INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–9
Code:IUIT-2025

MACHINE LEARNING

Anomaly Detection in Network Traffic Using Machine Learning

Sneha Santhosh, Deepasree Pradeep, Devika S
Saintgits Group of Institutions, Kottayam, Kerala

Abstract: With the expansion of online systems and increasing network vulnerability, the detection of malicious activity—especially DDoS attacks—is critical. Manual methods are slow and often ineffective. NetGuardAI is a machine learning-based system that automates the detection of network traffic anomalies, particularly focusing on DDoS patterns. The project enables users to upload or generate traffic data, processes it with multiple machine learning models, and highlights the best-performing one. Visualizations and feature importance charts enhance interpretability, while export features allow for reportable outputs. The models used include both classical (Random Forest, Logistic Regression) and unsupervised (Isolation Forest), with Random Forest emerging as the most accurate.

Keywords: network traffic, anomaly detection, DDoS, Isolation Forest, Random Forest, supervised learning, Flask application, feature importance

1 Introduction

Network traffic represents the continuous flow of data between connected systems. This traffic can be legitimate or malicious, and identifying threats early is critical. DDoS attacks, where multiple systems flood a target, are among the most disruptive. Traditional firewalls and manual inspection can't always keep pace.

NetGuardAI proposes a smart system that automates anomaly detection in traffic logs, using machine learning. By training multiple models and comparing their performance, the system gives users insights into threats without needing deep domain expertise. The application is designed for easy use, interpretability, and extensibility into real-time or production systems.

2 Libraries Used

In the project for various tasks, following packages are used.

```
NumPy
Pandas
Seabor
n  NLTK
Matplot
lib
BERT
Scikit-learn
HTML/CSS (frontend styling)
Jupyter Notebook / Google Colab (for development)
```

3 Methodology

In this work, six machine learning models are implemented and evaluated for the detection of anomalies in network traffic, with a particular focus on identifying Distributed Denial of Service (DDoS) attacks. The models used include Logistic Regression, Random Forest Classifier, Naive Bayes, Support Vector Machine (SVM), Passive-Aggressive Classifier, and Isolation Forest (unsupervised). Among these, the Random Forest Classifier demonstrated superior performance in terms of accuracy and other evaluation metrics. Various stages in the implementation process are:

- **Data Loading & Generation:** Users can upload .csv traffic logs or click "Generate" to create synthetic traffic with DDoS and normal samples.
- **Preprocessing & Cleaning:** Checks for missing values, encodes labels into binary format (0/1), and removes irrelevant features.
- **Feature Extraction**: Features like duration, bytes sent, connection counts, etc. are selected. Scaling is applied where needed.
- **Dataset Preparation:** Data split into training and testing sets using Scikit-learn's train_test_split().
- **Model Training:** Five machine learning models are used: Isolation Forest (unsupervised), Random Forest Classifier, Logistic Regression, Naive Bayes, Support Vector Machine. Choose a classification algorithm, such as Naive Bayes, Logistic Regression and train the model using the training set
- **Model Evaluation**: Metrics evaluated: Accuracy, Precision, Recall, F1-score Best- model selected using F1-score

4 Implementation

As the first step in the task, a .csv file containing labelled network traffic logs is either uploaded by the user or synthetically generated within the application interface. These files are stored locally and then converted into a Pandas Data-Frame for preprocessing and analysis. The dataset typically contains features such as duration, src_bytes, dst_bytes, count, srv_count, and a categorical label indicating "normal" or "ddos" traffic. Exploratory Data Analysis (EDA) on the class distribution is performed, revealing that the generated dataset is class-balanced. From the initial visualization in Figure 1, it is observed that approximately 72% of the samples are labeled as "normal," while 28% are labeled "ddos," ensuring adequate representation for both traffic types.

The preprocessing stage is completed in the following four distinct and sequential steps:

- **Encoding**: The target label column is encoded into numerical form (normal as 0 and
- ddos as 1).
- **Missing Value Handling**: All entries are checked for null values and dropped or imputed accordingly.
- **Feature Scaling**: For selected algorithms like SVM and Logistic Regression , StandardScaler is applied to normalize the feature space.
- **Balancing**: In synthetic generation mode, an equal number of normal and ddos entries are ensured to avoid bias in training.

This process of implementation was conducted entirely in Python, using libraries such as pandas, numpy, seaborn, and scikit-learn. All development and testing were carried out locally on Jupyter Notebook and Google Colab, and model training took approximately 30 minutes for the full pipeline, including preprocessing, feature extraction, training, and evaluation.

During the feature engineering phase, key attributes from network traffic logs—such as connection duration, bytes sent, and request counts—were extracted. These features were then normalized using StandardScaler to accommodate algorithms like SVM and Logistic Regression. The dataset was balanced and label-encoded, ensuring fair training conditions across both normal and DDoS classes.

The training phase involved the implementation of six machine learning algorithms: Logistic Regression, Naive Bayes, Support Vector Machine (SVM), Random Forest Classifier, Passive-Aggressive Classifier, and Isolation Forest. Among these, Isolation Forest was employed as an unsupervised anomaly detection baseline, while the remaining models operated under supervised learning paradigms. The models were trained using a 70:30 train-test split and evaluated through standard classification metrics including accuracy, precision, recall, and F1-score. For enhanced interpretability and system usability, feature importance charts and accuracy comparison graphs were generated during model evaluation. These visualizations assist in understanding both model performance and decision factors.

The system also includes a web-based interface built with Flask, allowing users to upload traffic logs, trigger model evaluation, and view the results via visual dashboards. The highest-performing model is automatically highlighted based on F1-score, and the interface provides export options for report generation. This interactive design ensures transparency in model behavior and facilitates practical deployment in security-focused environments.

The outcomes of this implementation, including performance metrics and system architecture, are further discussed in the subsequent section.

5 Results & Discussion

In the NetGuardAI system, five machine learning models were evaluated based on their performance in detecting anomalies in network traffic, particularly DDoS attacks. The Table 1 given below, summaries the information about the different models tested on a URL dataset.

Among the models, Naive Bayes demonstrated the best performance with an accuracy of 76%, recall of 92%, and F1-score of 0.71, indicating that it was able to correctly identify most of the DDoS attacks, even though it had slightly lower precision (57%) due to some false alarms.

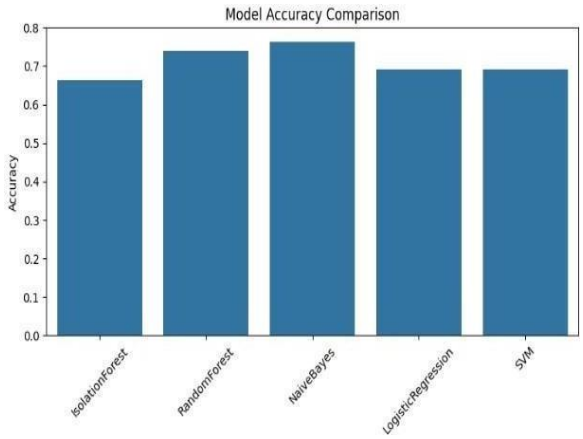
Random Forest also performed well, achieving a 74% accuracy and offering a balanced trade-off between precision (58%) and recall (54%), making it a reliable choice for balanced threat detection.

In contrast, Isolation Forest, Logistic Regression, and SVM showed poor results with zero precision and recall, suggesting they failed to detect any DDoS traffic in the given dataset. These results highlight that Naive Bayes is the most suitable model for scenarios where maximizing attack detection is critical, while Random Forest is preferable when maintaining balance between false positives and detection rate is important.

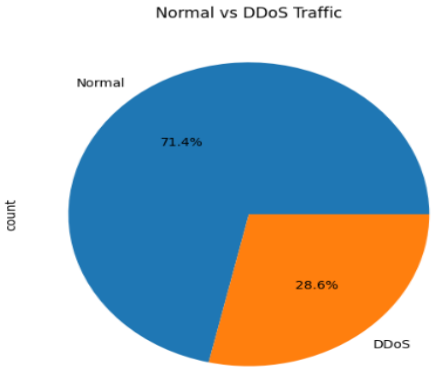
Model No.	Model	Accuracy	Precision	Recall	F1 Score	Model	Precision
1	Isolation Forest	0.66	0.00	0.00	0.00	Isolation Forest	0.00
2	Random Forest	0.74	0.58	0.54	0.56	Random Forest	0.58
3	Naive Bayes	0.76	0.57	0.92	0.71	Naive Bayes	0.57
4	Logistic Regression	0.69	0.00	0.00	0.00	Logistic Regression	0.00
5	SVM	0.69	0.00	0.00	0.00	SVM	0.00

Table 1: Summary of Classification Models Tested on the URL Dataset

In this application, there are multiple visualizations to help users interpret model performance and network behavior effectively. The figure 1 given below represents a bar chart ((a)) titled Model Accuracy Comparison, which displays the accuracy of five machine learning models—Isolation Forest, Random Forest, Naive Bayes, Logistic Regression, and SVM—helping to visually compare which model performs best. And the second chart ((b)) is a pie chart showing the percentage of normal versus DDoS traffic in the dataset. It reveals the data distribution, highlighting that about 71.4% of the traffic is normal while 28.6% is malicious, which is useful for understanding class imbalance.



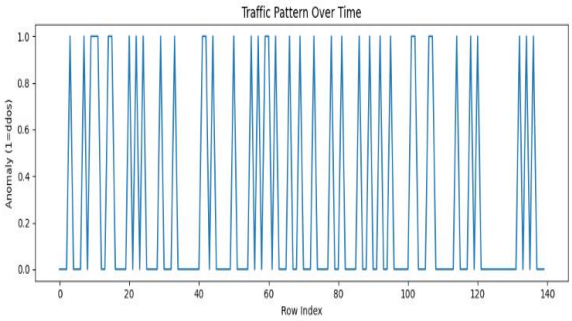
((a))Visualization of Model Accuracy Comparison



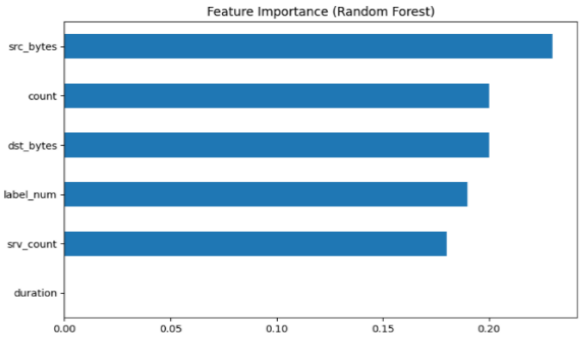
((b))Representation of Normal Vs DDOS Traffic

Figure 1: Comparison of Model and Traffic

The figure 2 shown below gives the information about the traffic pattern and the feature importance. The display of Traffic Pattern Over Time ((a)), is a time-series line plot that maps the sequence of predicted anomalies (1 for DDoS, 0 for normal) across data entries. This helps detect bursts or irregular patterns in traffic, which can indicate ongoing attacks. The visual output of feature importance chart ((b)), which is generated using the Random Forest model ranks features like src_bytes, count, and dst_bytes based on their influence in making predictions. This chart helps explain model decisions and provides insights into which traffic characteristics are most indicative of anomalies. Together, these graphs enhance the interpretability, transparency, and effectiveness of the ML-based detection system.



((a)) Illustration of Traffic Pattern Over Time



((b)) Display of Feature Importance

6 Conclusions

NetGuardAI is a web-based application designed to detect anomalies in network traffic, especially DDoS attacks, using multiple machine learning models. It supports both uploading real datasets and generating synthetic data, making it versatile and user-friendly. The system uses models like Isolation Forest, Random Forest, Naive Bayes, Logistic Regression, and SVM to analyze traffic and displays their performance using visual outputs such as bar charts, pie charts, time-series plots, and feature importance graphs. Among all models, Naive Bayes achieved the best detection results with the highest recall and F1-score, making it ideal for identifying attacks. Random Forest also performed well, offering balanced accuracy and explainability. The app includes features to export results and navigate easily through different sections, enhancing usability. Overall, NetGuardAI demonstrates the effective use of ML in cybersecurity and provides a foundation for future upgrades like real-time traffic monitoring and deployment in live environments.

Acknowledgments

We would like to express our heartfelt gratitude and appreciation to Intel© Corporation for providing an opportunity to this project. First and foremost, we would like to extend our sincere thanks to our team mentor Ms.Akshara Sasidharan for his invaluable guidance and constant support throughout the project. We are deeply indebted to our college Saintgits College of Engineering and Technology for providing us with the necessary resources, and sessions on machine learning. We extend our gratitude to all the researchers, scholars, and experts in the field of machine learning and natural language processing and artificial intelligence, whose seminal work has

paved the way for our project. We acknowledge the mentors, institutional heads, and industrial mentors for their invaluable guidance and support in completing this industrial training under Intel©-Unnati Programme whose expertise and encouragement have been instrumental in shaping our work.

References

[1] Canadian Institute for Cybersecurity. *CICIDS 2017 Dataset*. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[2] *Scikit-learn: Machine Learning in Python*. Available: https://scikit-learn.org/

[3] N. Bhatia and V. Sharma, "ML-Based Intrusion Detection Systems," *IEEE Journal*, 2022.

[4] *Hands-On Machine Learning with Scikit-Learn, TensorFlow, and Keras*, 2nd ed., O'Reilly Media, 2019- Géron.

[5] *ML for Anomaly Detection*. Available: https://towardsdatascience.com/ml-for-anomaly-detection

[6] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, O'Reilly Media, 2009.

[7] *Flask Documentation*. Available: https://flask.palletsprojects.com/

[8] Matplotlib Development Team. *Matplotlib: Python plotting*. Available at: https://matplotlib.org/

A. Main code sections for the solution

A.1 Environment Setup and Dependencies

All the development was done in VS Code on a local machine. The Python environment was set up using pip with dependencies specified in requirements.txt:

```
pip install -r requirements.txt
```

Main Libraries Used:

- pandas - For loading, manipulating, and preprocessing network traffic data (.csv files).
- numpy - For generating random numbers in synthetic datasets and handling numerical arrays.
- seaborn - For styling and enhancing the appearance of charts and plots.
- scikit-learn - For implementing machine learning models (Isolation Forest, Random Forest, SVM, etc.), splitting data, and calculating metrics like accuracy, precision, recall, and F1-score.
- matplotlib - For plotting charts such as model comparison (bar chart), pie chart, time-series plot, and feature importance.
- flask - To build the web application interface that supports routing, file upload, and rendering pages.

A.2 Loading the Dataset

The dataset is loaded to the frame pandas. And for consistency in model training the strings are mapped to the integers.

```
import pandas as pd

# Load the uploaded or generated dataset
df = pd.read_csv('data/synthetic_network.csv')

# Map string labels to integers (for consistency in model training)
df['label'] = df['label'].map({
   'normal': 0,
   'ddos': 1
}).astype(int)
```

A.3 Python function for visualization

The generate_charts() function creates and saves visual representations of model performance and traffic distribution, which help interpret results effectively.

```
import matplotlib.pyplot as plt
import seaborn as sns

def generate_charts(results, df):
   plt.figure(figsize=(8, 5))
   sns.barplot(x=list(results.keys()), y=[r['accuracy'] for r in results.values()])
   plt.title("Model Accuracy Comparison")
   plt.ylabel("Accuracy")
   plt.ylim(0, 1)
   plt.savefig('static/images/bar_chart.png')
   plt.close(
   label_counts = df['label'].value_counts()
   labels = ['Normal', 'DDoS']
   sizes = [label_counts.get(0, 0), label_counts.get(1, 0)]
   plt.figure(figsize=(6, 6))
   plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
   plt.title("Traffic Type Distribution")
   plt.savefig('static/images/pie_chart.png')
   plt.close()

   plt.figure(figsize=(10, 4))
   plt.plot(df['label'].values, marker='o', linestyle='-', color='purple')
   plt.title("Traffic Labels Over Time")
   plt.xlabel("Data Row Index")
   plt.ylabel("Label (0 = Normal, 1 = DDoS)")
   plt.savefig('static/images/timeseries_plot.png')
   plt.close()
```

A.4 Data cleaning & pre-processing

This step ensures that the data is clean, consistent, and suitable for machine learning. It includes label encoding and selection of numerical features.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('data/synthetic_network.csv')  # or the uploaded file path

# Replace string labels with numeric values
df['label'] = df['label'].replace({
   'normal': 0,
   'ddos': 1
}).astype(int)

# Drop missing values if any (optional safety step)
df.dropna(inplace=True)

# Select only numerical columns for ML processing
X = df[['duration', 'src_bytes', 'dst_bytes', 'count', 'srv_count']]
y = df['label']
```

A.5 Feature Extraction Using Numerical Traffic Features

Numerical traffic features are extracted directly from the dataset to train machine learning models. These features represent important behavioral patterns in network communication that help distinguish between normal and DDoS traffic.

```
# Selecting important features from the dataset
features = ['duration', 'src_bytes', 'dst_bytes', 'count', 'srv_count']


# Extracting input features (X) and target labels (y)
X = df[features]        # Feature set for model training
y = df['label']         # 0 = normal, 1 = ddos
```

A.6 Visualization of Feature Importance

This step helps identify which network traffic features contribute the most to the model's decision-making. We use the RandomForestClassifier to extract and plot feature importance.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest model
model = RandomForestClassifier()
model.fit(X, y)

importances = model.feature_importances_
features = X.columns

plt.figure(figsize=(8, 5))
sns.barplot(x=importances, y=features)
plt.title("Feature Importance (Random Forest)")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.savefig("static/images/feature_importance.png")
plt.close()
```

A.7 Dataset Preparation

In this step, the dataset is split into training and testing subsets to train and evaluate the machine learning models effectively. We use train_test_split from scikit-learn to divide the data.

```
from sklearn.model_selection import train_test_split

# Features (X) and labels (y) have already been prepared during pre-processing
# For example:
# X = df[['duration', 'src_bytes', 'dst_bytes', 'count', 'srv_count']]
# y = df['label']

# Split the dataset into 70% training and 30% testing
```

A.8 Model Training and Evalation

This step involves training the machine learning model on the training set and evaluating its performance using accuracy, precision, recall, and F1-score on the test set.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Train the model (using Random Forest as an example)
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)

# Display results
print("Accuracy :", accuracy)
print("Precision:", precision)
print("Recall   :", recall)
print("F1 Score :", f1)
```