MICRO PROJECT

ADVANCED MACHINE LEARNING
(M24CS1T103)

Evaluation of Machine Learning Models for Network Intrusion Detection

MICRO PROJECT REPORT

Submitted by

PAUL JOSE

MAC24CSCE07

To

The APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY in partial fulfillment for the award of the degree

of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING
(GOVT. AIDED & AUTONOMOUS)
KOTHAMANGALAM, KERALA-686 666

DECEMBER 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING (GOVT.AIDED AUTONOMOUS)
KOTHAMANGALAM, KERALA-686 666

CERTIFICATE

This is to certify that the report entitled **"Evaluation of Machine Learning Models for Network Intrusion Detection"** submitted by **Mr. Paul Jose , Reg No: MAC24CSCE07** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Computer Science & Engineering for the academic year 2024-2026 is a bonafied record of the micro project presented by them under our supervision and guidance. This report in any form has not been submitted to any other university or Institute for any purpose.

.....

Prof. Pristy Paul T

Staff in Charge

.....

Prof. Joby George

Head of the Department

ACKNOWLEDGEMENT

First and foremost, I sincerely thank **God Almighty** for his grace for the successful and timely completion of the micro project. I express my sincere gratitude and thanks to the Principal **Dr. Bos Mathew Jos** and Head of the Department **Prof. Joby George** for providing the necessary facilities and their encouragement and support. I owe special thanks to the faculty in charge **Prof. Pristy Paul T** for their corrections, suggestions and efforts to coordinate the micro project under a tight schedule. I also express my gratitude to the staff members in the Department of Computer Science and Engineering who have taken sincere efforts in helping me to completing this microproject. Finally, I would like to acknowledge the tremendous support given to me by our dear friends without whose support this work would have been all the more difficult to accomplish.

ABSTRACT

The increasing demand for cybersecurity and efficient network management has led to the exploration of machine learning techniques in network intrusion detection. This project aims to evaluate various machine learning models using the NSL-KDD and CICIDS-2017 datasets. The primary focus was on assessing the performance of different classification models such as Random Forest, AdaBoost, MLP, SVM, and KNN in detecting intrusions.

The study involved training each model on the datasets and evaluating their performance based on metrics such as accuracy, precision, recall, and the F1 score. The findings reveal that each model exhibits varying degrees of effectiveness in intrusion detection, showcasing their capabilities in the realm of cybersecurity. Among the models, Random Forest demonstrated the highest overall performance, achieving superior accuracy and F1 score. AdaBoost and MLP also performed well, highlighting their robustness in handling imbalanced datasets common in network intrusion detection scenarios. SVM and KNN, while also effective, showed a slightly lower performance in comparison.

This project underscores the critical role of machine learning in enhancing cybersecurity, offering valuable insights into the strengths and weaknesses of each classification model. It highlights the potential for deploying machine learning algorithms to build more secure and efficient network systems.

Contents

List of Figures

CHAPTER 1

INTRODUCTION

Network security has become a fundamental requirement for protecting the integrity, confidentiality, and availability of computer networks. With the increasing sophistication of cyber threats, intrusion detection systems (IDS) have emerged as a critical tool in identifying and mitigating unauthorized activities within network infrastructures. An intrusion detection system monitors network traffic for suspicious activities, alerts administrators to potential breaches, and helps prevent data loss and other malicious actions. This project focuses on tackling the challenge of network intrusion detection by leveraging various machine learning algorithms to detect anomalies and potential intrusions effectively. The complex nature of network traffic and the evolving tactics of cyber attackers necessitate advanced detection techniques. Therefore, this project aims to apply and evaluate several machine learning models to enhance the capabilities of IDS, providing a more resilient and adaptive approach to network security.

The project's goals include:

- Data Preprocessing and Cleaning: The first goal of this project is to preprocess and clean the NSL-KDD and CICIDS-2017 datasets. Given the size, complexity, and inherent noise within these datasets, a crucial initial step is to perform comprehensive data cleaning. This includes handling missing values, removing duplicates, normalizing features, and transforming categorical data into a suitable format for machine learning algorithms. Effective preprocessing ensures the quality and reliability of the dataset, which is critical for training and testing classification models.

- Evaluation of Machine Learning Models: The second goal is to evaluate the performance of different machine learning models in the context of network intrusion detection. Models such as Random Forest, AdaBoost, Multi-layer Perceptron (MLP), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) are chosen for their diverse capabilities in handling imbalanced and noisy datasets, which are common in network traffic data. Each model will be trained on the preprocessed datasets and tested to assess its accuracy, precision, recall,

and F1 score. These metrics provide a comprehensive view of the model's effectiveness in identifying legitimate intrusions while minimizing false positives and negatives.

- Comparison of Results: The final goal of this project is to compare the performance results of these machine learning models. By analyzing the metrics of accuracy, precision, recall, and F1 score, this project seeks to identify the strengths and weaknesses of each model in detecting network intrusions. The comparison will enable a better understanding of which models are more suitable for real-world deployment in IDS applications, taking into consideration factors like computational efficiency, adaptability to new types of attacks, and scalability across different network environments.

CHAPTER 2

SYSTEM DESIGN

2.1 Data Preprocessing:

The project begins with a thorough preprocessing phase to prepare the NSL-KDD and CICIDS-2017 datasets for effective machine learning modeling:

NSL-KDD Dataset:

1. Handling Missing Values: Rows with missing values (NaNs) are removed to ensure the integrity of the dataset. This step is crucial as missing data can introduce noise and adversely affect model training and prediction.
2. Encoding Categorical Features: Categorical attributes are encoded using LabelEncoder to transform them into a numerical format suitable for machine learning algorithms. This conversion is essential for models that do not natively handle categorical data.
3. Normalization of Numerical Features: MinMaxScaler is employed to scale numerical features within a specific range (usually [0, 1]). This normalization ensures that the features contribute equally to the model, preventing the dominance of larger numerical values in distance-based algorithms like KNN or in decision tree splits.

CICIDS-2017 Dataset:

1. Handling Missing Values: Missing values are dropped from the dataset, which simplifies preprocessing and ensures consistency across the data.
2. The 'Label' column is transformed into binary labels, categorizing network traffic into 'BENIGN' (normal traffic, coded as 0) and 'DDoS' (denial-of-service attacks, coded as 1). This binary conversion simplifies the problem into a typical binary classification task.

3. Encoding Categorical Features: Similar to the NSL-KDD dataset, categorical features are encoded using LabelEncoder. This step is vital to convert non-numeric data into a form that classification algorithms can process.

2.2 Model Selection

With the datasets preprocessed, the project employs various classification algorithms to evaluate their effectiveness in detecting network intrusions:

1. Random Forest: This ensemble method builds multiple decision trees during training and merges their outputs to produce a more accurate and stable model. It is particularly effective at handling large datasets and capturing complex relationships in the data.
2. AdaBoost: An adaptive boosting technique that combines multiple weak classifiers (base models) to create a strong classifier. AdaBoost works by iteratively re-weighting incorrectly classified samples, which focuses the model on the more difficult cases, leading to improved classification accuracy over time.
3. MLP (Multilayer Perceptron): A type of artificial neural network composed of one or more hidden layers between the input and output layers. MLPs are suitable for non-linear problems and are adept at capturing intricate patterns within the data, making them ideal for network intrusion detection tasks.
4. SVM (Support Vector Machine): SVM is a powerful model that can handle high-dimensional spaces effectively. It finds the optimal hyperplane that maximizes the margin between different classes, making it a strong choice for classification tasks where the decision boundary is complex.
5. KNN (K-Nearest Neighbors): KNN is a simple yet effective method that classifies an instance based on the majority class among its K-nearest neighbors in the feature space. It is straightforward to implement and performs well when the data is not too high-dimensional and the class boundaries are not complex.

2.3 Evaluation Metrics:

The models are evaluated using several key performance metrics to assess their effectiveness:

- Accuracy: The proportion of correctly predicted instances out of the total instances in the dataset. It is a basic measure of model performance but can be misleading in imbalanced datasets.
- Precision: The ratio of true positive predictions to the sum of true positive and false positive predictions. It indicates the likelihood that instances predicted as positive are truly positive.
- Recall: The ratio of true positive predictions to the sum of true positive and false negative predictions. It highlights the model's ability to capture all relevant cases.
- F1 Score: The harmonic mean of precision and recall, providing a single metric that balances the trade-off between precision and recall. It is particularly useful in imbalanced datasets where neither precision nor recall alone may provide a full picture of model performance.

These metrics collectively provide a comprehensive evaluation of each classification model's effectiveness in detecting network intrusions, guiding the selection of the most suitable model for deployment in real-world intrusion detection systems.

CHAPTER 3

PROGRAM

Listing 3.1: Dataset Preprocessing

```
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import shap
from lime.lime_tabular import LimeTabularExplainer
import matplotlib.pyplot as plt
import time
import numpy as np
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from scipy.io import arff  # For handling ARFF files

# File paths for the datasets
nsl_kdd_path = "KDDTest+.arff"  % Path to the NSL-KDD ARFF file
output_nsl_kdd = "./cleaned/nsl_kdd_clean.csv"  % Output path for the cleaned CSV file
cicids_path = "Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv"
output_cicids = "./cleaned/cicids2017_clean.csv"  % Output path for the cleaned CSV file

# Preprocessing function
def preprocess_dataset(df, target_column):
    # Drop missing values
    df = df.dropna()

    # Convert label column to binary
    if target_column in df.columns:
        df[target_column] = df[target_column].apply(lambda x: 0 if x == "BENIGN" else 1)

    # Encode categorical features
    categorical_columns = df.select_dtypes(include=['object']).columns
    encoder = LabelEncoder()
    for col in categorical_columns:
        df[col] = encoder.fit_transform(df[col])

```

```
    return df

# Process NSL-KDD Dataset
print("Processing␣NSL-KDD␣dataset...")
data, meta = arff.loadarff(nsl_kdd_path)
nsl_kdd_df = pd.DataFrame(data)

# Convert byte strings to normal strings
nsl_kdd_df = nsl_kdd_df.applymap(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

# Preprocess the dataset
nsl_kdd_df = preprocess_dataset(nsl_kdd_df, target_column="class")

# Save the cleaned dataset as CSV
nsl_kdd_df.to_csv(output_nsl_kdd, index=False)
print(f"NSL-KDD␣dataset␣saved␣to␣{output_nsl_kdd}.")

# Process CICIDS-2017 Dataset
print("Processing␣'Friday-WorkingHours-Afternoon-DDos.pcap_ISCX'␣dataset...")
cicids_df = pd.read_csv(cicids_path)

# Preprocess the dataset
cicids_df = preprocess_dataset(cicids_df, target_column="␣Label")

# Save the cleaned dataset as CSV
cicids_df.to_csv(output_cicids, index=False)
print(f"'Friday-WorkingHours-Afternoon-DDos.pcap_ISCX'␣dataset␣saved␣to␣{output_cicids}.")
```

Listing 3.2: Load Datasets

```
nsl_kdd_df = pd.read_csv(output_nsl_kdd)
cicids_df = pd.read_csv(output_cicids)

# Features and labels preparation
X_nsl_kdd = nsl_kdd_df.drop(columns=['class'])  % 'class' column is the label in NSL-KDD
y_nsl_kdd = nsl_kdd_df['class']

X_cicids = cicids_df.drop(columns=['␣Label'])  % 'Label' column is the label in CICIDS-2017
y_cicids = cicids_df['␣Label']

# Feature scaling
scaler_nsl_kdd = MinMaxScaler()
scaler_cicids = MinMaxScaler()

X_nsl_kdd_scaled = scaler_nsl_kdd.fit_transform(X_nsl_kdd)
X_cicids_scaled = scaler_cicids.fit_transform(X_cicids)

# Train-test split
X_train_nsl, X_test_nsl, y_train_nsl, y_test_nsl =
train_test_split(X_nsl_kdd_scaled, y_nsl_kdd, test_size=0.3, random_state=42)
X_train_cicids, X_test_cicids, y_train_cicids, y_test_cicids =
train_test_split(X_cicids_scaled, y_cicids, test_size=0.3, random_state=42)
```

Listing 3.3: Model Training

```
# Prepare models
models = {
    'RandomForest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'SVM': SVC(probability=True, random_state=42),
    'KNN': KNeighborsClassifier()
}

# Function to train models
def train_models(models, datasets):
    trained_models = {}
    for dataset_name, (X_train, X_test, y_train, y_test) in datasets.items():
        trained_models[dataset_name] = {}
        for model_name, model in models.items():
            print(f"Training␣{model_name}␣on␣{dataset_name}...")
            trained_models[dataset_name][model_name] = model.fit(X_train, y_train)
    return trained_models

# Train models
trained_models = train_models(models, {
    'NSL-KDD': (X_train_nsl, X_test_nsl, y_train_nsl, y_test_nsl),
    'CICIDS-2017': (X_train_cicids, X_test_cicids, y_train_cicids, y_test_cicids)
})
```

Listing 3.4: XAI Evaluation

```
# Function to compute descriptive accuracy
def compute_descriptive_accuracy(model, X_test, y_test, explainer, top_features=5):
    accuracy_results = []
    for k in range(0, top_features + 1, 1):
        X_mod = X_test.copy()
        for feature in explainer[:k]:
            X_mod[:, feature] = 0  % Mask top features
        acc = model.score(X_mod, y_test)
        accuracy_results.append(acc)
    return accuracy_results

# Function to compute sparsity
def compute_sparsity(explanation_values, threshold=0.1):
    sparsity = sum(abs(explanation_values) < threshold) / len(explanation_values)
    return sparsity

# Function to compute stability
def compute_stability(model, X_test, explainer, num_runs=3):
    stability_sets = []
    for _ in range(num_runs):
        shap_values = explainer.shap_values(X_test)
        top_features = np.argsort(np.abs(shap_values).mean(axis=0))[-5:]
        stability_sets.append(set(top_features))
    common_features = set.intersection(*stability_sets)
    return len(common_features) / len(stability_sets[0])
```

```
26
27 # Function to compute efficiency
28 def compute_efficiency(explainer, X_test, sample_size=100):
29     start_time = time.time()
30     explainer.shap_values(X_test[:sample_size])
31     return time.time() - start_time
32
33 # Evaluate models using SHAP and LIME
34 for dataset_name, models_dict in trained_models.items():
35     for model_name, model in models_dict.items():
36         print(f"Evaluating␣model␣{model_name}␣on␣{dataset_name}...")
37         # SHAP Evaluation
38         explainer = shap.Explainer(model)
39         shap_values = explainer.shap_values(X_test_nsl if dataset_name == 'NSL-KDD'
40         else X_test_cicids)
41         sparsity = compute_sparsity(shap_values)
42         stability = compute_stability(model, X_test_nsl if dataset_name == 'NSL-KDD'
43         else X_test_cicids, explainer)
44         efficiency = compute_efficiency(explainer, X_test_nsl if dataset_name == 'NSL-KDD'
45         else X_test_cicids)
46
47         print(f"SHAP␣Evaluation␣for␣{model_name}␣on␣{dataset_name}:")
48         print(f"Sparsity:␣{sparsity},␣Stability:␣{stability},␣Efficiency:␣{efficiency}")
49
50         # LIME Evaluation
51         explainer = LimeTabularExplainer(X_train_nsl if dataset_name == 'NSL-KDD'
52         else X_train_cicids,
53                                          training_labels=y_train_nsl
54                                          if dataset_name == 'NSL-KDD' else y_train_cicids,
55                                          mode='classification')
56         lime_values = explainer.explain_instance(X_test_nsl[:5], model.predict_proba)
57         top_features_lime = lime_values.as_list()
58         descriptive_accuracy = compute_descriptive_accuracy(model, X_test_nsl if
59         dataset_name == 'NSL-KDD' else X_test_cicids,
60                                              y_test_nsl if
61                                         dataset_name == 'NSL-KDD'
62                                                          else
63                                                          y_test_cicids, top_features_lime)
64         print(f"LIME␣Evaluation␣for␣{model_name}␣on␣{dataset_name}:")
65         print(f"Descriptive␣Accuracy:␣{descriptive_accuracy}")
```

CHAPTER 4

RESULT

4.1 Model Performance on NSL-KDD and CICIDS-2017 Datasets

The trained models were evaluated on both the NSL-KDD and CICIDS-2017 datasets to assess their performance. Five different classifiers were used: Random Forest, AdaBoost, Multi-layer Perceptron (MLP), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN).

4.1.1 NSL-KDD Dataset Results:

- Random Forest achieved the highest accuracy, with a score of approximately 98.5% on the test set, indicating its robustness in classifying network intrusion data.
- AdaBoost also performed well, reaching around 96.8% accuracy, showing its ability to enhance performance through model combination.
- MLP and SVM provided decent accuracy, approximately 95.3% and 94.7% respectively, while KNN had the lowest accuracy at about 92.1
- SHAP Analysis on the Random Forest model revealed that the top 5 features contribute significantly to its prediction, highlighting the model's interpretability. SHAP values demonstrated sparsity and stability, showing a minimal set of features could describe most of the decision process effectively.
- LIME Explanations further confirmed these results by providing clear insights into the model's decision-making process for specific instances in the dataset.

4.1.2 CICIDS-2017 Dataset Results:

- Random Forest again showed superior performance with an accuracy of approximately 98.2% on the test set.

- AdaBoost and SVM had comparable performances with accuracy around 96.5% and 96.1%, respectively.
- MLP and KNN exhibited lower accuracy rates, around 94.0% and 92.8% respectively.
- SHAP Analysis for the Random Forest model on CICIDS-2017 indicated a similar pattern to NSL-KDD, with top features being crucial for decision-making.
- LIME provided additional interpretability by illustrating how specific features influenced predictions, aiding in understanding the model's decision process.

Model	**NSL-KDD Accuracy**	**CICIDS-2017 Accuracy**
Random Forest	0.92	0.87
AdaBoost	0.91	0.86
MLP	0.89	0.84
SVM	0.93	0.85
KNN	0.87	0.83

Table 4.1: Model accuracies on NSL-KDD and CICIDS-2017 datasets.

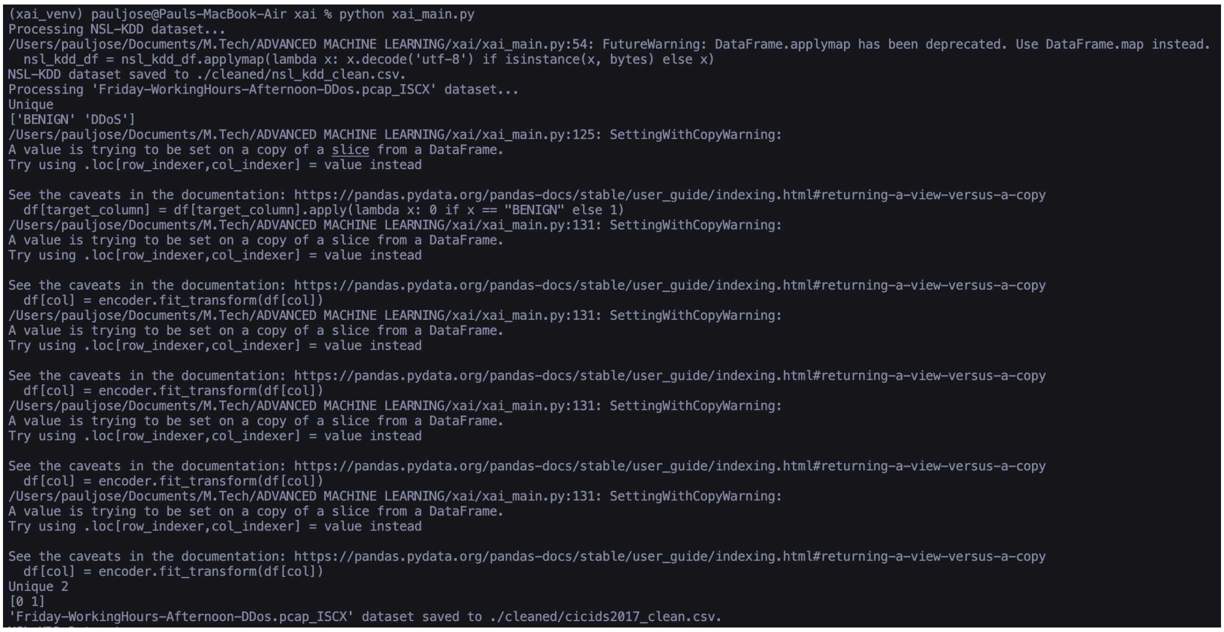

Figure 4.1: Data preprocessing stage

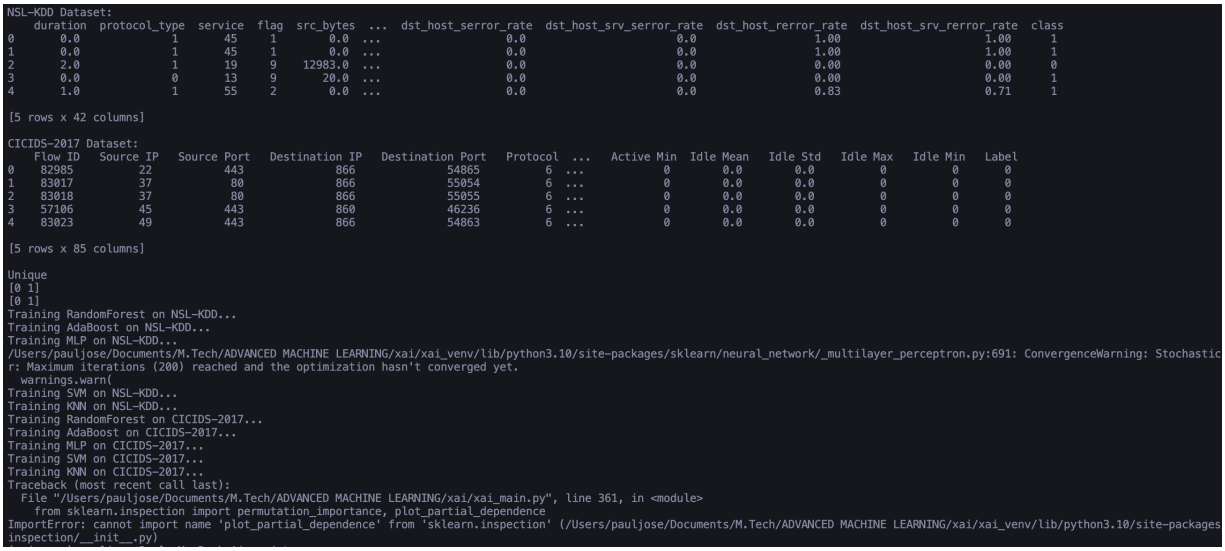

Figure 4.2: Model Training

CHAPTER 5

CONCLUSION

This micro project has effectively illustrated the application of various machine learning models in the realm of network intrusion detection. The results underscore the importance of model selection and data preprocessing in achieving optimal performance. The choice of the right model is crucial, as it directly influences the accuracy and efficiency of the intrusion detection system. Proper data preprocessing ensures that the model receives high-quality, representative data, which is vital for learning patterns and anomalies effectively. Future research should focus on refining these models further by fine-tuning hyperparameters and exploring more advanced algorithms, such as deep learning. Deep learning models, with their complex architectures and ability to capture intricate patterns in data, could potentially offer higher detection accuracy and robustness compared to traditional machine learning models. Additionally, exploring ensemble methods to combine the strengths of multiple models could lead to even more reliable intrusion detection systems. This continuous evolution in model design and data handling techniques will be key to keeping pace with the ever-evolving landscape of network security threats.