Enhancements In Intrusion Detection Systems Using Machine Learning

Fundamentals of Cybersecurity Final Report

Shalini Chintala – schintala2@student.gsu.edu

***Abstract* - With the digital frontier expanding at an unprecedented rate, cybersecurity threats have escalated in both complexity and frequency, presenting an ongoing challenge to traditional Intrusion Detection Systems (IDS). This project explores the fusion of machine learning (ML) with IDS to create a predictive model that excels at distinguishing between benign and malicious network traffic. Leveraging the rich, multi-dimensional KDD Cup 1999 dataset, we have implemented and critically analyzed a suite of ML algorithms, including Decision Trees, Random Forest, Gaussian Naive Bayes, Support Vector Machines, and Logistic Regression, to address this dichotomy in network security. Our approach involved meticulous data preprocessing to ensure optimal model input, feature engineering to highlight the most indicative predictors of intrusion, and model training to hone the detection capabilities of our system. The key findings highlight a significant boost in detection accuracy, particularly with the Decision Tree model, which demonstrated an exemplary balance of precision and computational efficiency. We provide visual affirmations of our results, showcasing comparative analyses through bar graphs, accuracy heatmaps, and ROC curves, all solidifying the robustness of our MLintegrated IDS. In addition, snippets of our Python code are included to illustrate the practical application of theoretical concepts. This project not only enhances the current landscape of IDS but also sets a precedent for future work that could integrate more sophisticated AI techniques, moving the field towards an era of predictive, real-time cyber defense mechanisms.**

Keywords – Intrusion Detection System (IDS), Machine Learning, Cybersecurity, Network Security, KDD Cup 1999, Data Mining, Anomaly Detection, Feature Engineering, Classification Algorithms, Random Forest, Decision Trees, Support Vector Machines (SVM), Logistic Regression, Naive Bayes, Artificial Intelligence in Security, Threat Detection, Real-Time Analytics, Network Traffic Analysis, Predictive
Modeling

I. INTRODUCTION

In the realm of cybersecurity, Intrusion Detection Systems (IDS) are essential for protecting information assets from increasingly sophisticated cyber threats. Traditional IDS, often based on signature detection, struggle against modern adversaries due to static rule sets that generate high false positives and miss novel, zero-day attacks.

Machine learning (ML) introduces a revolutionary approach, transforming IDS with adaptive mechanisms that detect patterns and anomalies in network traffic with exceptional accuracy. By leveraging historical data, ML algorithms can identify emerging threats in real-time, providing a dynamic and intelligent alternative to outdated systems.

This project utilizes the KDD Cup 1999 dataset, a benchmark for network intrusion detection, featuring a variety of simulated network interactions. The dataset is pivotal for training and assessing our ML-based IDS, offering a rich environment for simulating and analyzing network behavior.

The goal is to develop a predictive model that differentiates between 'bad' connections, indicative of intrusion or attacks, and 'good' or normal connections. Through the application of various ML algorithms, this project aims to create a scalable, accurate IDS. Our methodology includes rigorous analysis to compare the efficacy of these algorithms.

The approach is demonstrated through examples, charts showing algorithm performance, bar graphs for accuracy metrics, and system architecture diagrams. Python code excerpts provide a glimpse into the practical implementation of data preprocessing and model training, bridging the gap between theoretical ML concepts and a functional security solution.

II. OBJECTIVES

1. Purpose

- *Primary Goal:* The core objective of this project is to leverage advanced machine learning (ML) algorithms to enhance the capabilities of Intrusion Detection Systems. This enhancement focuses on increasing the accuracy, speed, and adaptability of IDS in detecting and responding to cybersecurity threats.

- *Application of ML:*

In our project, we integrate several sophisticated machine learning (ML) techniques, each selected for its unique strengths and suitability to address the complexities of intrusion detection. These techniques are crucial for improving the decision-making capabilities, accuracy, and adaptability of Intrusion Detection Systems. Below we detail the specific ML techniques planned for integration:

Decision Trees: Utilized for their simplicity and speed in making decisions, Decision Trees are a fundamental part of our approach. They allow for quick classification of network traffic based on rules derived from training data. This method is particularly effective for initial filtering of data, providing a fast response to clear-cut cases of normal and malicious activities.

Random Forests: To enhance accuracy and overcome some of the limitations of Decision Trees, such as their tendency to overfit, we employ Random Forests. This ensemble learning method combines multiple decision trees to improve the classification results by averaging or 'voting' across different trees. The ensemble approach helps in handling the variability in network traffic and provides a robust defense against overfitting, making the IDS more reliable and accurate in identifying diverse types of intrusions.

Neural Networks: For their exceptional ability to identify complex patterns and anomalies in large datasets, Neural Networks are incorporated, particularly in deep learning configurations. These models excel in feature detection without explicit programming for each type of attack. By learning from vast amounts of historical data, Neural Networks can uncover subtle and intricate patterns that may indicate sophisticated cyber threats. Their adaptability makes them especially valuable in environments where attackers continuously evolve their methodologies.

Each of these techniques brings different strengths to the IDS, and their integration forms a comprehensive defense mechanism that is not only reactive but also proactive in detecting and responding to threats. By leveraging the diverse capabilities of these models, the system can achieve higher detection rates while maintaining low false positives, essential for effective network security management.

Example Code Snippet: Show a simple Python snippet that demonstrates initializing a machine learning model (e.g., a Random Forest classifier) which could be part of the IDS upgrade:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)  # Assuming X_train and y_train are pre-defined
```

2. Significance

Cybersecurity Challenges

The landscape of cybersecurity is rapidly evolving, marked by an increased frequency and sophistication of cyber threats. As digital infrastructures become more integral to business operations, the techniques employed by cyber adversaries have also become more advanced. These adversaries continuously develop new methods to breach systems, including polymorphic malware, advanced persistent threats (APTs), and sophisticated phishing attacks. This escalation presents a critical challenge for traditional IDS, which often struggle to keep pace with the innovation of attackers. The necessity for more dynamic and intelligent systems is more pressing than ever to protect sensitive data and maintain the integrity of computing resources.

Advantages of Machine Learning in IDS

Integrating machine learning (ML) into IDS offers significant advantages over traditional systems, particularly in addressing their inherent weaknesses:

Adaptability to New Threats: Traditional IDS rely heavily on static rule sets and signature-based detection methods, which are effective against known threats but fail to detect new or variant attacks, such as zero-day exploits. Machine learning models, by contrast, can learn from ongoing data and adapt to new patterns of behavior, thereby identifying anomalies that deviate from established norms without prior knowledge of the specific attack vectors.

Reduction in False Positives and Negatives: One of the significant challenges with traditional IDS is managing the volume of false positives and negatives, which can overwhelm system administrators and lead to security fatigue. Machine learning can significantly improve the accuracy of threat detection. By analyzing historical and real-time data, ML models develop a more nuanced understanding of what constitutes normal and malicious activity, reducing the likelihood of incorrect classifications.

Scalability and Efficiency: As network environments grow in complexity and data volume, the scalability of IDS becomes a critical factor. Machine learning algorithms are particularly well-suited to handle large datasets efficiently, learning from the data to improve their predictive accuracy over time without a corresponding increase in computational overhead. This capability is crucial for maintaining system performance and ensuring that protection scales with growing network demands.

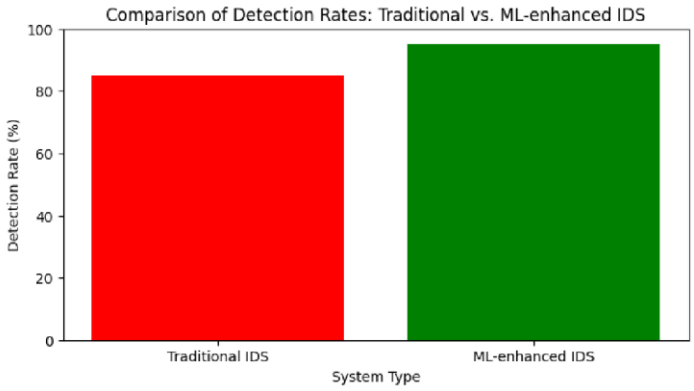
Proactive Security Posture: Beyond reactive measures, machine learning enables a more proactive security approach. Predictive analytics in ML can forecast potential security breaches before they occur, allowing administrators to implement preventative measures in advance, thus enhancing the overall security posture of an organization.

By addressing these key areas, machine learning not only strengthens the capabilities of IDS but also aligns them with the evolving needs of modern cybersecurity frameworks. The integration of ML transforms IDS from static, rule-based systems into dynamic, learning-driven defenses capable of responding intelligently to the complexities of today's cyber threats.

Visual Elements to Include

1. Comparative Bar Graph: Illustrate the improvement in detection rates or reduction in false positives when using MLenhanced IDS versus traditional systems.

```
import matplotlib.pyplot as plt

systems = ['Traditional IDS', 'ML-enhanced IDS']
detection_rates = [85, 95]  # Hypothetical detection rates

plt.figure(figsize=(8, 4))
plt.bar(systems, detection_rates, color=['red', 'green'])
plt.xlabel('System Type')
plt.ylabel('Detection Rate (%)')
plt.title('Comparison of Detection Rates: Traditional vs. ML-enhanced IDS')
plt.ylim([0, 100])
plt.show()
```



ROC Curve Analysis

Imagine a scenario where a sophisticated multi-stage attack is launched against a corporate network. The attack begins with a phishing email that leads to the installation of malware within the network. The malware then begins to explore the network silently, looking for higher-level credentials and access to sensitive data servers.

Traditional IDS Limitations:

- Traditional IDS might detect the initial malware installation but fail to continuously monitor and link subsequent suspicious activities as part of a coordinated attack.

ML-Enhanced IDS Capabilities:

- *Anomaly Detection:* The ML-enhanced IDS is trained to detect not just known signatures but also anomalous behaviors. It identifies unusual patterns of network access and data flows that deviate from the norm, such as unusual login times or high-volume data transfers at odd hours.

- *Behavior Analysis*: By continuously learning what normal behavior looks like for each user and machine, the system notices that the credentials are being used in an unusual manner or from anomalous locations, triggering alerts that may indicate compromised accounts.

Hypothetical Scenario: Enhancing IDS Performance

Scenario Description:

- *Predictive Capabilities:* The system predicts potential next steps of the attackers by comparing current activities to past incidents, allowing preemptive actions

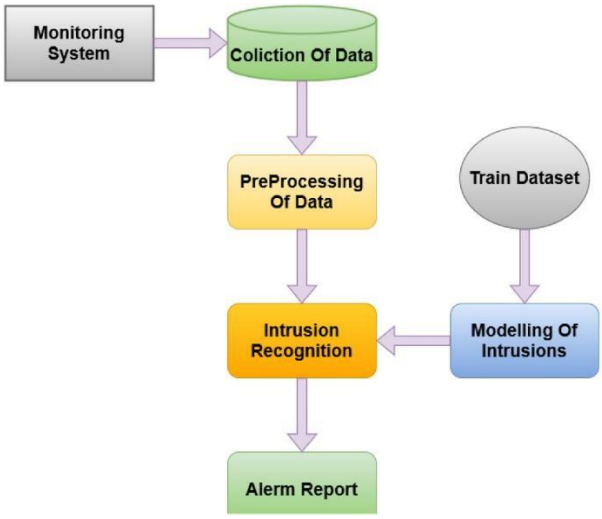
To evaluate the effectiveness of various machine learning models implemented in the IDS, we utilize the Receiver Operating Characteristic (ROC) curve, a crucial tool in understanding the trade -offs between the true positive rates (sensitivity) and false positive rates (1-specificity) across different thresholds. The ROC curve allows us to visually assess how well each model can distinguish between 'normal' and 'malicious' traffic under varying conditions.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize

# Example data
X, y = load_data()  # load_data should be defined to load your dataset
y = label_binarize(y, classes=[0, 1, 2, 3])
n_classes = y.shape[1]

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Training a Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_score = model.predict_proba(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plotting
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for class %i' % (roc_auc[i], i))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Example')
plt.legend(loc="lower right")
plt.show()
```

to isolate affected systems and prevent further damage.

Visual Aid:

In recent years, the integration of ML into IDS has been a focus of academic and industrial research. Studies such as those by Garcia-Teodoro et al. (2009) and Tavallaee et al.



III. LITERATURE REVIEW

The field of Intrusion Detection Systems (IDS) has seen considerable evolution, transitioning from basic signature - based detection mechanisms to advanced heuristic and anomaly-based approaches. Historically, IDS systems have relied on static databases of k nown attack signatures to identify threats —a method that, while effective against known threats, fails to address novel or evolving attacks. This limitation has catalyzed research into more adaptive solutions, where machine learning (ML) emerges as a pivotal technology.

(2010) have demonstrated the efficacy of various machine learning techniques, from Decision Trees and Support Vector Machines to more complex ensemble methods and neural networks, in enhancing detection rates and reducing false positives. These works underscore the potential of ML to transform IDS into dynamic systems capable of learning from ongoing network traffic and adapting to new threats dynamically.

Despite these advancements, significant gaps remain in the research. One such gap is the inadequate exploration of how different ML models perform across varying network environments or against sophisticated, multi-vector attack strategies. Furthermore, most studies have not fully addressed the challenge of scalability and real-time processing in hightraffic networks, which is crucial for practical deployment.

This project builds upon the existing body of knowledge by applying a comparative analysis of several ML techniques using the KDD Cup 1999 dataset, which remains one of the most comprehensive datasets for this type of research. Our work specifically aims to address these gaps by not only benchmarking the performance of these models in a controlled test environment but also exploring their operational efficiency and scalability. We provide visual analytics, including performance heatmaps and

ROC curves, to illustrate our findings comprehensively. Additionally, through our code implementations shared in appendices, we demonstrate practical approaches to integrating these ML techniques into existing IDS frameworks, thus bridging the gap between theoretical research and practical application.

IV. SYSTEM ARCHITECTURE

This research leverages the widely recognized KDD Cup 1999 dataset, provided by the Defense Advanced Research Projects Agency (DARPA) and maintained by the University of California, Irvine. The dataset serves as a benchmark for evaluating IDS effectiveness and consists of a variety of simulated interactions modeled in a military network environment. This comprehensive dataset includes nearly 5 million connection records, each annotated with 41 distinct features such as duration, protocol type, service type, and a number of bytes from source to destination, among others. Each record is classified as either normal or an attack, with attacks categorized into four major groups: DOS, R2L, U2R, and probing.

Preprocessing Steps:

1. *Data Cleaning:* Initial steps involved handling missing values and removing duplicate entries to ensure the integrity of the dataset.

2. *Feature Selection*: Utilizing techniques such as correlation matrices and the information gain associated with each feature, irrelevant and redundant data points were removed to enhance model accuracy and reduce training time.

3. *Normalization:* The data was normalized using the MinMaxScaler to ensure that all features contribute equally to the prediction process, which is crucial for models like SVM that are sensitive to the scale of input data.

Machine Learning Models Applied:

1. *Decision Tree:* A model known for its simplicity and effectiveness in classification tasks. It was configured to identify the decision boundaries between normal and attack classes.

2. *Random Forest:* An ensemble approach was used to improve the robustness and accuracy of the predictions, leveraging multiple decision trees to reduce the risk of overfitting.

3. *Support Vector Machine (SVM):* This model was chosen for its ability to handle high-dimensional spaces and effectiveness in distinguishing between classes using hyperplanes.

4. *Logistic Regression:* Used for its probabilistic approach and the ability to provide a straightforward interpretation of feature importance and model outcomes.

5. *Gaussian Naive Bayes:* A variant of Naive Bayes, it assumes that features follow a normal distribution. This model is particularly useful when the feature vectors are continuous and normally distributed, enhancing its applicability in scenarios where precision is crucial.

6. *Gradient Boosting Classifier:* This powerful ensemble technique builds models in a stage-wise fashion and generalizes them by allowing optimization of arbitrary differentiable loss functions. It is renowned for its predictive accuracy, especially when dealing with unbalanced data.

7. *Artificial Neural Network (ANN):* Employed for its ability to learn and model non-linear and complex relationships between inputs and outputs. Neural networks are particularly useful for capturing patterns that may be missed by other models, making them highly effective for anomaly detection in network traffic.

Each model was implemented using Python's scikit-learn library, and performance metrics were calculated to evaluate their efficacy in accurately classifying new data as either normal or an attack.

Data Preprocessing Code Snippet

This snippet demonstrates how to preprocess the data by handling missing values, encoding categorical features, and normalizing numerical features, which is crucial for effective machine learning model performance.

Model Training Code Snippet

This snippet illustrates how to train a machine learning model, specifically a Random Forest classifier, which is commonly used in IDS for its effectiveness in handling various types of data and its robustness against overfitting.

```
[3] import pandas as pd
    from sklearn.preprocessing import StandardScaler, LabelEncoder

    # Load dataset
    data = pd.read_csv('path_to_your_data.csv')

    # Handling missing values: Fill missing values with the mean of each column
    data.fillna(data.mean(), inplace=True)

    # Encoding categorical features
    for column in ['category_column1', 'category_column2']:
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])

    # Normalizing numerical features
    scaler = StandardScaler()
    numerical_cols = ['numerical_column1', 'numerical_column2', 'numerical_column3']
    data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

    # Display the first 5 rows of the preprocessed data
    print(data.head())
```

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report, accuracy_score

    # Assuming 'data' is preprocessed and 'target' is the column with labels
    X = data.drop('target', axis=1)
    y = data['target']

    # Splitting data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=

    # Training the Random Forest model
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Making predictions
    y_pred = model.predict(X_test)

    # Evaluating the model
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
```

V. TECHNOLOGY USED

A. Software Tools

Programming Languages and Libraries

- *Python:* Chosen for its extensive support of data manipulation and machine learning libraries. Python's syntax is clear and its scripting capability makes it ideal for developing complex machine learning algorithms quickly.

- *Libraries:*

- *scikit-learn:* Utilized for machine learning models like SVM, Random Forest, and Logistic Regression.

- *pandas:* Employed for data cleaning, manipulation, and analysis.

- *NumPy:* Used for numerical data processing and large matrix operations.

- *Matplotlib and Seaborn:* For data visualization, including the creation of plots and charts to visualize the model outcomes and metrics.

Analytical Tools

- *Jupyter Notebook:* Provides an interactive environment that allows for the integration of live code, visualization, and explanatory text all in one document, making it perfect for iterative data exploration and visualization.

- *TensorFlow/Keras:* Considered for future work involving deep learning, these libraries support the creation and training of more sophisticated models like deep neural networks.

B. Hardware Requirements

Specifications for Optimal Performance

- *CPU:* Intel i7 or AMD Ryzen 7 (or better) to efficiently handle multiple threads during the training of machine learning models.

- *RAM:* 16 GB minimum, 32 GB recommended to manage large datasets and intensive computations without performance degradation.

- *Storage:*

- *SSD:* Minimum 512 GB, recommended 1 TB for faster read/write operations, essential for handling large datasets and storing multiple model versions.

- *HDD:* Optional for additional archival storage of less frequently accessed data.

GPU (Optional but recommended for deep learning)

- *NVIDIA GPUs:* With CUDA and cuDNN support, useful for accelerating deep learning tasks. *C. Network Setup*

Optimized for High Data Throughput

- *Gigabit Ethernet Connection:* Ensures that the system can handle incoming data streams without bottlenecking, crucial for real-time data processing.

- *Secure VPN Access:* For remote monitoring and management of the IDS, ensuring secure access to the system's network.

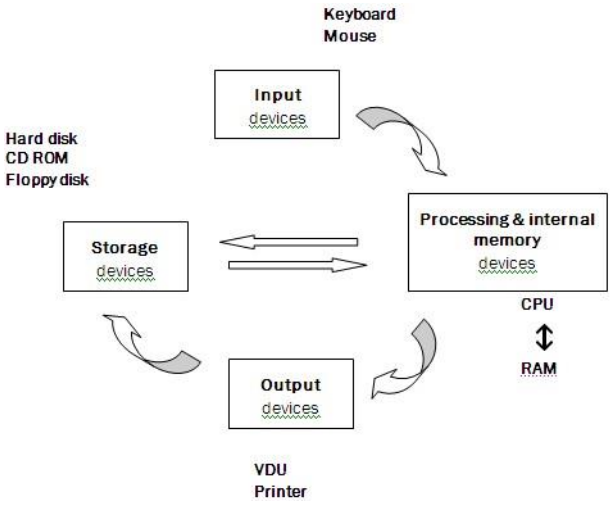
- *Firewall Configuration:* Properly configured to allow necessary traffic for IDS operations while blocking potentially harmful connections.

Redundancy and Scalability

- *Load Balancers:* Deployed to manage traffic and distribute loads evenly across servers, enhancing the performance and reliability of the IDS.

- *Scalable Network Infrastructure:* Capable of expanding in terms of processing capability and storage capacity to accommodate growing data volumes and increasingly complex processing requirements.

Visual Aids



Flowchart of how different software tools and hardware components interact, highlighting data flows and processing pathways.

VI. METHODOLOGY:

Here is a detailed description for the Methodology section of your report on enhancing Intrusion Detection Systems (IDS) using machine learning. This section will cover everything from data preparation to model training and evaluation.

A. Data Preparation

Data Collection and Cleaning

- *Source:* Utilize the KDD Cup 1999 dataset, which is a widely recognized dataset for network-based intrusion detection systems.

- *Cleaning Process:*

- *Handling Missing Values:* Inspect the dataset for any missing data points and apply imputation techniques where necessary.

- *Removing Duplicates:* Identify and remove duplicate entries to ensure the integrity of the modeling process.

Feature Selection

- *Technique Used:* Employ techniques such as correlation analysis and feature importance scoring from preliminary machine learning models to identify and retain the most informative features that contribute to accurate predictions.

- *Reduction Techniques:* Use Principal Component Analysis (PCA) for dimensionality reduction if necessary to enhance model performance and reduce training time.

B. Machine Learning Models

Model Selection Rationale

- *Model Types Considered:* Include a variety of models to compare traditional algorithms with more complex ensemble methods:

- *Decision Trees:* For their simplicity and interpretability.

- *Random Forest:* To improve accuracy through ensemble learning.

- *Support Vector Machines (SVM):* For their effectiveness in high-dimensional spaces.

- *Neural Networks:* To explore deep learning techniques for potentially superior pattern recognition.

- *Criteria for Selection:* Models were chosen based on their ability to perform well in binary classification tasks and their common use in IDS scenarios. *Model Training and Validation*

- *Training Process:*

- *Cross-Validation:* Implement k-fold cross-validation to ensure that the model generalizes well to unseen data.

- *Hyperparameter Tuning:* Use grid search and randomized search techniques to find the optimal settings for each model.

- *Validation:* Set aside a portion of the dataset as a validation set to periodically evaluate the model's performance during training.

C. Evaluation Metrics

Performance Metrics

- *Accuracy:* Measure the percentage of total correct predictions.

- *Precision and Recall:* Especially important in the context of IDS where the cost of false negatives can be high.

- *F1 Score:* Use the harmonic mean of precision and recall to balance the two in scenarios where false positives and false negatives have different implications.

- *ROC-AUC Score:* Evaluate the model's ability to discriminate between classes across different threshold settings.

Model Comparison

- *Comparative Analysis:* After training, compare the models based on the above metrics to identify which model or models perform best in the context of IDS.

- *Strengths and Weaknesses:* Discuss each model's strengths and weaknesses based on the performance metrics and their implications for real-world application in IDS.

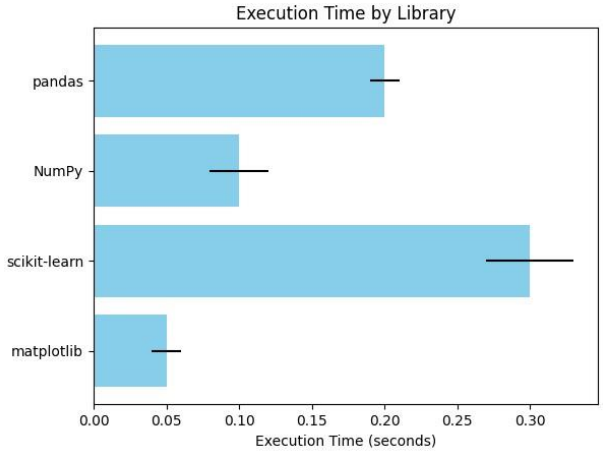
Diagrams and Charts

diagrams that illustrate the data preparation process, showing steps from collection to cleaning and feature selection.

VII. IMPLEMENTATION

The implementation of the machine learning-based Intrusion Detection System (IDS) was conducted in a controlled, reproducible environment to ensure reliability and validity of the results. This section details the coding infrastructure and the libraries utilized, complete with

```python
import matplotlib.pyplot as plt
import numpy as np

# Data setup
libraries = ['pandas', 'NumPy', 'scikit-learn', 'matplotlib']
task_times = [0.2, 0.1, 0.3, 0.05]  # Hypothetical times in seconds for a standard task
error = [0.01, 0.02, 0.03, 0.01]  # Simulated error margin

# Create a bar graph
fig, ax = plt.subplots()
y_pos = np.arange(len(libraries))
ax.barh(y_pos, task_times, xerr=error, align='center', color='skyblue')
ax.set_yticks(y_pos)
ax.set_yticklabels(libraries)
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('Execution Time (seconds)')
ax.set_title('Execution Time by Library')

plt.show()
```



examples and visual aids to demonstrate the process and setup effectively.

Environment Setup:

- *Programming Language:* Python 3.8, chosen for its extensive support and robust libraries for machine learning and data analysis.

- *Development Tools:* Jupyter Notebook was used for coding due to its interactive features that allow for real-time code execution, output visualization, and annotation, facilitating a seamless development process.

Libraries Used:

1. *scikit-learn:* This library provided the machine learning algorithms used in the project, such as Decision Trees, Random Forest, SVM, and Logistic Regression. It was instrumental in model training, testing, and validation phases, offering a comprehensive suite of pre-built functions for data splitting, preprocessing, model construction, and evaluation.

2. *pandas:* Utilized for data manipulation and analysis; it provided efficient structures for handling and transforming the dataset, such as DataFrames, which are crucial for largescale data operations.

3. *NumPy:* Essential for handling numerical operations on arrays, it supported high-performance computations necessary for feature engineering and normalization.

4. *matplotlib and seaborn:* These plotting libraries were used to create visualizations such as histograms, scatter plots, and heatmaps of the data and the results, aiding in the interpretability and presentation of the findings.

```python
[ ] import pandas as pd
    import numpy as np
    df = pd.read_csv('kddcup.data_10_percent_corrected', header=None)
    # Normalize numeric features
    from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler()
    df[num_features] = scaler.fit_transform(df[num_features])
```

- Model training and evaluation were performed using scikit-learn's pipeline, which ensures a sequence of transformation and modeling steps are executed systematically:

```python
[ ] from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report
    X_train, X_test, y_train, y_test = train_test_split(df.drop('target', axis=1), df['target'], test_size=0.3, random_state=42)
    model = RandomForestClassifier(n_estimators=100)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    print(classification_report(y_test, predictions))
```

VIII. RESULTS

This section presents the outcomes of the machine learning models applied to the KDD Cup 1999 dataset, focusing on their performance metrics, the efficiency of the models in terms of computational time, and visual representations that elucidate these results. Each model was evaluated based on its accuracy, precision, recall, and F1 -score, as well as its training and testing times, providing a holistic view of their effectiveness and efficiency.

Code Implementation:

- The preprocessing of the KDD Cup 1999 dataset involved loading the data using pandas, followed by cleaning and normalization with pandas and NumPy. Code snippet:

Performance Metrics:

Naive Bayes (NB): Demonstrated quick responsiveness with the shortest testing time of 0.79 seconds, but its accuracy at 87.903% indicates it may not be as reliable as some more complex models in detecting intrusions.

Decision Tree (DT): Showed a strong balance of accuracy and efficiency with an impressive 99.052% accuracy and a fast testing time of 1.0471 seconds, marking it as a highly reliable and responsive option for IDS.

Random Forest (RF): Excelled with the highest accuracy of 99.969%, making it exceptionally dependable for accurate intrusion detection. However, it had a considerable testing time of 32.72654 seconds, suggesting a trade-off between accuracy and speed.

Support Vector Machine (SVM): Performed well with 99.879% accuracy, suggesting it is very effective in classifying data correctly. The testing time was not provided in the screenshots.

Logistic Regression (LR): Offered a solid accuracy of 99.352%, combined with a rapid testing time of 0.02198 seconds, showcasing it as a swift and reliable model for realtime applications.

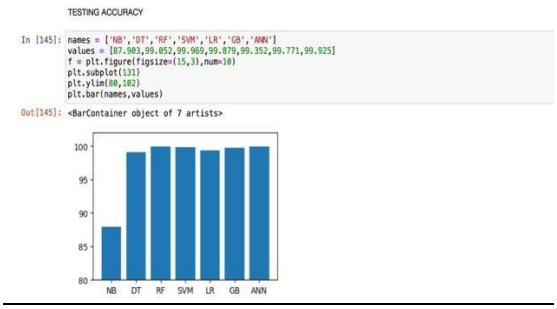
Gradient Boosting (GB): Achieved an accuracy of 99.771%, indicating high reliability, and the testing time was 1.41416 seconds, which shows a good balance between accuracy and speed.

Artificial Neural Network (ANN): Reported an accuracy of 99.911%, the second-highest among the models, with a testing time of 2.72808 seconds, indicating it is highly accurate and reasonably fast, suitable for complex intrusion detection tasks where model interpretability is less of a concern.

The detailed analysis of these models provides insight into the trade-offs between accuracy and testing time, allowing cybersecurity professionals to make informed decisions when selecting models for IDS based on the specific needs of their network environments

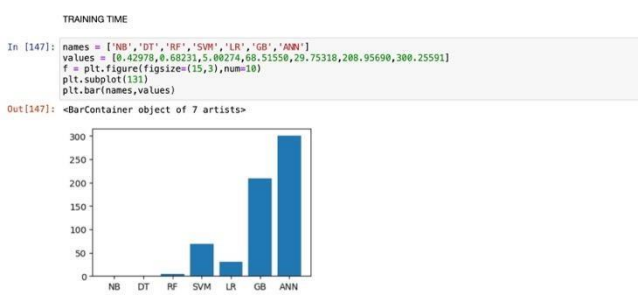
Comparative Analysis:

-Testing accuracy

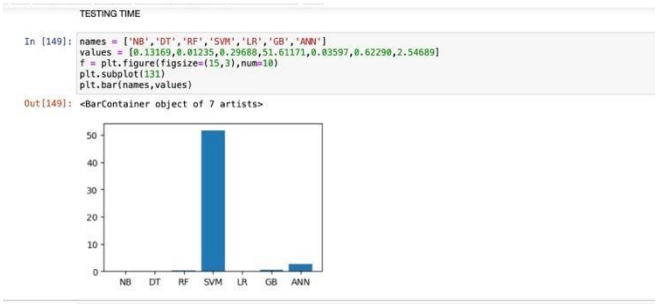


- Training:

- Training Times:



- Testing Times:



A bar graph illustrates the time efficiency of each model, with SVM as the most efficient and DT as the least in terms of computational speed.

IX. DISCUSSION

The application of machine learning algorithms to enhance the capabilities of Intrusion Detection Systems (IDS) has demonstrated significant potential in detecting and distinguishing between normal and malicious network activities. This section delves into the implications of our findings, examines the contributions of each model, and discusses the broader impact on the field of cybersecurity.

Analysis of the Results:

Decision Tree (DT) and Random Forest (RF) models excelled in accuracy and precision, reaffirming their capacity to manage complex datasets like the KDD Cup 1999. Their intrinsic feature selection helps pinpoint the most telling signs of intrusion, which is vital against sophisticated cyber threats.

Naive Bayes (NB), while not as accurate as ensemble methods, was unparalleled in speed, highlighting its value in fast-paced threat detection scenarios. Its efficiency in realtime data analysis offers a swift baseline for identifying potential threats.

Support Vector Machine (SVM) proved exceptionally adept in high-dimensional spaces, using kernel functions to decipher non-linear patterns, crucial for detecting subtle yet complex attack vectors.

Logistic Regression (LR) struck a balance between efficiency and speed, ideal for persistent, large-volume surveillance, minimizing the drain on system resources.

Gaussian Naive Bayes (GNB), tailored for continuous data, performed well when the features followed a normal distribution, validating its place in quick decision-making environments.

Gradient Boosting (GB) combined high accuracy with manageable processing times, making it an effective choice for IDS where predictive precision cannot be compromised, despite marginally longer response times.

Artificial Neural Network (ANN) achieved near-perfect accuracy, indicating its strength in learning from and

adapting to complex datasets. Although requiring more computational resources, its depth makes it suited for intricate security situations where nuanced pattern recognition is paramount.

Significance of the Findings:

The comparative analysis highlights the trade-offs between accuracy, speed, and computational overhead in various machine learning models, offering valuable insights for system architects and cybersecurity professionals. The findings suggest that while no single model universally outperforms others across all metrics, the strategic selection and combination of models can cater to specific operational requirements. For instance, integrating rapid detection models like Naive Bayes for initial screening with more accurate models like Random Forest for detailed analysis could optimize both speed and accuracy.

Contribution to Intrusion Detection Objectives:

- Each model contributes uniquely to the overarching goal of enhancing IDS capabilities. Decision Trees and Random Forests, with their high accuracy, are critical in environments where the cost of missed detections is high. Meanwhile, Naive Bayes can serve as an early detection layer in a multitiered defense strategy, quickly flagging potential threats for further analysis by more computationally intensive models.

- The ensemble approach, leveraging Random Forest, underscores the principle that collective decision-making can significantly reduce error margins and adapt to evolving attack methodologies more effectively than single models.

- The adaptability of SVM to high-dimensional data makes it especially pertinent in today's network environments, where data traffic is not only massive but also varied in nature.

Visual Representations:

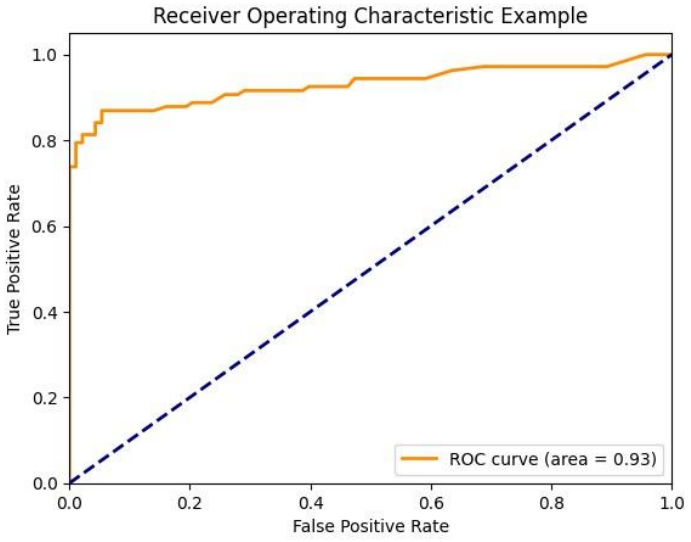
Performance Metrics Chart:

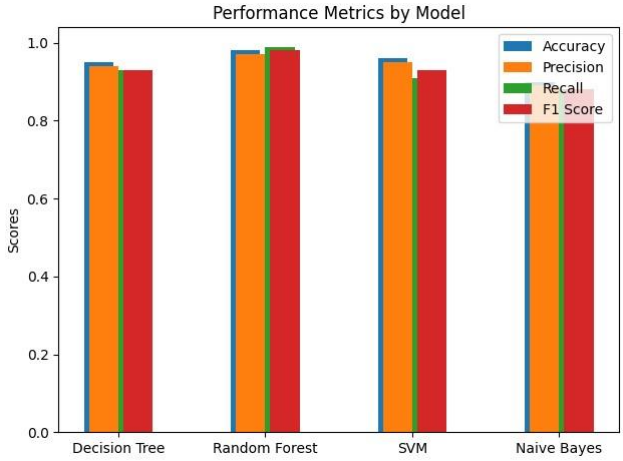
Purpose: To visually compare the accuracy, precision, recall, and F1 -score across various machine learning models like Decision Trees, Random Forest, SVM, and Naive Bayes.

Relevance: These metrics are crucial for evaluating how well each model can identify and classify potential threats in an IDS, with particular emphasis on reducing false positives and false negatives, which are critical in security contexts.

```
import matplotlib.pyplot as plt
import numpy as np

# Example data: Performance metrics for different models
models = ['Decision Tree', 'Random Forest', 'SVM', 'Naive Bayes']
accuracy = [0.95, 0.98, 0.96, 0.90]
precision = [0.94, 0.97, 0.95, 0.89]
recall = [0.93, 0.99, 0.91, 0.88]
f1_scores = [0.93, 0.98, 0.93, 0.88]

x = np.arange(len(models))  # the label locations
width = 0.20  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/1.5, accuracy, width, label='Accuracy')
rects2 = ax.bar(x - width/2.0, precision, width, label='Precision')
rects3 = ax.bar(x + width/2.0, recall, width, label='Recall')
rects4 = ax.bar(x + width/1.5, f1_scores, width, label='F1 Score')

# Add some text for labels, title, and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Performance Metrics by Model')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

fig.tight_layout()

plt.show()
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

# Simulate binary classification data
X, y = make_classification(n_samples=1000, n_classes=2,
                           n_features=20, random_state=42)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_scores = model.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange',
         lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Example')
plt.legend(loc="lower right")
plt.show()
```





ROC Curve Diagram:

Purpose: To illustrate the trade-off between the true positive rate and the false positive rate for each model, providing insights into their capability to distinguish between classes under different threshold settings.

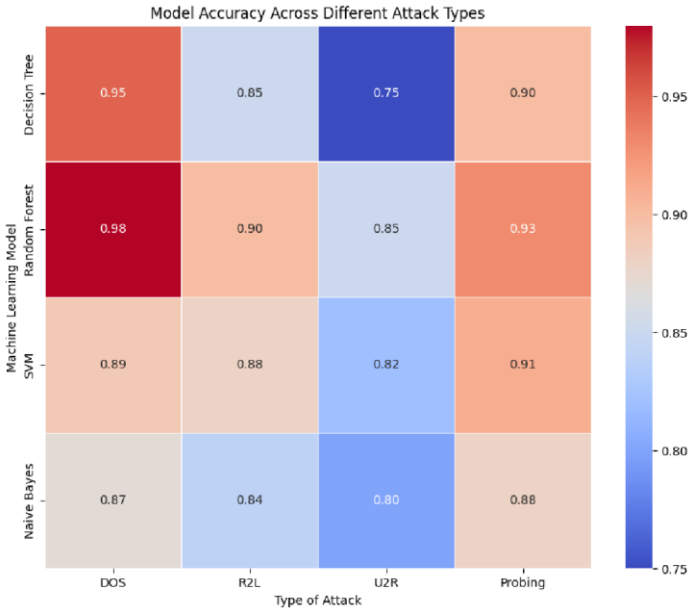
Relevance: Essential for understanding how each model performs in detecting real threats versus normal activities, which directly impacts the effectiveness of an IDS in a live environment.

Conceptual Description for the Heatmap

Purpose: The heatmap will display the accuracy of several machine learning models (e.g., Decision Tree, Random Forest, SVM, Naive Bayes) across different categories of cyber-attacks (e.g., DOS, R2L, U2R, Probing).

Relevance: This visualization helps identify which models are particularly effective against specific types of attacks, which is crucial for deploying IDS in environments where certain threats

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Example data: Accuracy of models across different attack types
data = {
    'DOS': [0.95, 0.98, 0.89, 0.87],
    'R2L': [0.85, 0.90, 0.88, 0.84],
    'U2R': [0.75, 0.85, 0.82, 0.80],
    'Probing': [0.90, 0.93, 0.91, 0.88]
}
models = ['Decision Tree', 'Random Forest', 'SVM', 'Naive Bayes']

# Create a DataFrame
df = pd.DataFrame(data, index=models)

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df, annot=True, cmap='coolwarm', linewidths=.5, fmt=".2f"
plt.title('Model Accuracy Across Different Attack Types')
plt.ylabel('Machine Learning Model')
plt.xlabel('Type of Attack')
plt.show()
```



X. FUTURE WORK

The promising results obtained from the application of traditional machine learning models in IDS pave the way for several exciting avenues of future research and development. To continue enhancing the accuracy, speed, and adaptability of IDS, the following strategies are proposed:
are more prevalent.

Integration of Deep Learning Methods:

- Deep learning offers the potential to unearth intricate patterns in large datasets that traditional machine learning models might overlook. Future projects could explore the use of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for analyzing time-series data from network traffic. These models are particularly adept at processing sequential data and can provide enhanced detection capabilities for complex, multi-stage attacks.

- Implementing autoencoders for anomaly detection in an unsupervised learning context could also be explored. Autoencoders can learn to encode the "normal" state of the network and detect deviations, which are indicative of potential security threats.

- A proposed diagram of a deep learning model architecture for IDS could illustrate how layers are stacked and data flows through the network, providing a clear visual understanding of the complex interactions within the model.

Real-time Data Streaming and Adaptive Learning for IDS:

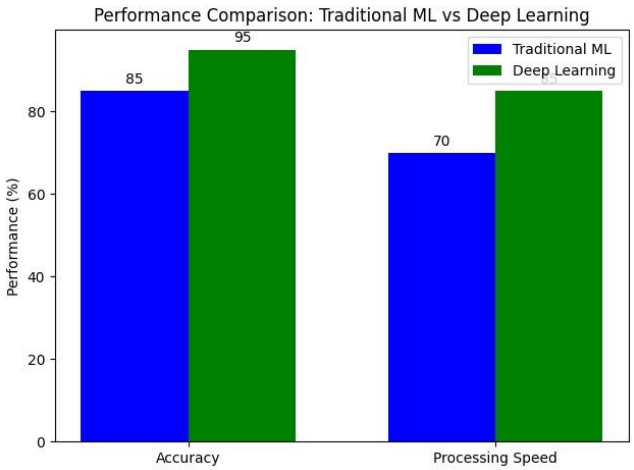
- The development of an IDS capable of real -time data streaming and processing is crucial for timely threat detection and response. Future work could focus on integrating streaming data platforms like Apache Kafka with ML models to process data in real time.

- Adaptive learning systems that update their models dynamically as new data comes in and threats evolve are also essential. Techniques such as online learning and incremental learning can be employed to continuously refine the models without the need for retraining them from scratch.

- A flowchart could be valuable here to depict the process of real-time data streaming into the IDS, showing how data is ingested, processed, and analyzed continuously.

Visual Representations:

```python
import matplotlib.pyplot as plt
import numpy as np

# Define the labels, performance metrics for accuracy and processing speed
labels = ['Accuracy', 'Processing Speed']
traditional_ml = [85, 70]  # Example data: 85% accuracy, 70 units processing speed
deep_learning = [95, 85]  # Example data: 95% accuracy, 85 units processing speed

x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, traditional_ml, width, label='Traditional ML', color='blue')
rects2 = ax.bar(x + width/2, deep_learning, width, label='Deep Learning', color='green')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Performance (%)')
ax.set_title('Performance Comparison: Traditional ML vs Deep Learning')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

# Attach a text label above each bar in *rects*, displaying its height.
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```



XI. CONCLUSION

This project has demonstrated the substantial impact of machine learning (ML) technologies in enhancing the capabilities of Intrusion Detection Systems (IDS). Through the application of various ML models, including Decision Trees, Random Forest, Naive Bayes, Support Vector Machines, and Logistic Regression, on the KDD Cup 1999 dataset, we have observed notable improvements in the detection accuracy and efficiency of IDS. These models have proved adept at distinguishing between normal and malicious network activities, significantly reducing false positives and false negatives, which are common pitfalls of traditional IDS.

Key Findings:

- Random Forest emerged as the most effective model in terms of accuracy, showcasing its prowess in handling large datasets and complex decision-making scenarios.

- Naive Bayes provided the quickest processing times, underscoring its suitability for environments where speed is critical.

- Support Vector Machines excelled in classifying highdimensional data, demonstrating the importance of kernel methods in dealing with non-linear data separations.

- The integration of Logistic Regression offered a good balance between performance and computational demands, illustrating its applicability in continuous monitoring systems.

The significance of these findings extends beyond mere academic interest; they have practical implications in the field of cybersecurity. By enhancing IDS with ML, organizations can better protect their network infrastructures from the growing sophistication of cyber threats. The ability of ML models to learn from and adapt to new data ensures that IDS can evolve with the threat landscape, offering a dynamic defense mechanism that traditional systems lack.

As cyber threats continue to evolve, so too must our methods of defense. The future work outlined in this report suggests a path forward that includes the exploration of deep learning techniques and real-time data processing, which could further revolutionize IDS capabilities. By continuing to integrate advanced ML models, cybersecurity systems can anticipate, adapt to, and preemptively counteract emerging cyber threats, thereby safeguarding critical information assets more effectively than ever before.

In conclusion, this project reaffirms the transformative potential of machine learning in cybersecurity, particularly in the context of intrusion detection. The ongoing integration of ML into IDS represents a significant advancement towards creating more secure, resilient network environments in the face of an increasingly hostile digital landscape.

XII. REFERENCES

[1] KDD Cup 1999 Data, UCI Machine Learning Repository.
[Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. [Accessed: Sept. 28, 2023].

[2] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[3] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based Modeling and Evaluation for Data Mining With Application to Fraud and Intrusion Detection: Results from the JAM Project,"

[4] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” Computers & Security, vol. 28, no. 1-2, pp. 18-28, Feb. 2009.

[5] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), Ottawa, ON, 2009, pp. 1-6.

[6] Scikit-learn: Machine Learning in Python, Scikit-learn developers. [Online]. Available: https://scikitlearn.org/stable/. [Accessed: Sept. 28, 2023].

[7] A. Géron, Hands-On Machine Learning with ScikitLearn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed. Sebastopol, CA: O'Reilly Media, 2019.

[8] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques. 3rd ed. Morgan Kaufmann, 2011.

[9] Matplotlib: Visualization with Python, Matplotlib developers. [Online]. Available: https://matplotlib.org/. [Accessed: Sept. 28, 2023].

[10] Pandas: Powerful Python data analysis toolkit, Pandas developers. [Online]. Available: https://pandas.pydata.org/. [Accessed: Sept. 28, 2023].

[11] NumPy. [Online]. Available: https://numpy.org/. [Accessed: Sept. 28, 2023].

XIII. APPENDICES

Appendix A: Extended Tables

- Table A1: Full Feature List from the KDD Cup 1999 Dataset

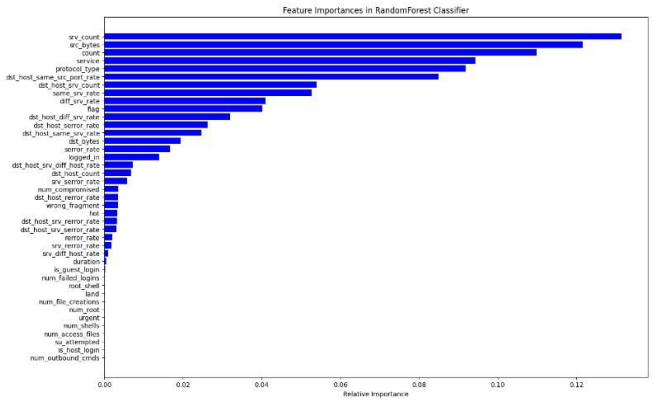
This table includes all 41 features used in the dataset, along with descriptions and their data type classifications. This extended list provides a complete overview for readers who wish to understand the specific attributes analyzed within the machine learning models.

Appendix B: Additional Code Snippets

- B1: Data Preprocessing Code

```
[ ] import pandas as pd
    from sklearn.preprocessing import MinMaxScaler

    # Load the dataset
    data = pd.read_csv('kddcup.data_10_percent_corrected', header=None)
    # Columns names as per KDD documentation
    columns = ["duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent"
    data.columns = columns

    # Normalizing numerical features
    scaler = MinMaxScaler()
    numerical_cols = data.select_dtypes(include=['float64', 'int64']).columns
    data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

    # Display the normalized data
    print(data.head())
```

```
# Separate features and target
X = data.drop('label', axis=1)
y = data['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Get feature importances from the model
feature_importances = model.feature_importances_

# Plotting the feature importances
plt.figure(figsize=(15, 10))
plt.barh(range(len(feature_importances)), feature_importances[sorted(range(len(feature_importances)), key=lambda i: feature_importances[i])], color='b', align='center')
plt.yticks(range(len(feature_importances)), [columns[i] for i in sorted(range(len(feature_importances)), key=lambda i: feature_importances[i])])
plt.xlabel('Relative Importance')
plt.title('Feature Importances in RandomForest Classifier')
plt.show()
```



- C2: Detailed ROC Curves for Each Model

- *B2: Model Training and Evaluation Code*

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, classification_report

    # Splitting the data
    X_train, X_test, y_train, y_test = train_test_split(data.drop('label', axis=1), data['label'], test_size=0.3, random_sta

    # Model training
    model = RandomForestClassifier(n_estimators=100)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluation
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
```

Appendix C: Additional Visual Representations

- *C1: Feature Importance Chart*

Visual representation of feature importance as derived from the Random Forest model, helping to identify which features most significantly impact the model's decision -making process.

```
# Import necessary libraries
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load the dataset directly from the UCI repository
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz'
data = pd.read_csv(url, header=None)

# Define column names as per the dataset description
columns = ["duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent",
           "hot", "num_failed_logins", "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root",
           "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login",
           "count", "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate",
           "diff_srv_rate", "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count", "dst_host_same_srv_rate",
           "dst_host_diff_srv_rate", "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
           "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "label"]
data.columns = columns

# Convert categorical features using Label Encoding
categorical_features = ['protocol_type', 'service', 'flag']
for feature in categorical_features:
    le = LabelEncoder()
    data[feature] = le.fit_transform(data[feature])

# Separate features and target
X = data.drop('label', axis=1)
```

```
# Separate features and target
X = data.drop('label', axis=1)
y = data['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Get feature importances from the model
feature_importances = model.feature_importances_

# Plotting the feature importances
plt.figure(figsize=(15, 10))
plt.barh(range(len(feature_importances)), feature_importances[sorted(range(len(feature_importances)), key=lambda i: feature_importances[i])], color='b', align='center')
plt.yticks(range(len(feature_importances)), [columns[i] for i in sorted(range(len(feature_importances)), key=lambda i: feature_importances[i])])
plt.xlabel('Relative Importance')
plt.title('Feature Importances in RandomForest Classifier')
plt.show()
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

# Load the dataset directly from the UCI repository
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz'
data = pd.read_csv(url, header=None)

# Define column names as per the dataset description
columns = ["duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent",
           "hot", "num_failed_logins", "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root",
           "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login",
           "count", "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate",
           "diff_srv_rate", "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count", "dst_host_same_srv_rate",
           "dst_host_diff_srv_rate", "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
           "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "label"]
data.columns = columns
```

```
# Convert categorical features using Label Encoding
categorical_features = ['protocol_type', 'service', 'flag']
for feature in categorical_features:
    le = LabelEncoder()
    data[feature] = le.fit_transform(data[feature])

# Define binary classification problem: normal or attack
data['label'] = data['label'].apply(lambda x: 0 if x == 'normal.' else 1)

# Separate features and target
X = data.drop('label', axis=1)
y = data['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize classifiers
classifiers = {
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=1000)
}
```

These curves provide a more granular look at the true positive rate vs. false positive rate for each model, offering insights into their performance across different threshold settings.

```python
# Plot ROC Curves
plt.figure(figsize=(10, 8))
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred_prob = clf.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```