMP. Similarly, the most used service for these attack in general is through http, follow by private, domain_u , smtp,ftp and others as shown in Fig. 9.

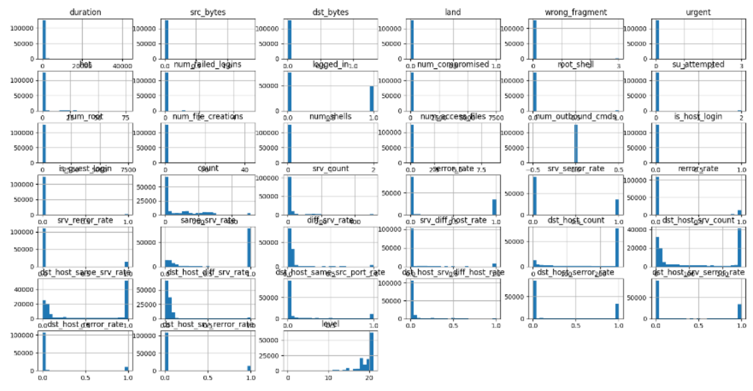
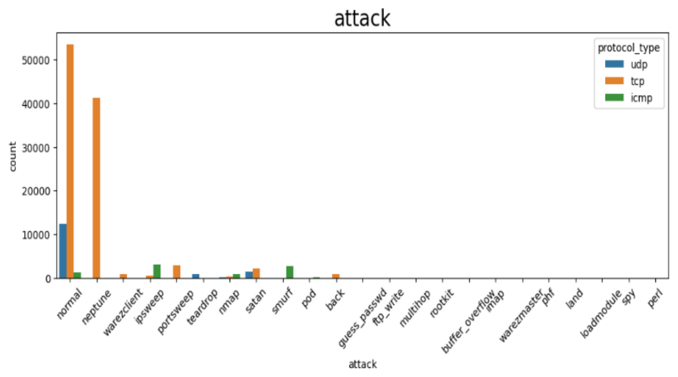
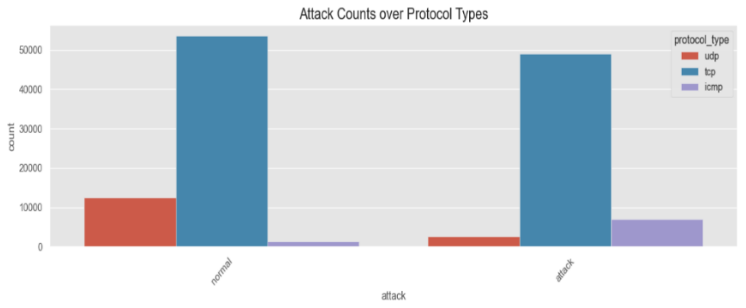


Fig. 7: Counts of features in the KDD dataset.





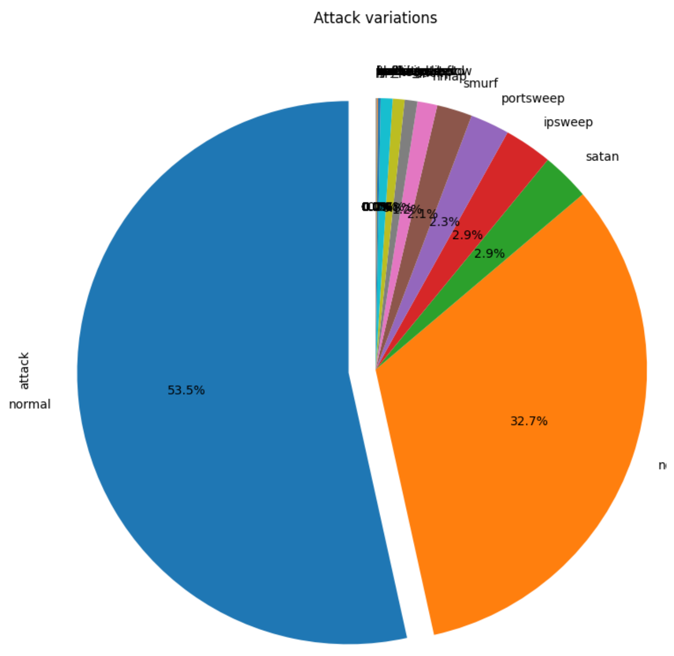


Fig. 8: Proportion of normal and various attacks type couple with prototype form.

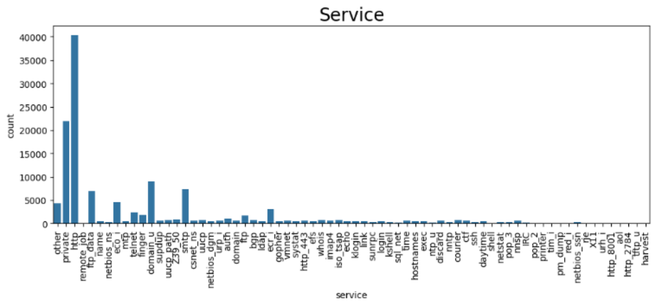
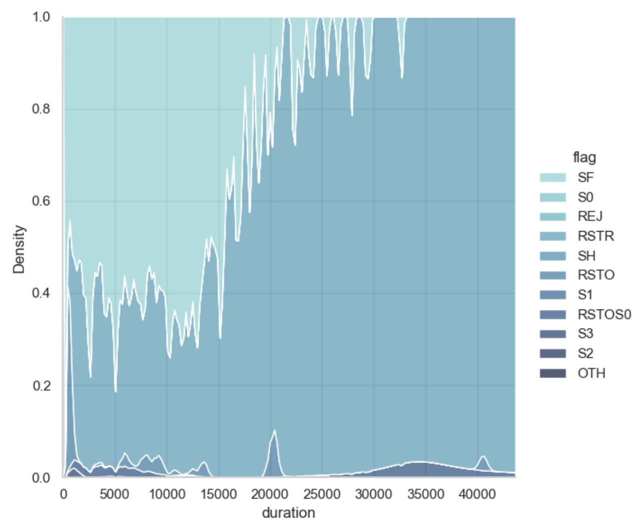
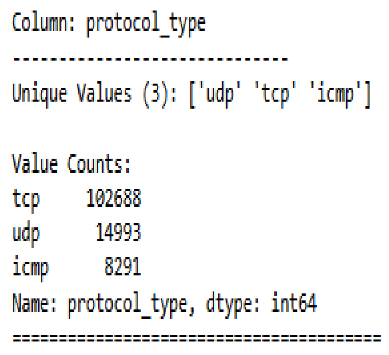


Fig. 9: Counts of attack through the various service mode.

.

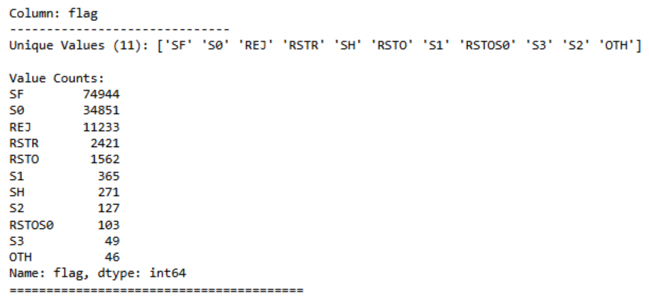


To avoid redundancy and ensure only unique values in the dataset are used for the analysis and simulation, I check for all possible unique value and notice their counts by main categories in the dataset. The result is presented in Fig. 10.



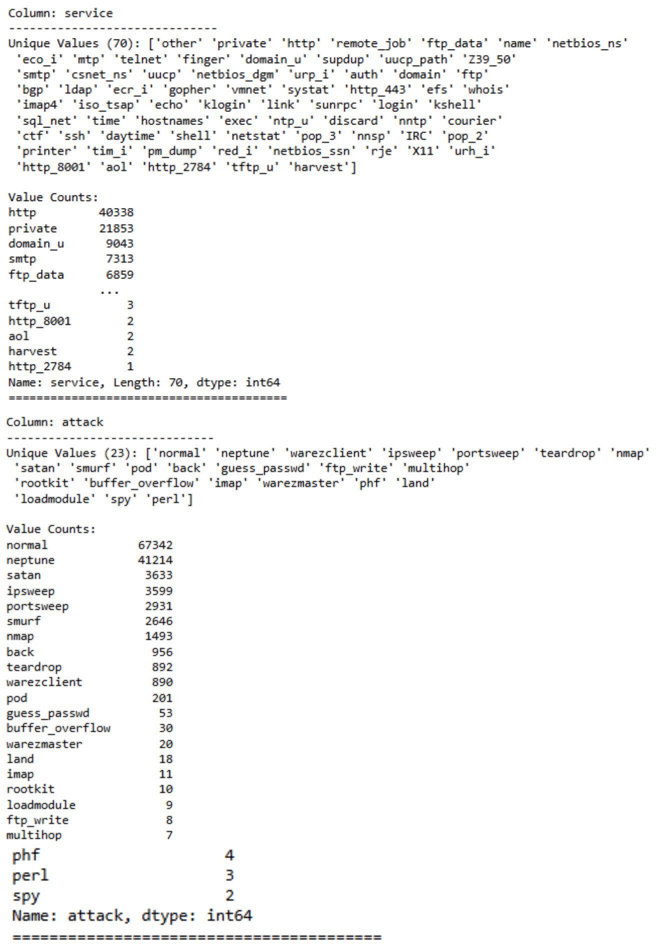
```
Column: protocol_type
------------------------------
Unique Values (3): ['udp' 'tcp' 'icmp']

Value Counts:
tcp     102688
udp      14993
icmp      8291
Name: protocol_type, dtype: int64
========================================
```



```
Column: flag
------------------------------
Unique Values (11): ['SF' 'S0' 'REJ' 'RSTR' 'SH' 'RSTO' 'S1' 'RSTOS0' 'S3' 'S2' 'OTH']

Value Counts:
SF        74944
S0        34851
REJ       11233
RSTR       2421
RSTO       1562
S1          365
SH          271
S2          127
RSTOS0      103
S3           49
OTH          46
Name: flag, dtype: int64
========================================
```



```
Column: service
------------------------------
Unique Values (70): ['other' 'private' 'http' 'remote_job' 'ftp_data' 'name' 'netbios_ns'
 'eco_i' 'mtp' 'telnet' 'finger' 'domain_u' 'supdup' 'uucp_path' 'Z39_50'
 'smtp' 'csnet_ns' 'uucp' 'netbios_dgm' 'urp_i' 'auth' 'domain' 'ftp'
 'bgp' 'ldap' 'ecr_i' 'gopher' 'vmnet' 'systat' 'http_443' 'efs' 'whois'
 'imap4' 'iso_tsap' 'echo' 'klogin' 'link' 'sunrpc' 'login' 'kshell'
 'sql_net' 'time' 'hostnames' 'exec' 'ntp_u' 'discard' 'nntp' 'courier'
 'ctf' 'ssh' 'daytime' 'shell' 'netstat' 'pop_3' 'nnsp' 'IRC' 'pop_2'
 'printer' 'tim_i' 'pm_dump' 'red_i' 'netbios_ssn' 'rje' 'X11' 'urh_i'
 'http_8001' 'aol' 'http_2784' 'tftp_u' 'harvest']

Value Counts:
http         40338
private      21853
domain_u      9043
smtp          7313
ftp_data      6859
             ...
tftp_u           3
http_8001        2
aol              2
harvest          2
http_2784        1
Name: service, Length: 70, dtype: int64
========================================

Column: attack
------------------------------
Unique Values (23): ['normal' 'neptune' 'warezclient' 'ipsweep' 'portsweep' 'teardrop' 'nmap'
 'satan' 'smurf' 'pod' 'back' 'guess_passwd' 'ftp_write' 'multihop'
 'rootkit' 'buffer_overflow' 'imap' 'warezmaster' 'phf' 'land'
 'loadmodule' 'spy' 'perl']

Value Counts:
normal             67342
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: attack, dtype: int64
========================================
```

Fig.10: Counts of unique value for prototype, flag, service and attack.

Subsequently, we also examine the level of correlation between dataset features to identify those who are better make for the model. From **Fig. 11**, my main finding is that comprehensively understanding the correlation between columns is critical for constructing a robust model, as the inclusion of less significant or redundant columns can introduce bias and potentially compromise model performance. By this analysis, I discovered that connection-related features often measured as traffic rates exhibit very high correlations, indicating that these variables share overlapping information. Similarly, host-based traffic features, such as destination-related metrics, also show strong inter-correlations. These insights suggest that retaining all correlated features may not contribute additional predictive value; instead, it could lead to multicollinearity, resulting in an overfitted or biased model. Consequently, applying techniques such as feature selection, dimensionality reduction, or even combining related features is essential to ensure the final model is both efficient and accurate.

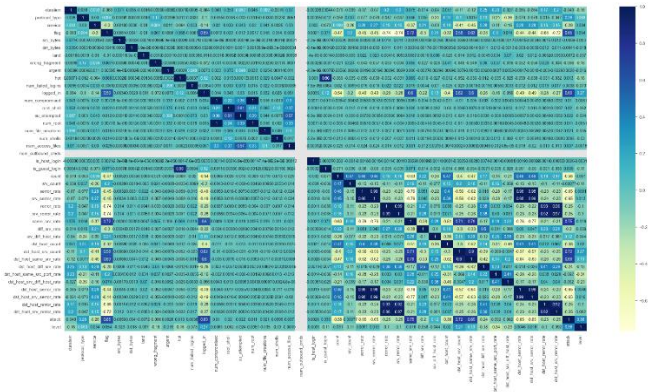


Fig. 11: Correlation plot of features in the dataset.

Next, I consider feature selection. The feature selection is an important process when training the models using the KDD dataset, as it includes about 43 features. I do not inspect the features manually as the dataset includes many features.

When performing feature selection using the filter-based feature algorithm, 80% of the dataset is used to train the model. The features are then ranked according to their respective importance scores calculated during the training process. The ranking of the features is displayed in Fig. 12. As shown in Fig. 12, the scores of the top two features are significantly higher than the rest. Other than that, most features have importance scores of 0.03 to 0.55. From the result given by the filter-based feature algorithm, the top 10 to 15 features are needed to train the models.

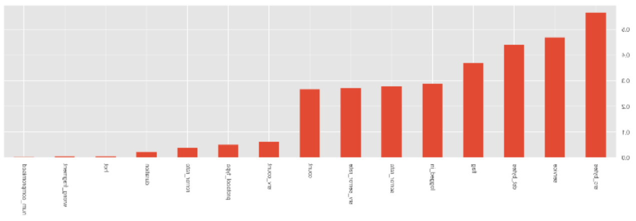
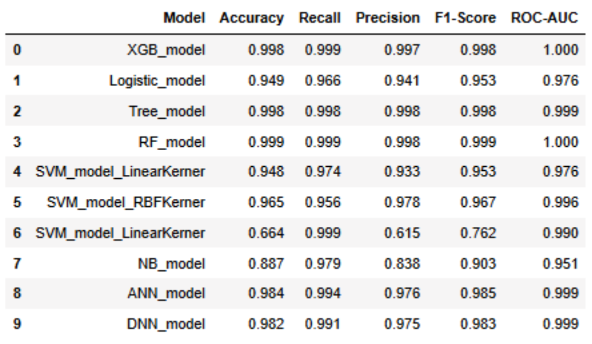


Fig. 12: Important features from the application of filter-based feature selection method.

The accuracy, F1-Score and other metrics of all the implemented ML algorithms is presented in **Table 1**. All machine learning computational methods utilized the conventional hyper-parameter variables specified by the Scikit-learn ML binary and tested again using the training dataset. Besides that, the accuracies of the models are also visualized using the confusion matrixes as shown in **Fig. 13**. The table and the confusion matrices indicated that RF and XGB models have best accuracy followed by Tree Decision, ANN and DNN models, while Naye Bayes Model is biased towards the normal class.

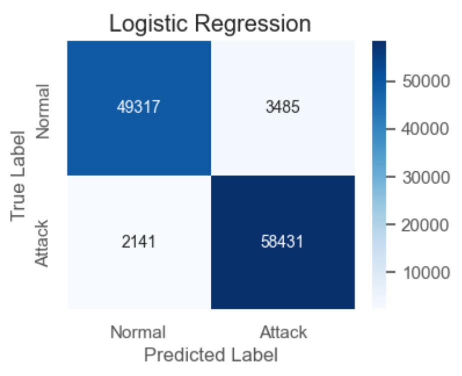


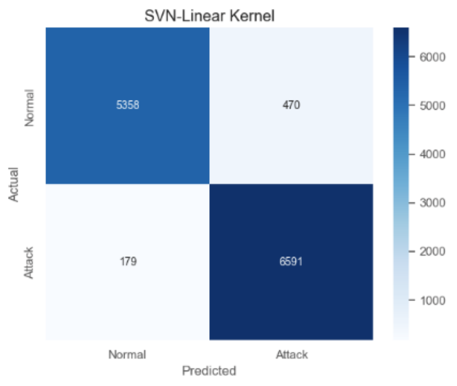
	Model	Accuracy	Recall	Precision	F1-Score	ROC-AUC
0	XGB_model	0.998	0.999	0.997	0.998	1.000
1	Logistic_model	0.949	0.966	0.941	0.953	0.976
2	Tree_model	0.998	0.998	0.998	0.998	0.999
3	RF_model	0.999	0.999	0.998	0.999	1.000
4	SVM_model_LinearKerner	0.948	0.974	0.933	0.953	0.976
5	SVM_model_RBFKerner	0.965	0.956	0.978	0.967	0.996
6	SVM_model_LinearKerner	0.664	0.999	0.615	0.762	0.990
7	NB_model	0.887	0.979	0.838	0.903	0.951
8	ANN_model	0.984	0.994	0.976	0.985	0.999
9	DNN_model	0.982	0.991	0.975	0.983	0.999

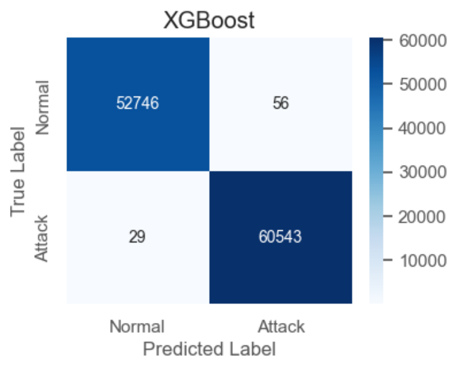
Table 1: Performance of ML algorithms for KDD—based Network IDS dataset.

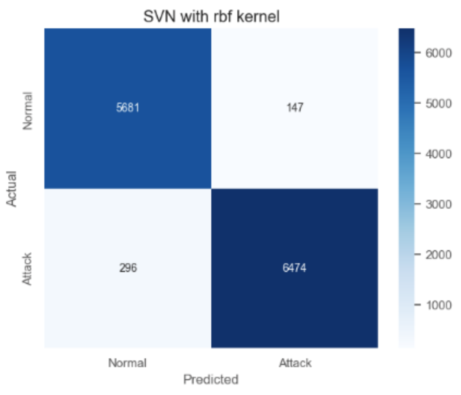
The first notable insight from these results is that several models achieve remarkably high accuracy on the KDD dataset, with many models surpassing 99% accuracy and some, like the XGBoost (XGB), Artificial Neural Network (ANN), and Deep Neural Network (DNN) models, nearly reaching a perfect 100%. This indicates that these algorithms are highly effective at correctly identifying the vast majority of both normal and malicious traffic. The Decision Tree and Naive Bayes classifiers also exhibit strong performance with accuracy in the 99% range, although they fall slightly short of the top performers. In contrast, some SVM-based models configured with an SVM linear kernel display lower accuracy scores, suggesting either a lack of nuanced decision boundaries compared to tree-based or deep learning approaches, or potentially suboptimal hyperparameter settings for this particular dataset.

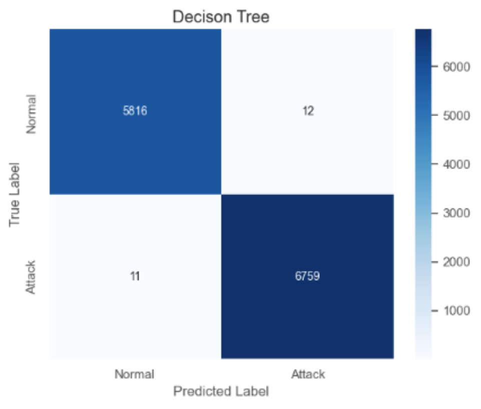
When focusing on the F1 scores, a metric that balances both precision and recall, it is clear that these models not only classify the majority of samples correctly (high accuracy) but also do so with an excellent balance of low false positives and low false negatives. Models like XGB, Random Forest, SVM with RBF, and neural networks (ANN, DNN) exhibit near-ideal F1 scores, indicating that they can detect attacks reliably while rarely misclassifying benign traffic. This combination of high accuracy and strong F1 scores is particularly crucial in intrusion detection scenarios, where missing malicious threats (low recall) or inundating security teams with false alarms (low precision) can be equally problematic. Overall, these results demonstrate that carefully tuned tree-based models, ensemble methods, and deep learning approaches are particularly well-suited for the KDD intrusion detection dataset, providing both consistent and robust performance.

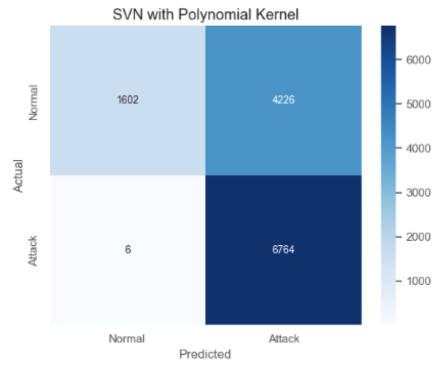

Logistic Regression
True Label
Normal
Attack
49317
3485
2141
58431
50000
40000
30000
20000
10000
Normal
Attack
Predicted Label

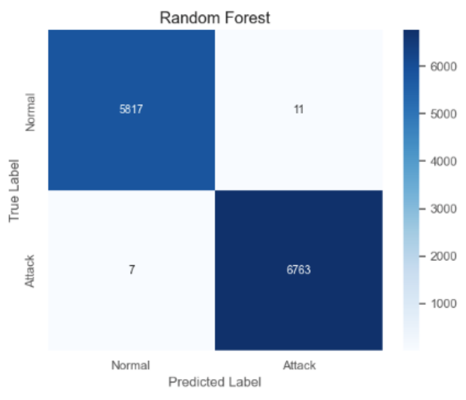

SVN-Linear Kernel
Actual
Normal
Attack
5358
470
179
6591
6000
5000
4000
3000
2000
1000
Normal
Attack
Predicted

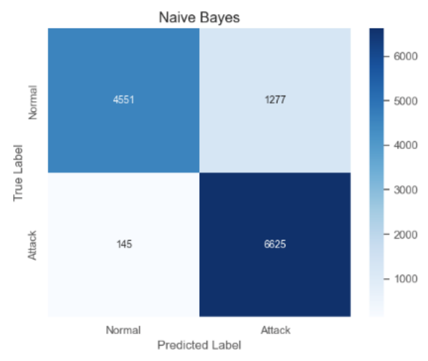

XGBoost
True Label
Normal
Attack
52746
56
29
60543
60000
50000
40000
30000
20000
10000
Normal
Attack
Predicted Label


SVN with rbf kernel
Actual
Normal
Attack
5681
147
296
6474
6000
5000
4000
3000
2000
1000
Normal
Attack
Predicted


Decison Tree
True Label
Normal
Attack
5816
12
11
6759
6000
5000
4000
3000
2000
1000
Normal
Attack
Predicted Label


SVN with Polynomial Kernel
Actual
Normal
Attack
1602
4226
6
6764
6000
5000
4000
3000
2000
1000
Normal
Attack
Predicted


Random Forest
True Label
Normal
Attack
5817
11
7
6763
6000
5000
4000
3000
2000
1000
Normal
Attack
Predicted Label


Naive Bayes
True Label
Normal
Attack
4551
1277
145
6625
6000
5000
4000
3000
2000
1000
Normal
Attack
Predicted Label

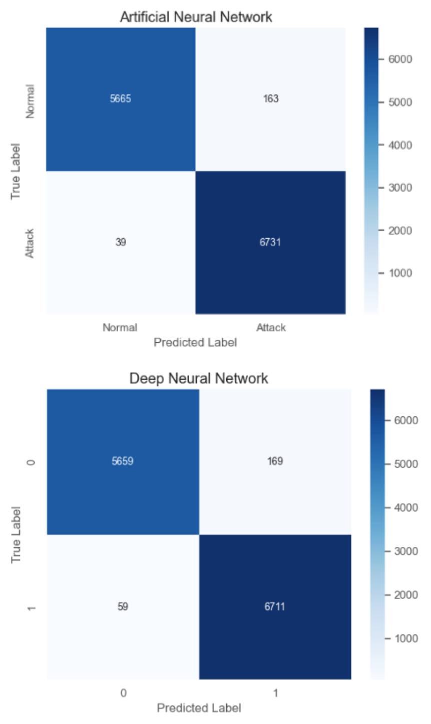


Fig. 13: Binary calculated confusion matrices by all the ML models.

6. Conclusion

This report presents a brief overview, implementation, and comparative analysis of state-of-the-art machine learning methods for intrusion detection. By evaluating whether an attack has occurred based on characteristics such as source IP, destination IP, packet length, and other network parameters, we have demonstrated that supervised learning methods significantly outperform the unsupervised approach employed in our study. Our investigation clearly shows that a variety of machine learning techniques can be effectively applied to intrusion detection tasks. The results highlight the substantial positive impact of these techniques on improving overall system performance, notably in terms of achieving high accuracy and reducing false negatives. In particular, the Random Forest Model (RFM) emerged as the most promising algorithm, delivering near-perfect results across critical metrics including accuracy, recall, F1-score, and confusion matrix assessments.

Looking ahead, further enhancements in intrusion detection systems appear promising through the integration of deep learning methodologies and more advanced feature selection strategies. The careful selection of features is crucial, as redundant or less informative variables can lead to overfitting and diminished model performance. Future work will therefore focus on exploring additional deep learning approaches that could offer more robust generalization and scalability, as well as refining the feature engineering process to minimize noise and multicollinearity. By addressing these aspects, we anticipate that subsequent models will not only provide even higher detection accuracy but also further reduce the rate of false alarms, ultimately contributing to more secure and reliable network defense mechanisms.

References:

[1] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, K.-Y. Tung, Intrusion detection system: A comprehensive review, Journal of Network and Computer Applications 36 (1) (2013) 16–24. doi:doi:10.1016/j. jnca.2012.09.004.

[2] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: Techniques, systems and challenges, Computers & Security 28 (1-2) (2009) 18–28. doi:https://doi.org/10.1016/j.cose.2008.08.003.

[3] A. Khraisat, I. Gondal, P. Vamplew, An anomaly intrusion detection system using C5 decision tree classifier, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2018, pp. 149–155. doi:10.1007/978-3-030-04503-6_14.

[4] C. Kreibich, J. Crowcroft, Honeycomb: creating intrusion detection signatures using honeypots, ACM SIGCOMM computer communication review 34 (1) (2004) 51–56. doi:10.1145/972374.972384.

[5] Kaspersky, What is a zero-day attack? - definition and explanation, https://www.kaspersky.com/ resource-center/definitions/zero-day-exploit.

[6] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, O'Reilly Media, Inc, 2019.

[7] H. Hindy, R. Atkinson, C. Tachtatzis, J.-N. Colin, E. Bayne, X. Bellekens, Utilising deep learning techniques for effective zero-day attack detection, Electronics 9 (10) (2020) 1684. doi:10.3390/ electronics9101684.

[8] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," Ieee communications surveys & tutorials, vol. 16, no. 1, pp. 303–336, 2013.

[9] G. G. Liu, "Intrusion detection systems," in Applied Mechanics and Materials, vol. 596. Trans Tech Publ, 2014, pp. 852–855.

[10] C. Chio and D. Freeman, Machine Learning and Security: Protecting Systems with Data and Algorithms. O'Reilly Media, Inc., 2018.

[11] M. Rege and R. B. K. Mbah, "Machine learning for cyber defense and attack," DATA ANALYTICS 2018, p. 83, 2018.

[12] K. Wiggers, Gmail is now blocking 100 million more spam emails a day, thanks to TensorFlow, 2019.

[13]. R. K. Vigneswaran, R. Vinayakumar, K. P. Soman and P. Poornachandran, "Evaluating Shallow and Deep Neural Networks for Network Intrusion Detection Systems in Cyber Security," 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bengaluru, India, 2018, pp. 1-6, doi: 10.1109/ICCCNT.2018.8494096.

[14]. G. Karatas, O. Demir and O. K. Sahingoz, "Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset," in IEEE Access, vol. 8, pp. 32150-32162, 2020, doi: 10.1109/ACCESS.2020.2973219.

[15] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," *Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2009.

[15a] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," ACM Transactions on Information and System Security, vol. 3, no. 4, pp. 262–294, 2000.

[16] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, O'Reilly Media, Inc, 2019.

[17] S. R. Safavian, D. Landgrebe, A survey of decision tree classifier methodology, IEEE Transactions

on Systems, Man, and Cybernetics 21 (3) (1991) 660–674. doi:10.1109/21.97458.

[18] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.

[19] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297. doi:

10.1007/BF00994018.

[20] D. D. Lewis, Naive (bayes) at forty: The independence assumption in information retrieval, in: European conference on machine learning, Vol. 1398, Springer, 1998, pp. 4–15. doi:10.1007/BFb0026666.

[21] W. Liu, Z.Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26. doi:10.1016/j.neucom.2016.12.038.

[22] E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation

18 (7) (2006) 1527–1554. doi:10.1162/neco.2006.18.7.1527.

[23] Y. Bengio, Learning deep architectures for AI, Now Publishers Inc, 2009. doi:10.1561/2200000006.

[24] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science

313 (5786) (2006) 504–507. doi:10.1126/science.1127647