INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–10

Code:IUIT-2025

MACHINE LEARNING

AI-Powered Traffic Classification Model

Vasudev V, Steny Thankkam Raju, and Jalphy Reji

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: Modern networks face increasing challenges in monitoring and securing traffic due to the exponential growth of data, encrypted communication, and sophisticated cyber threats. Traditional rule-based security measures and deep packet inspection (DPI) techniques are becoming less effective in detecting and classifying threats, especially in encrypted traffic. Manual intervention in network traffic classification is inefficient, leading to delayed threat detection and security vulnerabilities. To address these issues, AI-driven solutions can analyze traffic patterns, detect anomalies, classify applications, and enhance security in real-time, ensuring adaptive and intelligent network defense. Our project proposes an AI/ML-powered system for automated network traffic analysis and threat detection. The system leverages supervised and unsupervised learning models to classify application traffic and detect malicious activities such as port scanning, malware communication, and denial of service attempts. Feature extraction from network traffic is performed using a tool such as CICFlowMeter, and models are trained using real-world datasets. The proposed solution aims to improve detection accuracy, reduce false positives, and ensure scalable, privacy-preserving security for modern networks.

Keywords: Network Traffic Classification, Threat Detection, Anomaly Detection, Packet Analysis, Machine Learning, Encryption, Cybersecurity, Supervised Learning, Unsupervised Learning, CICFlowMeter, Privacy-Preserving AI, Real-Time Detection, Network Intrusion, Network Threat Analyzer

1 Introduction

In today's digital age, the network infrastructure serves as the backbone of global communication, commerce, and information exchange. With the exponential growth in data traffic, the increased adoption of encrypted communications, and the increasing complexity of cyber attacks, securing networks has become more challenging than ever. Traditional rule-based intrusion detection systems and deep packet inspection techniques often fail to detect emerging and sophisticated threats, especially when traffic is encrypted or obfuscated. Manual monitoring and reactive defense mechanisms also lead to delays in threat identification and mitigation.

This growing need for proactive, intelligent, and scalable network security solutions has led to the integration of Artificial Intelligence (AI) and Machine Learning (ML) into network traffic analysis. Machine learning techniques can automatically learn patterns from traffic behavior, enabling real-time detection of anomalies, classification of application traffic, and identification of malicious activity with high accuracy.

The objective of this project is to develop a Network Threat Analyzer-a system that uses AI/ML techniques to classify network traffic and detect potential threats based on extracted flow-level features. By using a tool like CICFlowMeter for data extraction and implementing supervised and unsupervised learning models, the system aims to identify suspicious behavior, reduce false alarms, and ensure privacy-preserving analysis even for encrypted traffic. This solution aligns with modern cybersecurity needs by enhancing automation, accuracy, and scalability in network defense.

Latest surveys highlight the growing use of machine learning in network security, where researchers have achieved promising results using decision trees, random forests, deep learning, and ensemble techniques for detecting intrusions and classifying applications. Building on this foundation, our project aspires to deliver a lightweight, accurate, and user-friendly framework for detecting malicious activity in network traffic uploads - making cybersecurity more intelligent and accessible.

2 Libraries Used

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

```
flask
flask-cors
pandas
numpy
scikit-learn
joblib
```

3 Methodology

This project uses supervised machine learning to detect and classify malicious network traffic. The process involves multiple stages, from data acquisition to deployment of a real-time prediction system. The methodology is outlined as follows:

- **Dataset Loading:** The `UNSW-NB15` dataset is used in this project. It was developed by the Australian Centre for Cyber Security and contains realistic modern network traffic generated using IXIA PerfectStorm. The dataset includes a wide range of features (over 40) and is labeled to identify both benign traffic and various categories of malicious activity such as Exploits, Fuzzers, DoS, Backdoors, Reconnaissance, and Worms. The training and testing data are loaded into Python using `pandas`.
- **Feature Selection:** From the full set of attributes, only a carefully chosen subset of relevant features is retained for classification. This step is guided by domain knowledge and correlation analysis to improve performance and reduce overfitting.

- **Preprocessing and Cleaning:** Data cleaning is applied only on the selected features. Missing values are handled appropriately. Categorical variables like protocol types are encoded numerically, and all numeric features are scaled using standardization techniques.
- **Dataset Splitting:** The processed data is split into training and testing subsets. This ensures that the model is evaluated on unseen data and prevents data leakage.
- **Model Training:** A `Random Forest Classifier` from the `scikit-learn` library is trained on the training set. The model is chosen for its ability to handle complex, high-dimensional datasets effectively and for its resistance to overfitting.
- **Model Saving:** After training, the model is serialized and saved using `joblib`, making it ready for deployment.
- **Web Deployment:** A web interface is developed using the `Flask` framework. It accepts network traffic data in CSV format, applies the same feature extraction pipeline, and uses the trained model to make real-time predictions, which are returned to the user via the browser.

4 Implementation

As the first step in implementation, the `UNSW-NB15` training and testing datasets are loaded using `pandas` into `DataFrames`. These datasets are publicly available and contain detailed network flow features, along with labels for various types of attacks and benign traffic.
GitHub Repository: https://github.com/Steny005/network-threat-classifier.

The feature extraction phase is handled through a custom-defined function located in the `utils/feature_extraction.py` script. This function selects only a subset of critical flow-level features based on domain knowledge and statistical relevance. The selected features are then cleaned, missing values are handled, categorical fields like protocol types are encoded numerically, and numerical features are standardized using scaling techniques.

Once cleaned, the data is split into training and testing subsets. A `Random Forest Classifier` from `scikit-learn` is used for model training. After fitting the model, it is serialized and saved using `joblib` into the `model/rf_model.pkl` file for later use.

For deployment, a Flask-based web backend is developed in `app.py`. It loads the saved model, processes uploaded `.csv` traffic files using the same feature extraction pipeline, and performs real-time classification. The frontend is built with a basic HTML page (`templates/index.html`) that allows users to upload their traffic data. Upon submission, the backend processes the file and returns the prediction results, indicating whether the traffic is benign or malicious.

This full system allows easy, real-time detection of potential network threats using a browser-based interface and a pre-trained machine learning model.

5 Results & Discussion

Popular classical machine learning algorithms from the `Python` library `scikit-learn` were employed to build and evaluate the model for network threat classification. Training and testing were performed on the UNSW-NB15 dataset, which contains labeled traffic data including malicious flows. The Random Forest classifier was selected due to its robust performance across precision, recall, and F1-score. The classification report summarizing these metrics is shown in Table 1.

Table 1: Classification Report of Random Forest Model on UNSW-NB15 Dataset

Model No.	Model Name	Class 0 (Benign)	Class 1 (Malicious)	Macro Avg	Weighted Avg
1.	Precision	0.94	0.92	0.93	0.93
2.	Recall	0.93	0.93	0.93	0.93
3.	F1-score	0.93	0.93	0.93	0.93
4.	Support	10477	9523	20000	20000
5.	Accuracy	-	-	0.93	-

To better understand the performance of the model in classifying network flows, a confusion matrix is created and given below. This matrix gives insight into the number of correct and incorrect predictions for each class. The confusion matrix is presented in Table 2.

Table 2: Confusion Matrix for Random Forest Classifier on UNSW-NB15 Dataset

	Predicted: 0	Predicted: 1
Actual: 0	9698	779
Actual: 1	635	8888

From Table 1 and Table 2, it is evident that the model achieves strong performance with an overall accuracy of 93%. The classifier performs slightly better in identifying benign traffic (precision: 0.94) than malicious traffic (precision: 0.92), with balanced recall for both classes (0.93). The confusion matrix shows that out of 10477 benign samples, 9698 were correctly classified, and 8888 out of 9523 malicious samples were accurately detected. This demonstrates reliability and effectiveness of the model for real-time intrusion detection in network systems.

6 Conclusions

The network threat detection system developed in this project achieved effective results using classical machine learning techniques, particularly the Random Forest Classifier. By focusing on relevant feature selection and applying proper data preprocessing, the model demonstrated strong accuracy in distinguishing between benign and malicious traffic from the UNSW-NB15 dataset. Its integration into a Flask-based web application allows real-time analysis, offering a practical and lightweight solution for network security operations.

Compared to resource-intensive deep learning models, classical approaches such as Random Forest offer a balanced trade-off between performance and efficiency. These models are easier to interpret, faster to train, and require fewer computational resources, making them ideal for real-world deployments where scalability and responsiveness are critical. The project proves that with structured feature engineering and model optimization, traditional ML methods remain powerful tools in the field of intelligent cybersecurity.

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 Akshara Sasidharan for her invaluable guidance and constant support throughout the project.We are deeply indebted to our college, Saintgits College of Engineering 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] BREIMAN, L. Random forests. *Machine Learning 45*, 1 (2001), 5–32.

[2] GARCIA-TEODORO, P., DIAZ-VERDEJO, J., MACIA-FERNANDEZ, G., AND VAZQUEZ, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security 28*, 1-2 (2009), 18–28.

[3] GRINBERG, M. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 2018.

[4] MOUSTAFA, N., AND SLAY, J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference (MilCIS)* (2015), IEEE, pp. 1–6.

[5] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[6] RING, M., WUNDERLICH, S., SCHEURING, D., LANDES, D., AND HOTHO, A. A survey of network-based intrusion detection data sets. *Computers & Security 86* (2019), 147–167.

A Main code sections for the solution

The project uses a machine learning model trained and deployed locally using scikit-learn and Flask. **GitHub Repository:** https://github.com/Steny005/network-threat-classifier.

A.1 Loading data from the source

Data for this project are taken from the source: https://research.unsw.edu.au/projects/unsw-nb15-dataset. The Python code section for this stage is shown below:

```
import pandas as pd
```

```
# Define path to the UNSW-NB15 training dataset
DATA_PATH = "UNSW_NB15 Training.csv"
CHUNK_SIZE = 100000

# Load the dataset in chunks
for i, chunk in enumerate(pd.read_csv(DATA_PATH, chunksize=CHUNK_SIZE, low_memory=
                                      False)):
    print(f"[INFO] Processing chunk {i+1}...")
    # Additional processing happens here
```

A.2 Data cleaning & pre-processing

This step includes reading the dataset in chunks, identifying the label column, and cleaning column names. It also includes converting class labels into binary (0 = benign, 1 = malicious):

```
def find_label_column(columns):
    for col in ['label', 'Label', 'class', 'Class', 'attack_cat']:
        if col in columns:
            return col
    raise ValueError(f"Could not find any known label column in: {columns}")

features_list = []
labels_list = []

# Read dataset in chunks
for i, chunk in enumerate(pd.read_csv(DATA_PATH, chunksize=100000, low_memory=
                                      False)):
    chunk.columns = chunk.columns.str.strip()

    try:
        label_col = find_label_column(chunk.columns)
    except ValueError:
        continue

    try:
        X_chunk, _ = extract_features_from_df(chunk)
        y_chunk = chunk[label_col].loc[X_chunk.index]

        # Normalize labels to binary (0 = benign/normal, 1 = malicious)
        y_chunk = y_chunk.apply(lambda x: 0 if str(x).lower() in ['normal', '
                                             benign', '0'] else 1)

        features_list.append(X_chunk)
        labels_list.append(y_chunk)
    except Exception:
        continue
```

A.3 Feature Extraction

To prepare the dataset for model training, only selected features were extracted using alias mapping and protocol normalization. The following function extracts relevant features from the raw data and handles missing values, unknown protocols, and inconsistencies of the column name.

```
def extract_features_from_df(df):
    # Normalize column names
    df.columns = df.columns.str.strip()
    norm_cols = {col.strip().lower().replace(' ', '').replace('_', ''): col for
                                              col in df.columns}
    col_map = {}

    # Match actual column names with standard feature names using aliases
    for std_feature, aliases in FEATURE_ALIASES.items():
        for alias in aliases:
            norm_alias = alias.strip().lower().replace(' ', '').replace('_', '')
            if norm_alias in norm_cols:
                col_map[std_feature] = norm_cols[norm_alias]
                break

    if not col_map:
        raise ValueError("No matching features found in input")

    # Convert protocol names to numeric values
    if 'proto' in col_map:
        proto_col = col_map['proto']
        df[proto_col] = df[proto_col].astype(str).str.strip().str.lower().map(
                                                  PROTOCOL_NAME_TO_NUMBER).fillna(-
                                                  1)

    # Select and clean features
    selected = df[[col_map[f] for f in col_map]].copy()
    selected.columns = list(col_map.keys())
    selected = selected.replace([float('inf'), -float('inf')], pd.NA)
    selected = selected.apply(pd.to_numeric, errors='coerce')
    valid_idx = selected.dropna().index

    return selected.loc[valid_idx].reset_index(drop=True), None
```

Additionally, it handles missing or invalid values by replacing infinities, coercing data types to numeric, and dropping rows with NaNs. This ensures that the input data used for training is clean, consistent, and suitable for machine learning.

A.4 Dataset preparation

Splitting the dataset into training and testing sets.It helps to avoid overfitting and to accurately evaluate your model.

```
X = pd.concat(features_list, ignore_index=True)
y = pd.concat(labels_list, ignore_index=True)

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                          random_state=42)
```

A.5 Model Training& Evaluation

The model was trained using a Random Forest Classifier with 100 trees on the processed UNSW-NB15 dataset. An 80-20 train-test split was used to evaluate its performance. The model achieved accurate classification of benign and malicious traffic based on extracted features. Evaluation was done using a classification report and confusion matrix. The model is then evaluated using metrics such as accuracy, precision, recall, and F1-score. Finally, the trained model was saved using `joblib` for future use.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

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

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

y_pred = clf.predict(X_test)

# Evaluation
print("\n[INFO] Classification Report:")
print(classification_report(y_test, y_pred))

print("[INFO] Confusion Matrix:")
labels = sorted(y.unique())
cm = confusion_matrix(y_test, y_pred, labels=labels)

header = "Predicted âĘŠ" + "".join([f"{label:>12}" for label in labels])
print(f"{'Actual âĘŞ':<12}{header}")
for i, row in enumerate(cm):
    print(f"{labels[i]:<12}" + "".join([f"{val:>12}" for val in row]))
```

B Web Application: Flask-Based Interface for URL Analysis

To make the trained model accessible to users, a web application was developed using the Flask framework. Users can upload a CSV file containing network traffic data, and the app will predict whether the traffic is malicious or safe using the trained Random Forest model. The results can also be downloaded as a CSV file.

B.1 Initialize Flask App and Load Model

```
from flask import Flask, render_template, request, jsonify, send_file
import pandas as pd
import joblib
from utils.feature_extraction import extract_features_from_df
from io import BytesIO
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Load the trained model
model = joblib.load('model/rf_model.pkl')
```

B.2 Homepage Route

```
@app.route('/')
def index():
    return render_template('index.html')
```

Renders the main webpage (`index.html`) which contains the file upload UI.

B.3 Analyze Uploaded File

```
@app.route('/analyze', methods=['POST'])
def analyze():
    if 'file' not in request.files:
        return jsonify({'success': False, 'error': 'No file part in the request'})
                                                  , 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'success': False, 'error': 'No selected file'}), 400

    try:
        df = pd.read_csv(file)
        features, meta = extract_features_from_df(df)
        predictions = model.predict(features)
        meta['Prediction'] = ['Malicious' if p != 0 else 'Safe' for p in
                                                  predictions]
        return jsonify({'success': True, 'data': meta.to_dict(orient='records')})
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
```

This route accepts a CSV file, extracts features using the same preprocessing logic as the training phase, performs predictions using the model, and returns the results as JSON.

B.4 Download Predicted Results as CSV

```
@app.route('/download', methods=['POST'])
def download():
    try:
        data = request.get_json()
        records = data.get('records', [])
        if not records:
            return jsonify({'error': 'No data provided'}), 400

        df = pd.DataFrame(records)
        output = BytesIO()
        df.to_csv(output, index=False)
        output.seek(0)
        return send_file(
            output,
            mimetype='text/csv',
            as_attachment=True,
            download_name='predicted_results.csv'
        )
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

This endpoint allows users to download the prediction results as a CSV file.

B.5 Running the Application

```
if __name__ == '__main__':
    app.run(debug=True)
```

This starts the Flask app in debug mode on your local machine.