VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science (SCOPE)

CRYPTOGRAPHY REPORT

Random forest Modelling for Network

Intrusion Detection System

Team

Rohit Navaneethan (21BCE1029)

Hannah Gracelyne (21BCE1078)

Om Prakash (21BCE1950)

Faculty

Dr. Sobitha Ahila S

CODE :

```
#Importingthenecessarylibraries

fromsklearn.ensembleimportIsolationForestfromskl
earn.preprocessingimportStandardScalerimportpand
asaspd


#Loadandpreprocessthedataset

dataset=pd.read_csv('network_traffic.csv')

#Loadthedatasetfrom'network_traffic.csv'

X=dataset.iloc[:,:-
1].values#Extractthefeaturesfromthedatasetscaler=StandardScaler()

X_scaled=scaler.fit_transform(X)     #Scalethefeatures
```

```
model.fit(X_scaled)  #Fitthemodeltothescaleddata


#Performanomalydetection

predictions=model.predict(X_scaled)    #Predicttheanomaliesinthedataset (-1
for anomalies, 1 for normal instances)


# Print the detected
anomaliesanomalies=X[predictions==-
1]print("Detected Anomalies:")
```

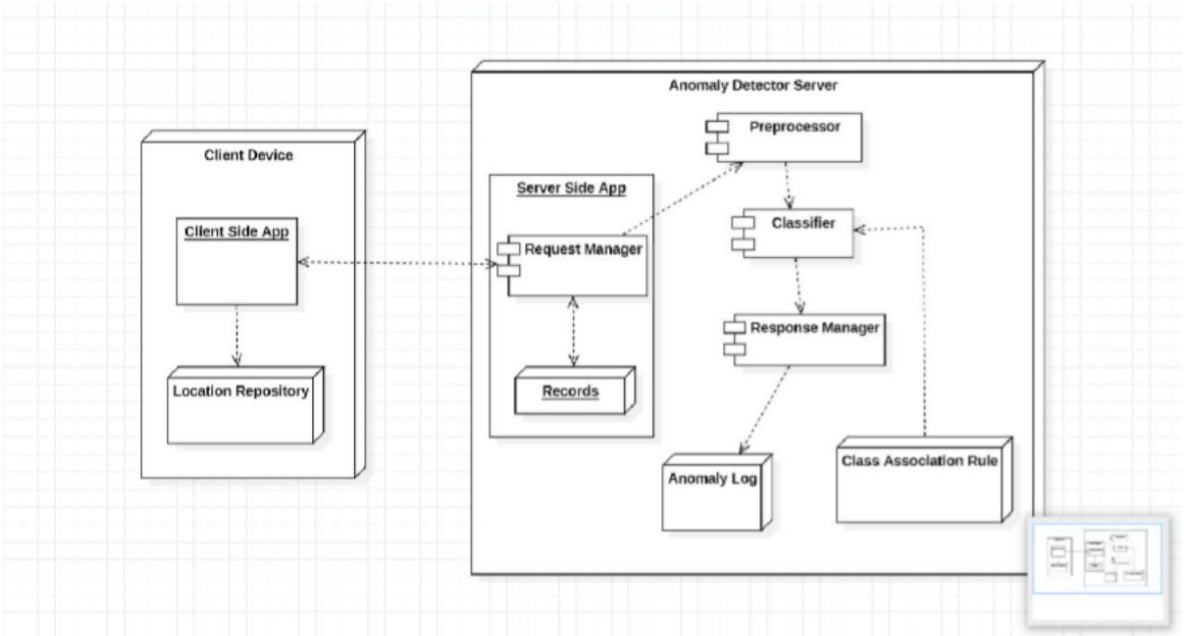
Explanation:

The above code demonstrates a basic implementation of an anomaly detection system using the Isolation Forest algorithm. Here is a breakdown of the steps involved:

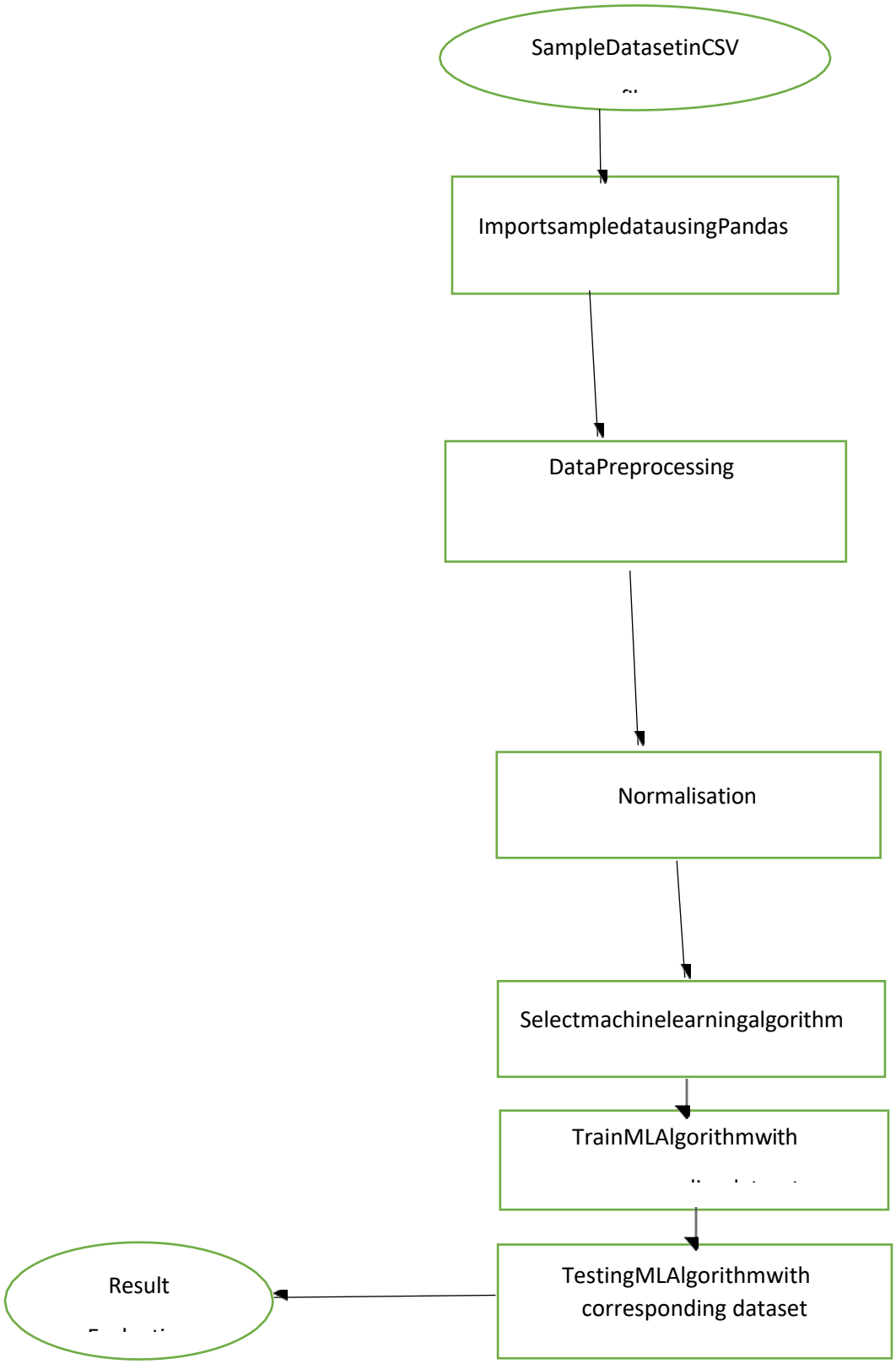
1. Import the necessary libraries, such as the IsolationForest class from the sklearn.ensemble module, and pandas for data handling.
2. Load the dataset, assuming it is stored in a CSV file named 'network_traffic.csv'.
3. Preprocess the data as per your requirements. This may include handling missing values, performing feature scaling, and transforming the data to a suitable format for the Isolation Forest algorithm.

4. Instantiate the IsolationForest class, specifying the desired contamination parameter. The contamination parameter determines the expected proportion of anomalies in the dataset.
5. Fit the Isolation Forest model to the preprocessed data using the fit() method.
6. Make predictions on the data using the predict() method of the model. The predictions will assign a label of -1 to instances deemed as anomalies, and 1 to normal instances.
7. Identify and flag the anomalies in the dataset by filtering the data based on the predicted labels (-1) and setting a new column 'is_anomaly' to True for these instances.
8. Optionally, save the updated dataset with the anomaly flags to a new CSV file, such as 'network_traffic_with_anomalies.csv'. This step allows you to analyze or further process the data with the anomaly information.

Mechanism Diagram: [Anomaly Detection System]



Flow Chart:


SampleDatasetinCSV
ImportsampledatausingPandas
DataPreprocessing
Normalisation
Selectmachinelearningalgorithm
TrainMLAlgorithmwith
TestingMLAlgorithmwith
corresponding dataset
Result

The goal of the project is to develop a machine learning-based anomaly detection system that can effectively identify unusual or abnormal patterns in data. The system aims to enhance the security and reliability of a specific application or system by detecting and alerting potential anomalies in real-time.

Project Methodology:

The project will follow the following methodology:

- **Requirement Gathering:** Identify the specific requirements and objectives of the anomaly detection system, considering the application domain, data sources, and desired system capabilities.
- **Data Collection and Preprocessing:** Gather relevant data for training and testing the machine learning models. Preprocess the data by handling missing values, normalizing, or scaling features, and performing any necessary transformations.
- **Algorithm Selection:** Research and select suitable machine learning algorithms for anomaly detection, considering factors such as the nature of the data, algorithm performance, and computational requirements.
- **Model Development and Training:** Develop the machine learning models based on the selected algorithms. Train the models using the pre-processed data, optimizing parameters and hyperparameters as necessary.
- **Real-time Monitoring Implementation:** Implement a system that continuously analyses incoming data in real-time using the trained models. Apply anomaly detection algorithms to identify anomalies and trigger appropriate actions or alerts.
- **Performance Evaluation:** Evaluate the performance of the anomaly detection system using appropriate metrics, such as precision, recall, and F1 score. Compare the results against predefined benchmarks or ground truth data.
- **Documentation and Deployment:** Prepare documentation that includes system architecture, implementation details, and user guidelines. Deploy the anomaly detection system in the target environment and provide necessary support for maintenance and updates.

Project Deliverables:

The project will deliver the following:

1. Anomaly detection system capable of real-time monitoring and detecting anomalies in the specified application or system.
2. Trained machine learning models optimized for anomaly detection.
3. Performance evaluation report including metrics, benchmarks, and analysis of system effectiveness.
4. Documentation, including system architecture, implementation details, and user guidelines.
5. Support for system deployment and maintenance, including necessary updates and bug fixes.

It is important to note that the specific project goals, scope, methodology, and deliverables may vary depending on the context and requirements of the project.

Implementation:

To perform a live implementation of a machine learning and AI-based detection system for cybersecurity, follow these steps:

1. **Data Collection:** Collect relevant and representative data for training and testing your detection system. This data should include normal and anomalous samples that reflect the real-world scenarios you want to detect.
2. **Data Preprocessing:** Preprocess the collected data by cleaning, transforming, and normalizing it as necessary. Ensure that the data is in a suitable format for training the machine learning model.
3. **Model Training:** Select an appropriate machine learning algorithm (such as Isolation Forest, Support Vector Machines, or Neural Networks) for your detection system. Split the pre-processed data into training and testing sets. Train the model using the training data, tuning hyperparameters as needed.

4. **Real-Time Data Collection:** Set up a mechanism to collect real-time data from the environment you want to monitor. This may involve accessing logs, network traffic, system events, or other relevant data sources.
5. **Real-Time Data Preprocessing:** Apply the same preprocessing steps used during training to preprocess the real-time data. This may include feature extraction, normalization, or other transformations specific to your detection system.
6. **Real-Time Detection:** Feed the pre-processed real-time data into the trained model for anomaly detection. Apply the model's prediction algorithm to classify the data as normal or anomalous in real-time.
7. **Alert Generation:** Based on the model's predictions, generate alerts or notifications when anomalous activity is detected. The alerts can be sent to a security operations centre (SOC), a monitoring dashboard, or other appropriate channels for further investigation or response.
8. **Evaluation and Feedback:** Continuously evaluate the performance of your detection system using appropriate metrics, such as precision, recall, and F1-score. Collect feedback from analysts or users to improve the system's accuracy and usability over time.
9. **Iterative Improvement:** Incorporate feedback and lessons learned to refine and enhance the detection system. This may involve retraining the model with updated data, fine-tuning algorithms, adding new features, or exploring other machine learning techniques.
10. **Maintenance and Monitoring:** Regularly monitor the performance of the detection system, maintain the models and algorithms, and update the system as needed to adapt to evolving threats and changes in the environment.

DATASET :

	SIZE	SPEED	TIME	ENERGY	LIMIT	Label
2	-1.87905	-2.48863	-0.58153	0.215178	0.49939	Normal
3	1.595852	0.121347	-0.27157	-0.52411	0.282197	Normal
4	-1.1196	-1.45843	0.756445	0.447445	0.364338	Normal
5	-0.51526	-0.3739	-0.05078	-0.79521	2.755073	Normal
6	0.242631	-0.33363	-1.15204	-0.14999	-1.12768	Normal
7	-0.62379	-1.11501	-0.51833	-1.27512	-0.48856	Normal
8	0.051627	-2.5209	-0.2808	-0.19072	2.141908	Normal
9	-0.44719	-0.81585	-0.90174	0.063768	-0.07138	Normal
10	-0.29644	1.956306	-1.08675	-0.65054	-0.08229	Normal
11	1.63807	2.129724	-0.04537	0.337234	0.394491	Normal
12	-3.16534	0.75283	-0.39429	0.482698	0.494495	Normal
13	-0.43602	-0.27229	-0.41768	1.18567	-1.07043	Normal
14	2.015031	1.231333	0.644007	1.468168	0.660927	Normal
15	0.383498	-0.48359	-0.56263	-1.6941	-0.15642	Normal
16	0.149536	0.16428	-0.88691	1.655685	0.494268	Normal
17	0.278064	0.109522	0.910903	-0.3127	0.472005	Normal
18	0.950165	-1.3721	1.24913	0.046435	-1.17969	Normal
19	-0.20919	0.462335	-1.31399	1.478526	-0.74155	Normal
20	-0.11937	-0.63732	-0.28723	0.902339	-1.1421	Normal
21	6.743759	-5.87246	-4.13624	-6.51278	-7.86016	Anomaly
22	0.505203	-1.68786	0.612288	1.638166	2.196718	Normal
23	-0.13483	-0.63423	-1.21333	1.415526	0.481339	Normal
24	-1.13402	0.976417	2.027685	1.721849	0.207019	Normal
25	-0.97355	-0.26467	0.658451	-0.2345	1.12871	Normal
26	-1.72096	0.310912	0.708026	-0.8527	1.320317	Normal
27	-0.58576	0.258454	-0.74201	0.519264	-0.83746	Normal
28	-0.77159	0.340051	1.708861	-2.09551	0.711573	Normal
29	-0.4393	-0.08117	-0.98631	1.236761	-0.11619	Normal

network_traffic(1)

```
# Importing the necessary libraries
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
# Load and preprocess the dataset
dataset = pd.read_csv('/content')  # Load the dataset from 'network_traffic.csv'
X = dataset.iloc[:, :-1].values  # Extract the features from the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Scale the features
```

```
# Train the Isolation Forest model
model = IsolationForest(contamination=0.1)  # Define the model with the desired contamination rate
model.fit(X_scaled)  # Fit the model to the scaled data
```

```
IsolationForest(contamination=0.1)
```

```
# Perform anomaly detection
predictions = model.predict(X_scaled)  # Predict the anomalies in the dataset (-1 for anomalies, 1 for normal instances)
```

```
# Print the detected anomalies
anomalies = X[predictions == -1]
print("Detected Anomalies:")
for anomaly in anomalies:
    print(anomaly)
```

```
[ 0.18854336  0.85903196 -2.70318336 -1.8603171  -3.0895287 ]
[-3.82347162 -2.37214315  8.99358445  6.60820832 -5.1873971 ]
[-6.54395644 -3.58464389  8.66144936 -3.4307337  -6.96263864]
[1.45415496 6.95219127 2.36932321 4.65206604 7.93137024]
[ 4.19786238 -9.90427125  6.0514697   3.30152076 -0.21189696]
[-4.43860218 -8.4753859  -5.72323375  5.68661592 -6.68955617]
[ 2.02191327 -3.27916165 -2.64420956  9.90205984  3.22409234]
```

IDS :

Introduction:

In the contemporary landscape of network security, characterized by the exponential growth of network-based services and the proliferation of sensitive information traversing digital highways, the imperative of safeguarding network integrity has never been more pressing. Despite the deployment of a myriad of security measures such as information encryption, access control, and intrusion prevention, the persistent threat of undetected intrusions looms large, necessitating robust and adaptive defense mechanisms. In this context, intrusion detection systems (IDS) emerge as indispensable guardians, tasked with the automated monitoring of network activities to swiftly identify and mitigate potential threats. However, the efficacy of traditional rule-based IDSs has been marred by inherent limitations in scalability, adaptability, and susceptibility to false alarms. As a response to these challenges, there is a discernible shift towards the integration of data mining techniques within IDS architectures, aiming to enhance detection accuracy, flexibility, and responsiveness to evolving threats. The emergence of data mining-powered IDSs heralds a paradigm shift in intrusion detection methodologies, moving away from manual rule encoding towards the automated extraction of actionable insights from vast volumes of network data. Among the arsenal of data mining algorithms, the random forests algorithm stands out as a promising candidate for revolutionizing intrusion detection due to its ensemble classification and regression capabilities, which enable the construction of robust decision trees for accurate and stable intrusion classification. By harnessing the power of random forests, IDSs can effectively discern between normal network traffic and potentially malicious activity, thereby bolstering network security defenses against a myriad of cyber threats. This work seeks to explore the transformative potential of integrating random forests within IDS frameworks, offering novel insights into its application across misuse, anomaly, and hybrid detection scenarios. Through empirical analysis and experimentation, we aim to demonstrate the efficacy and versatility of random forests in fortifying network defenses, paving the way for a new era of proactive and adaptive intrusion detection strategies in the realm of cybersecurity.

About the Dataset:

In our pursuit to advance the field of intrusion detection, we introduce a novel approach that distinguishes itself through the utilization of a meticulously crafted, proprietary dataset. Unlike conventional studies that rely on publicly available datasets, we recognize the inherent limitations of such datasets in accurately representing the diverse network environments and threat landscapes encountered in real-world scenarios. Therefore, we have meticulously designed and curated a comprehensive dataset encompassing network information, system details, and process data, tailored to mirror the complexities of contemporary network infrastructures. This bespoke dataset not only captures the nuances of normal network behavior but also encompasses a diverse range of intrusion scenarios, ensuring robustness and relevance in our experimentation. By leveraging our custom dataset, we aim to push the boundaries of intrusion detection research, providing insights into the efficacy and adaptability of random forests in discerning anomalies and malicious activities across various network contexts. Through our innovative dataset-driven approach, we strive to enhance the accuracy, reliability, and real-world applicability of intrusion detection systems, contributing to the advancement of cybersecurity practices in an ever-evolving digital landscape.

Data Pre-Processing Steps:

Data Collection: Gather network traffic data, system information, and process data from various sources, ensuring the dataset is representative of diverse network environments and potential intrusion scenarios.

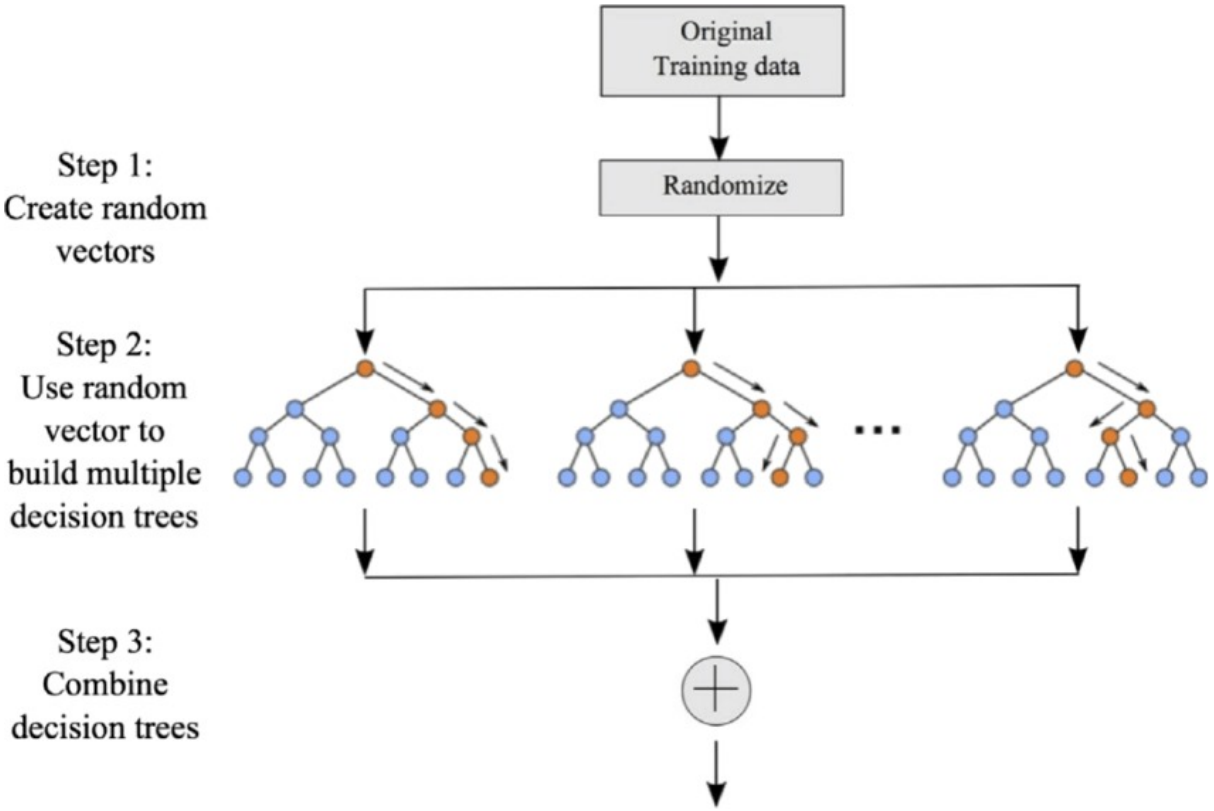
Data Cleaning: Remove any duplicate or irrelevant entries from the dataset to ensure data quality. Handle missing values appropriately by imputing them or removing them based on the nature of the data.

Data Splitting: Divide the dataset into training and testing sets to facilitate model training, hyperparameter tuning, and performance evaluation. Ensure that each set maintains the same class distribution to prevent bias.

Normalization/Standardization: Scale the features to a common range to prevent certain features from dominating others during model training. This step is particularly crucial for algorithms like Random Forest, which are sensitive to feature scales.

Data Encoding: Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding, ensuring compatibility with the Random Forest algorithm.

Architecture:



Original Training Data: The process begins with a dataset containing features extracted from network traffic, system information, and process data. This dataset comprises labeled instances of normal network behavior and known intrusions, serving as the original training data for the Random Forest model.

Creation of Random Vectors: Random vectors, also known as random subsets or feature subsets, are generated from the original dataset. These random vectors are created by randomly selecting a subset of features from the original feature set. The random selection process ensures diversity among the decision trees in the ensemble, reducing correlation between them.

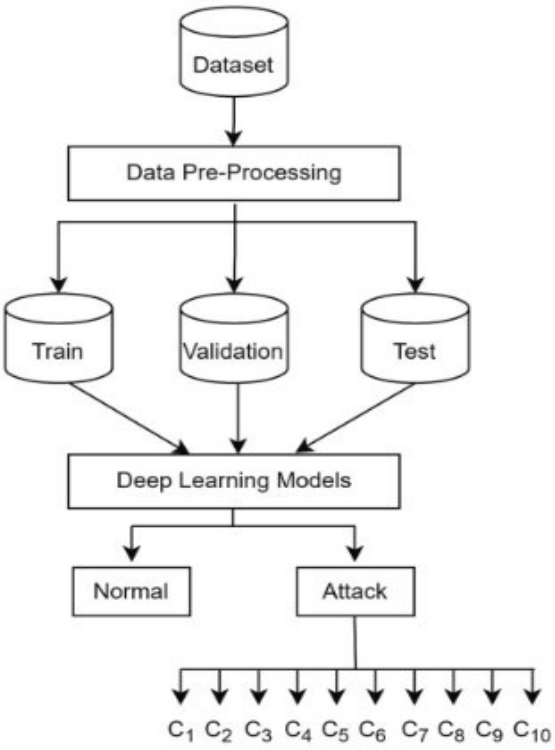
Building Multiple Decision Trees: Using each random vector, multiple decision trees are constructed independently. These decision trees are typically built using algorithms such as CART (Classification and Regression Trees) or ID3 (Iterative Dichotomiser 3). Each decision tree is trained to classify instances based on the features contained in its corresponding random vector.

Combining Decision Trees: Once all decision trees are built, they are combined to form the Random Forest ensemble. The combination process involves aggregating the predictions of individual decision trees to make a final classification decision. In the context of intrusion detection, the ensemble's decision is typically determined by majority voting, where the class with the most votes among all decision trees is selected as the final prediction.

Evaluation and Testing: The trained Random Forest model is evaluated and tested using a separate validation or testing dataset. Performance metrics such as accuracy, precision, recall, and F1-score are calculated to assess the model's effectiveness in detecting intrusions while minimizing false positives.

Deployment and Monitoring: Once validated, the Random Forest model can be deployed in a production environment for real-time intrusion detection. It continuously monitors network traffic and system activities, flagging any suspicious behavior indicative of potential intrusions. Regular monitoring and periodic updates to the model ensure its efficacy in adapting to evolving cybersecurity threats.

FLOWCHART:



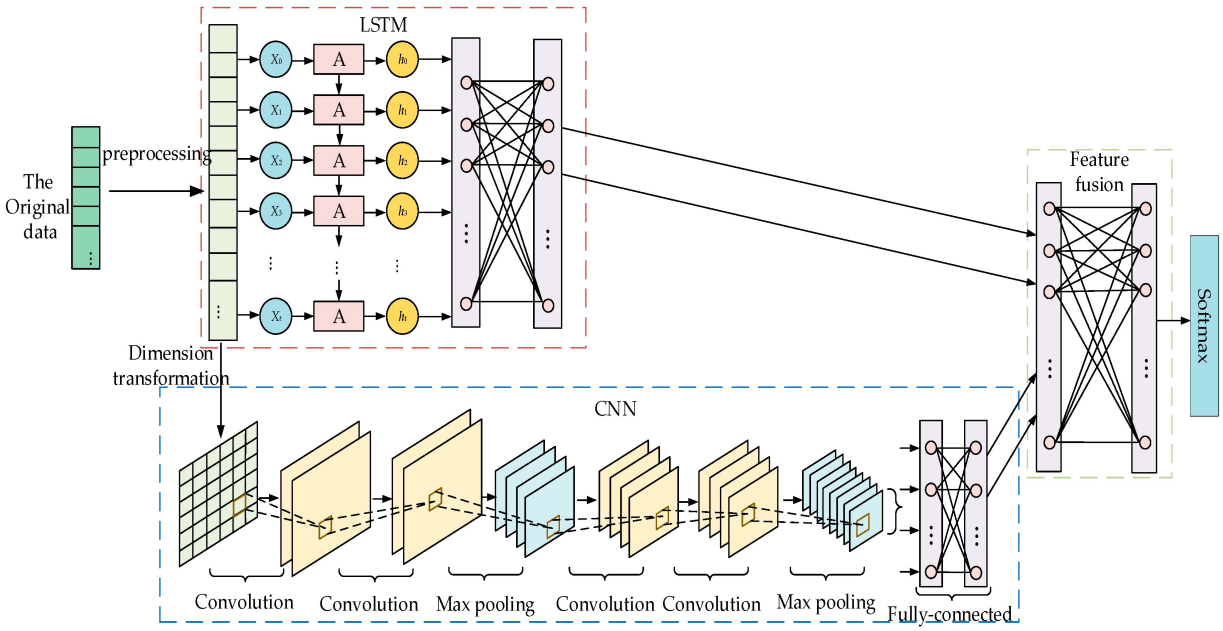
Validation of Data: A portion of the dataset is reserved for validation during the training phase. The validation set is used to fine-tune the model's parameters and prevent overfitting. This ensures the model generalizes well to unseen data.

Test the Data: The remaining portion of the dataset, known as the testing set, is employed to assess the model's performance on unseen data. This phase provides an unbiased evaluation of the model's ability to accurately classify instances of normal and attack behaviors.

Apply Deep Learning Model: The deep learning models, specifically CNN and LSTM classifiers, are generated using the normalized dataset. These models are designed to capture intricate patterns and dependencies within the data, with CNN focusing on spatial features and LSTM addressing temporal aspects.

Classify Intrusion into Normal or Attack: In the evaluation phase, the proposed Intrusion Detection System (IDS) based on deep learning methods is assessed. The models are evaluated using various metrics such as accuracy, recall, precision, loss, and F1-Score for both binary and multi-class classifications. The IDS classifies instances into normal or attack based on the patterns and features learned during the training phase, providing a comprehensive assessment of its performance in detecting intrusions.

MODEL ARCHITECTURE



Convolutional Neural Networks (CNN):

Input Layer: Takes in the raw input data.

Convolutional Layer: Extracts features from the data using multiple convolution kernels, each with weights and biases. The ReLU activation function is applied for non-linearity.

Pooling Layer: Down samples the data through max pooling, retaining critical information while reducing complexity.

Fully Connected (FC) Layer: Acts as a "classifier," weighting features from previous layers and remapping them to a sample-marker space. Dropout is employed to prevent overfitting.

Output Layer: Produces the final classification output.

Convolutional Layer Operation

Equation : $Xi=f(wi \otimes Xi-1+bi)$**,**

Xi represents the output result of convolution kernel i.

⊗ represents the convolution operation.

f(x) represents the activation function (ReLU - Rectified linear unit is used).

Explanation:

Convolutional layer extracts characteristic information by sweeping the input data.

ReLU activation function prevents gradient disappearance and speeds up model training.

Pooling Layer Operation

Objective: Describe the purpose of the pooling layer in the CNN.

Equation : $Qj=Max(P0j,P1j,P2j,P3j...Ptj)$

Qj represents the output result of the pooling region j.

Max pooling operation is used.

Explanation:

Pooling layer achieves invariance and reduces complexity by down sampling.

Max pooling retains critical information for better performance.

Fully Connected (FC) Layer

Objective: Explain the role of FC layers in the CNN.

Operation:

FC layers act as "classifiers" in the entire CNN.

Dropout operation is implemented to prevent overfitting.

Significance:

Weights features of convolutional and pooling layers, remapping them to the sample-marker space.

Long Short-Term Memory Networks (LSTM) Components:

Key Components:

Forget gate: Determines how much information is forgotten.

Input gate: Manages the update status of information.

Output gate: Determines the final output based on the current cell state.

Functionality:

LSTM solves the issues of gradient explosion or disappearance in traditional RNNs.

Long-term memory and short-term memory are activated through gate structures.

Feature Fusion Component

Structure:

Full connections between layers with activation functions for nonlinearity.

MLP composed of an input layer, hidden layers, and an output layer.

Operation:

Features extracted by CNN and LSTM are combined into comprehensive features through flattening and contact operations.

MLP performs nonlinear mapping, and the output layer predicts classification results.

Backpropagation and Parameter Updating:

Soft-max classification model is used for identifying intrusion behaviors.

Importing Dataset: In the first step, dataset is imported for use in the intrusion detection system. These datasets likely contain information on network traffic, potentially including both normal and attack instances.

Data Preprocessin: Data cleaning is applied to the dataset during the data preprocessing phase. This step involves handling missing or inconsistent data to ensure the datasets are ready for analysis. Following data cleaning, normalization is performed using the min–max method to standardize the data and bring it into a consistent range.

Training of Data: After preprocessing, the datasets are divided into training, validation, and testing sets. The training set is utilized to train the deep learning models, such as CNN and LSTM classifiers. This involves exposing the model to labeled data, allowing it to learn patterns and relationships within the features and labels.

**** These are the 3 files run client and analyser file in multiple computer and connect each computer on the same network. We can achieve NIDS once we get the data then just I will have to apply ML model . ****

CODE :

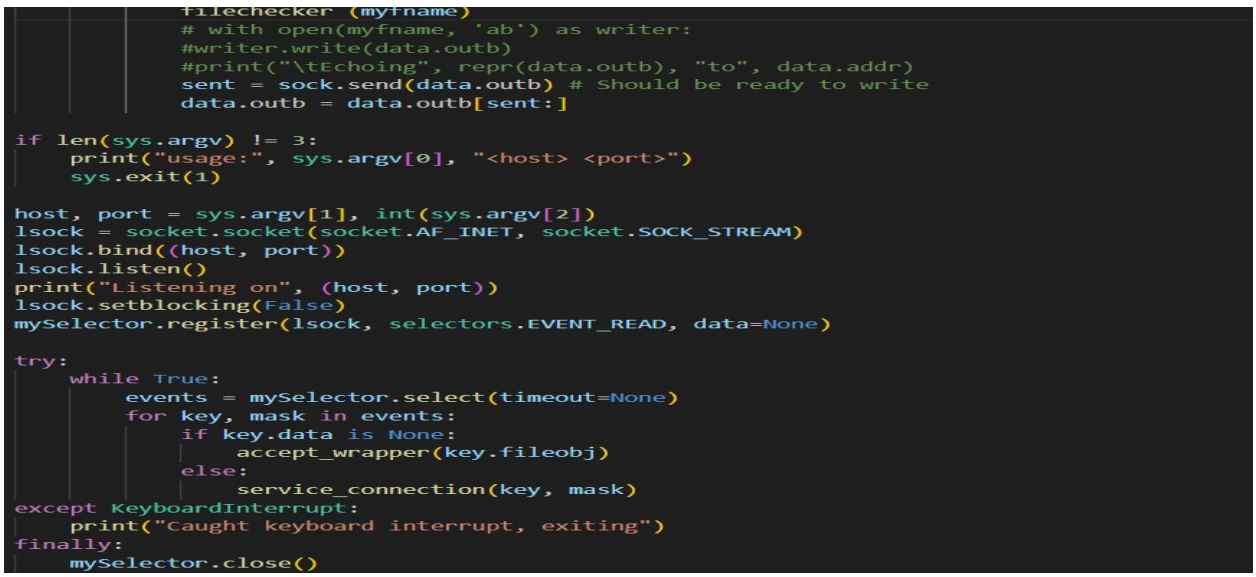
server_ids.py

```
import sys

import socket

import selectors

import types

import datetime

import os

mySelector = selectors.DefaultSelector()


def filechecker(myfilename):

    #read input file

    fin = open(myfilename, "rt")
```

```
    #read file contents to string
    data = fin.read()
    #replace all occurrences of the required string
    data = data.replace('\n\n', '\n')
    #close the input file
    fin.close()
    #open the input file in write mode
    fin = open(myfilename, "wt")
    #overwrite the input file with the resulting data
    fin.write(data)
    #close the file
    fin.close()


def accept_wrapper(sock):
    conn, addr = sock.accept() # should be ready to read
    print("========\nAt ", datetime.datetime.now(), "Accepted connection from", addr,
"\n======")
    print("\U0001f600" *5)
    conn.setblocking(False)
    data = types.SimpleNamespace(addr=addr, inb=b"", outb=b"")
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    mySelector.register(conn, events, data=data)


def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(20480) # Should be ready to read
        if recv_data:
            data.outb += recv_data
```

```
        else:
            print("=======\n AT ", datetime.datetime.now(), " Closed connection to", data.addr,
"\n======")
            mySelector.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if data.outb:
            mySize= sys.getsizeof(data.outb)
            print(mySize, " bytes recieved from ", data.addr[0], " on port ", data.addr[1])
            #print(str(sys.getsizeof(data.outb)) + " Bytes Recieved")
            myfname = ''.join(data.addr[0]).encode()
            myfname = str(myfname)
            myfname += str(data.addr[1])
            myfname += '.xyz'
            f = open(myfname, "ab")
            f.write(((data.outb)))
            #f.write(''.join(repr(data.outb)))
            f.close()
            filechecker (myfname)
            # with open(myfname, 'ab') as writer:
            #writer.write(data.outb)
            #print("\tEchoing", repr(data.outb), "to", data.addr)
            sent = sock.send(data.outb) # Should be ready to write
            data.outb = data.outb[sent:]

if len(sys.argv) != 3:
    print("usage:", sys.argv[0], "<host> <port>")
    sys.exit(1)

host, port = sys.argv[1], int(sys.argv[2])
```

```
lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
lsock.bind((host, port))
lsock.listen()
print("Listening on", (host, port))
lsock.setblocking(False)
mySelector.register(lsock, selectors.EVENT_READ, data=None)


try:
    while True:
        events = mySelector.select(timeout=None)
        for key, mask in events:
            if key.data is None:
                accept_wrapper(key.fileobj)
            else:
                service_connection(key, mask)
except KeyboardInterrupt:
    print("Caught keyboard interrupt, exiting")
finally:
    mySelector.close()
```



client_ids.py

```
import os
import platform
import sys
import socket
import selectors
import types
import psutil
import re
import uuid


sel = selectors.DefaultSelector()
messages = [b"Message 1 from client.", b"Message 2 from client.", b"message 3."]
mySysOS = "NIL"
# global myMessages
# myMessoges = [b"-", b"-", b"-"]


def getProcess():
    output = "PROCESSID, PROCESSNAME, STATUS, STARTTIME\n"
    for proc in psutil.process_iter():
        try:
            # Get process name & pid from process object.
            #print(proc)
            #print(str(proc.pid), ", ",str(proc.name()), str(proc.status()), str(proc.create_timeO)))
            #output += str(proc.pid), proc.nameO), proc.status(), str(proc.create_time())
            output += str(proc.pid) + "," + str(proc.name()) + "," + str(proc.status()) + "," +
str(proc.create_time()) + "\n"
        except (psutil.Nosuchprocess, psutil.AccessDenied, psutil.zombieprocess):
            pass
    return output
```

```
def getConnection():
    output = "PROCESSID, STATUS, LOCALIP, LOCALPORT, REMOTEIP, REMOTEPORT\n"
    for cons in psutil.net_connections(kind='inet'):
        try:
            output += str(cons.pid) + "," + str(cons.status) + "," + str(cons.laddr.ip) + "," +
str(cons.laddr.port) + ',' + (((str(cons.raddr).replace("()", ",,").replace("()", ",,")))) + "\n"
        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
            pass
    return output


def getSystem():
    output = "NAME, DETAILS\n"
    output += "platform,"+ str(platform.system()) + "\n"
    output += "platform-release, "+ platform.release() + "\n"
    output += "platform-version, "+ platform.version() + "\n"
    output += "architecture, "+ platform.machine() + "\n"
    output += "hostname, "+ socket.gethostname() + "\n"
    output += "ip-address,"+ socket.gethostbyname(socket.gethostname()) + "\n"
    output += "mac-address " + ':'.join(re.findall(' .. ', '%012x' % uuid.getnode())) + "\n"


    output += "processor, "+ (platform.processor()).replace("," " ") + "\n"
    output += "ram, "+ str(round(psutil.virtual_memory().total / (1024.0 ** 3)))+" GB"
    return output


def getWinSystem():
    #output = os.popen('wmic computersystem 1ist Name, UserName, Manufacturer, Model,
TotalPhysicalMemory /Format:Textvaluelist').read()
    output = os.popen('wmic computersystem list full /format:Textvaluelist').read()
    # print(output)
```

```
    return output


def getWinProcess():
    output = os.popen('wmic process get description, processid /format:csv').read()
    # print (output)
    return output


def getWinConnection():
    output = os.popen('netstat -an').read()
    # print (output)
    return output


def os_info():
    return platform.system()


def cpu_info():
    if platform.system() == 'Windows':
        return platform.processor()
    elif platform.system() == 'Darwin':
        command = '/usr/sbin/sysctl -n machdep.cpu.brand_string'
        return os.popen(command).read().strip()
    elif platform.system() == 'Linux':
        command = 'cat /proc/cpuinfo'
        return os.popen(command).read().strip()
    return 'platform not identified'


def start_connections(host, port, num_conns, sysData):
    server_addr = (host, port)
    for i in range(0, num_conns):
        connid = i + 1
```

```
        print("starting connection", connid, "to", server_addr)
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setblocking(False)
        sock.connect_ex(server_addr)
        events = selectors.EVENT_READ | selectors.EVENT_WRITE
        data = types.SimpleNamespace(
            connid=connid,
            msg_total=sum(len(m) for m in sysData),
            recv_total=0,
            messages=list(sysData),
            outb=b"",
        )
        sel.register(sock, events, data=data)


def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(10240)  # Should be ready to read
        if recv_data:
            print("SERVER Recieved [", sys.getsizeof(recv_data), "] Bytes For Connection",
data.connid)
            data.recv_total += len(recv_data)
        if not recv_data or data.recv_total == data.msg_total:
            print("SERVER Closing Connection", data.connid)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if not data.outb and data.messages:
            data.outb = data.messages.pop(0)
```

```
        if data.outb:
            print("SERVER Sending [", sys.getsizeof(repr(data.outb)), "] Bytes To Connection",
data.connid)
            sent = sock.send(data.outb) # Should be ready to write
            data.outb = data.outb[sent:]


if len(sys.argv) != 3:
    print("usage:", sys.argv[0], "<host> <port> <num_connections>")
    sys.exit(1)


host, port = sys.argv[1:3]
host = socket.gethostbyname(host)
hostname = socket.gethostname()
myMessages=[ ("SYSINFO, "+hostname+", "+str(host)+", "+str(port)+"~`~`~`~`~`\n").encode(),
(getSystem()).encode()]
start_connections(host, int(port), 1, myMessages)
myMessages=[ ("PROINFO, "+hostname+", "+str(host)+", "+str(port)+"~`~`~`~`~`\n").encode(),
(getProcess()).encode()]
start_connections(host, int(port), 1, myMessages)
myMessages=[ ("NETINFO, "+hostname+", "+str(host)+", "+str(port)+"~`~`~`~`~`\n").encode(),
(getConnection()). encode()]
start_connections(host, int(port), 1, myMessages)


print(cpu_info())


try:
    while True:
        events = sel.select(timeout=1)
        if events:
            for key, mask in events:
                service_connection(key, mask)
```

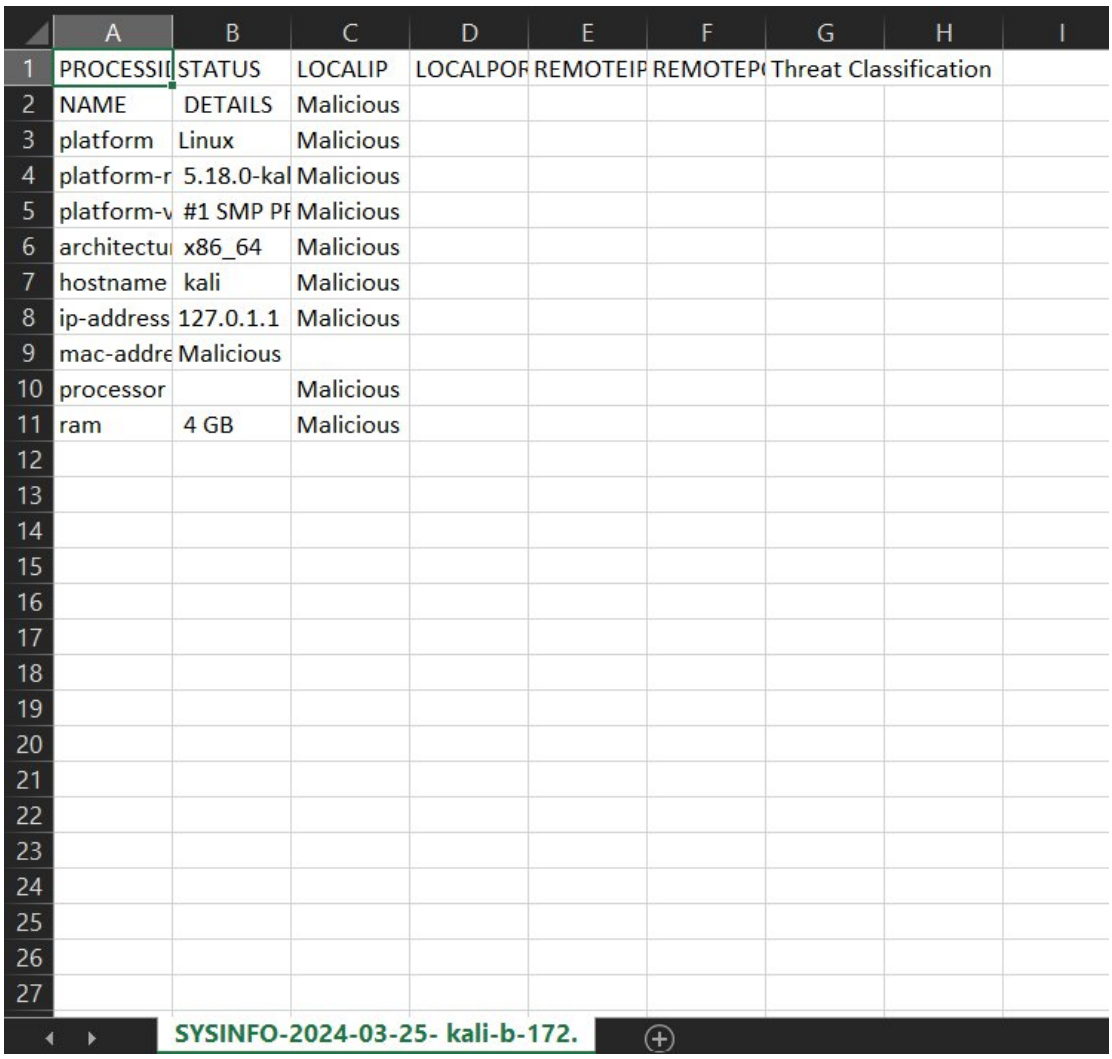
```
        # Check for a socket being monitored to continue.
        if not sel.get_map():
            break
except KeyboardInterrupt:
    print("caught keyboard interrupt, exiting")
finally:
    sel. close()
```

analysis_ids.py

```
import os
import datetime

myPath = os.getcwd()
files = os.listdir(myPath)

def classify_threat(line):
    # Define threat analysis rules here
    if "ESTABLISHED" in line:
        return "Highly Malicious"
    elif "NONE" in line:
        return "Not Malicious"
    else:
        return "Malicious"

for f in files:
    if f.endswith("xyz"):
        print(f)
        fx = open(f)
        lines = fx.readlines()
        fx.close()

        data = lines[0].split(",")
        mytype = data[0]
        myname = data[1]

        print(mytype + "--" + myname)
        mynewfilename = (
            mytype
```

```
        + "-"
        + str(datetime.date.today())
        + "-"
        + myname
        + "-"
        + (f.replace("'", "-")).replace(".xyz", ".csv")
    )
    new_file = open(mynewfilename, "w")

new_file.write("PROCESSID,STATUS,LOCALIP,LOCALPORT,REMOTEIP,REMOTEPORT,Thr
eat Classification\n")


    for i, line in enumerate(lines):
        if i == 0:
            print(line)
        else:
            # Append the threat classification to each line
            threat_classification = classify_threat(line)
            new_file.write(f"{line.strip()},{threat_classification}\n")
            print(i, line.strip(), threat_classification)
        i = i + 1


    new_file.close()
    os.remove(f)
    print(" ")
```

```
        data = lines[0].split(",")
        mytype = data[0]
        myname = data[1]

        print(mytype + "--" + myname)
        mynewfilename = (
            mytype
            + "-"
            + str(datetime.date.today())
            + "-"
            + myname
            + "-"
            + (f.replace("'", "-")).replace(".xyz", ".csv")
        )
        new_file = open(mynewfilename, "w")
        new_file.write("PROCESSID,STATUS,LOCALIP,LOCALPORT,REMOTEIP,REMOTEPORT,Threat Classification\n")

        for i, line in enumerate(lines):
            if i == 0:
                print(line)
            else:
                # Append the threat classification to each line
                threat_classification = classify_threat(line)
                new_file.write(f"{line.strip()},{threat_classification}\n")
                print(i, line.strip(), threat_classification)
            i = i + 1

        new_file.close()
        os.remove(f)
        print(" ")
```

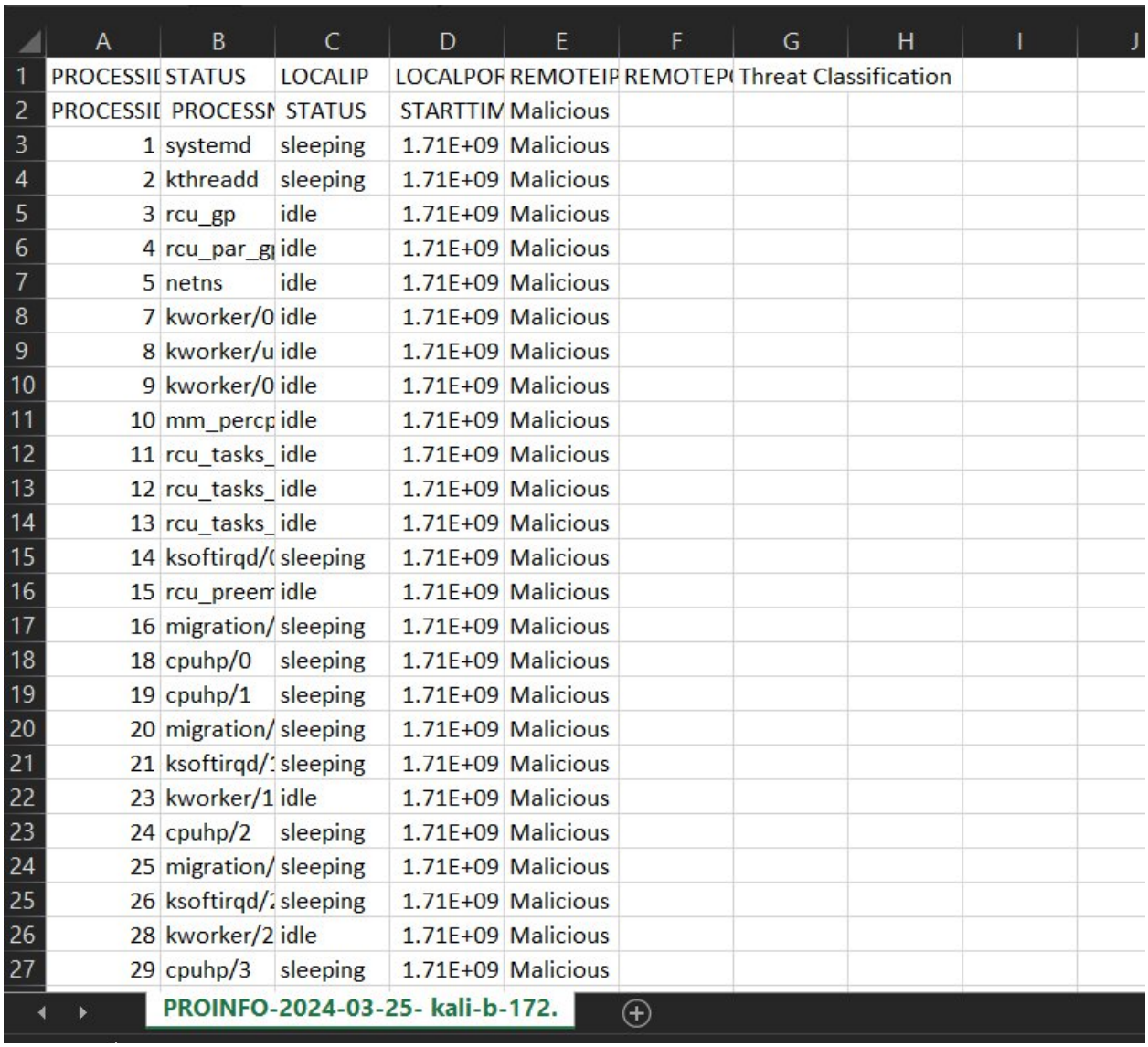


	A	B	C	D	E	F	G	H	I
1	PROCESSII	STATUS	LOCALIP	LOCALPOF	REMOTEIP	REMOTEP(	Threat Classification		
2	NAME	DETAILS	Malicious						
3	platform	Linux	Malicious						
4	platform-r	5.18.0-kal	Malicious						
5	platform-v	#1 SMP PF	Malicious						
6	architectu	x86_64	Malicious						
7	hostname	kali	Malicious						
8	ip-address	127.0.1.1	Malicious						
9	mac-addre	Malicious							
10	processor		Malicious						
11	ram	4 GB	Malicious						
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									
27									

SYSINFO-2024-03-25- kali-b-172.

	A	B	C	D	E	F	G	H	I
1	PROCESSID	STATUS	LOCALIP	LOCALPOR	REMOTEIP	REMOTEP(	Threat Classification		
2	PROCESSID	STATUS	LOCALIP	LOCALPOI	REMOTEII	REMOTEP	Malicious		
3	6228	NONE	fe80::7461	56895				Not Malicious	
4	0	TIME_WAI	172.20.128	54503	addr(ip='18	port=443)	Malicious		
5	0	TIME_WAI	172.20.128	54521	addr(ip='1(	port=443)	Malicious		
6	4	NONE	192.168.56	137				Not Malicious	
7	23980	ESTABLISH	172.20.128	53901	addr(ip='2(	port=443)	Highly Malicious		
8	6596	NONE	0.0.0.0	5353				Not Malicious	
9	3872	ESTABLISH	172.20.128	54490	addr(ip='2(	port=443)	Highly Malicious		
10	5536	NONE	127.0.0.1	50340				Not Malicious	
11	6228	NONE	::1	1900				Not Malicious	
12	6228	NONE	::1	56896				Not Malicious	
13	1472	LISTEN	0.0.0.0	135				Malicious	
14	27772	ESTABLISH	127.0.0.1	54542	addr(ip='1'	port=4444	Highly Malicious		
15	27632	NONE	::	54634				Not Malicious	
16	0	TIME_WAI	172.20.128	54519	addr(ip='1(	port=443)	Malicious		
17	1172	LISTEN	::	49664				Malicious	
18	0	TIME_WAI	172.20.128	54509	addr(ip='8.	port=443)	Malicious		
19	2396	NONE	0.0.0.0	5355				Not Malicious	
20	6228	NONE	172.20.128	56898				Not Malicious	
21	26832	ESTABLISH	172.20.128	53918	addr(ip='1'	port=5228	Highly Malicious		
22	27632	ESTABLISH	172.20.128	53898	addr(ip='5'	port=443)	Highly Malicious		
23	26832	NONE	0.0.0.0	63504				Not Malicious	
24	27632	NONE	0.0.0.0	52332				Not Malicious	
25	6228	NONE	192.168.56	56897				Not Malicious	
26	24328	CLOSE_WA	172.20.128	53996	addr(ip='1(	port=443)	Malicious		
27	0	TIME_WAI	172.20.128	54534	addr(ip='1'	port=443)	Malicious		

NETINFO-2024-03-25- NINAD-b-127



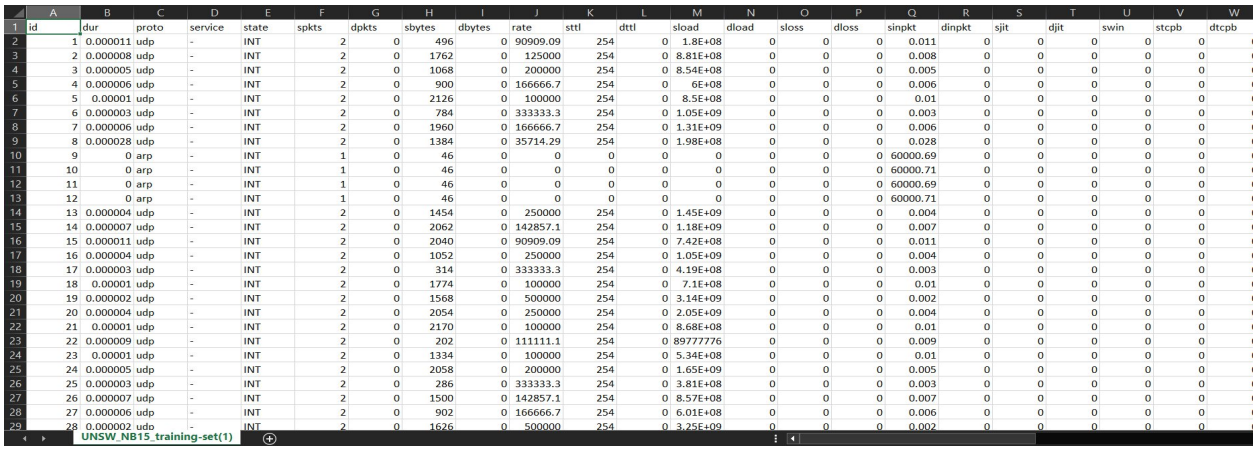
	A	B	C	D	E	F	G	H	I	J
1	PROCESSI	STATUS	LOCALIP	LOCALPOF	REMOTEIP	REMOTEP(	Threat Classification			
2	PROCESSI	PROCESSN	STATUS	STARTTIM	Malicious					
3	1	systemd	sleeping	1.71E+09	Malicious					
4	2	kthreadd	sleeping	1.71E+09	Malicious					
5	3	rcu_gp	idle	1.71E+09	Malicious					
6	4	rcu_par_g	idle	1.71E+09	Malicious					
7	5	netns	idle	1.71E+09	Malicious					
8	7	kworker/0	idle	1.71E+09	Malicious					
9	8	kworker/u	idle	1.71E+09	Malicious					
10	9	kworker/0	idle	1.71E+09	Malicious					
11	10	mm_percp	idle	1.71E+09	Malicious					
12	11	rcu_tasks_	idle	1.71E+09	Malicious					
13	12	rcu_tasks_	idle	1.71E+09	Malicious					
14	13	rcu_tasks_	idle	1.71E+09	Malicious					
15	14	ksoftirqd/(	sleeping	1.71E+09	Malicious					
16	15	rcu_preem	idle	1.71E+09	Malicious					
17	16	migration/	sleeping	1.71E+09	Malicious					
18	18	cpuhp/0	sleeping	1.71E+09	Malicious					
19	19	cpuhp/1	sleeping	1.71E+09	Malicious					
20	20	migration/	sleeping	1.71E+09	Malicious					
21	21	ksoftirqd/	sleeping	1.71E+09	Malicious					
22	23	kworker/1	idle	1.71E+09	Malicious					
23	24	cpuhp/2	sleeping	1.71E+09	Malicious					
24	25	migration/	sleeping	1.71E+09	Malicious					
25	26	ksoftirqd/	sleeping	1.71E+09	Malicious					
26	28	kworker/2	idle	1.71E+09	Malicious					
27	29	cpuhp/3	sleeping	1.71E+09	Malicious					

PROINFO-2024-03-25- kali-b-172.

**** We ran multiple VM's and then fetched data one by one from these VM's and then compiled it together. After getting CSV file we applied our own CNN model and then got this this accuracy. ****

Random forest Modelling for Network Intrusion Detection System

DATASET :



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss	dloss	sinpkt	dinpkt	sjit	djit	swin	stcpb	dtcpb
2	1	0.000011	udp	-	INT	2	0	496	0	90909.09	254	0	1.8E+08	0	0	0	0.011	0	0	0	0	0	
3	2	0.000008	udp	-	INT	2	0	1762	0	125000	254	0	8.81E+08	0	0	0	0.008	0	0	0	0	0	
4	3	0.000005	udp	-	INT	2	0	1068	0	200000	254	0	8.54E+08	0	0	0	0.005	0	0	0	0	0	
5	4	0.000006	udp	-	INT	2	0	900	0	166666.7	254	0	6E+08	0	0	0	0.006	0	0	0	0	0	
6	5	0.00001	udp	-	INT	2	0	2126	0	100000	254	0	8.5E+08	0	0	0	0.01	0	0	0	0	0	
7	6	0.000003	udp	-	INT	2	0	784	0	333333.3	254	0	1.05E+09	0	0	0	0.003	0	0	0	0	0	
8	7	0.000006	udp	-	INT	2	0	1960	0	166666.7	254	0	1.31E+09	0	0	0	0.006	0	0	0	0	0	
9	8	0.000028	udp	-	INT	2	0	1384	0	35714.29	254	0	1.98E+08	0	0	0	0.028	0	0	0	0	0	
10	9	0	arp	-	INT	1	0	46	0	0	0	0	0	0	0	0	60000.69	0	0	0	0	0	
11	10	0	arp	-	INT	1	0	46	0	0	0	0	0	0	0	0	60000.71	0	0	0	0	0	
12	11	0	arp	-	INT	1	0	46	0	0	0	0	0	0	0	0	60000.69	0	0	0	0	0	
13	12	0	arp	-	INT	1	0	46	0	0	0	0	0	0	0	0	60000.71	0	0	0	0	0	
14	13	0.000004	udp	-	INT	2	0	1454	0	250000	254	0	1.45E+09	0	0	0	0.004	0	0	0	0	0	
15	14	0.000007	udp	-	INT	2	0	2062	0	142857.1	254	0	1.18E+09	0	0	0	0.007	0	0	0	0	0	
16	15	0.000011	udp	-	INT	2	0	2040	0	90909.09	254	0	7.42E+08	0	0	0	0.011	0	0	0	0	0	
17	16	0.000004	udp	-	INT	2	0	1052	0	250000	254	0	1.05E+09	0	0	0	0.004	0	0	0	0	0	
18	17	0.000003	udp	-	INT	2	0	314	0	333333.3	254	0	4.19E+08	0	0	0	0.003	0	0	0	0	0	
19	18	0.00001	udp	-	INT	2	0	1774	0	100000	254	0	7.1E+08	0	0	0	0.01	0	0	0	0	0	
20	19	0.000002	udp	-	INT	2	0	1568	0	500000	254	0	3.14E+09	0	0	0	0.002	0	0	0	0	0	
21	20	0.000004	udp	-	INT	2	0	2054	0	250000	254	0	2.05E+09	0	0	0	0.004	0	0	0	0	0	
22	21	0.00001	udp	-	INT	2	0	2170	0	100000	254	0	8.68E+08	0	0	0	0.01	0	0	0	0	0	
23	22	0.000009	udp	-	INT	2	0	202	0	111111.1	254	0	89777776	0	0	0	0.009	0	0	0	0	0	
24	23	0.00001	udp	-	INT	2	0	1334	0	100000	254	0	5.34E+08	0	0	0	0.01	0	0	0	0	0	
25	24	0.000005	udp	-	INT	2	0	2058	0	200000	254	0	1.65E+09	0	0	0	0.005	0	0	0	0	0	
26	25	0.000003	udp	-	INT	2	0	286	0	333333.3	254	0	3.81E+08	0	0	0	0.003	0	0	0	0	0	
27	26	0.000007	udp	-	INT	2	0	1500	0	142857.1	254	0	8.57E+08	0	0	0	0.007	0	0	0	0	0	
28	27	0.000006	udp	-	INT	2	0	902	0	166666.7	254	0	6.01E+08	0	0	0	0.006	0	0	0	0	0	
29	28	0.000002	udp	-	INT	2	0	1626	0	500000	254	0	3.25E+09	0	0	0	0.002	0	0	0	0	0	

UNSW_NB15_training-set(1)

(82332, 45)

Modules

Random Forest Classifier

• Ensemble Method Strength: Random forest classifier is an ensemble learning method that combines multiple decision trees to create a robust model. This ensemble technique is particularly advantageous for network intrusion detection systems (NIDS) as it can effectively handle the complexity and variability of network traffic data.

• Feature Importance Analysis: Random forests provide a straightforward approach to assessing feature importance, which is crucial in NIDS for identifying the most relevant network traffic attributes indicative of intrusions. By analyzing feature importance, security analysts can prioritize their efforts in monitoring and mitigating potential threats.

• Robustness to Overfitting: Random forests are less prone to overfitting compared to individual decision trees, making them suitable for NIDS where generalization to unseen data is crucial. The randomization of feature selection and bootstrap sampling during training helps in building diverse trees, which collectively reduce overfitting.

• Handling Imbalanced Data: In the context of NIDS, network traffic data often suffer from class imbalance, where normal traffic significantly outweighs malicious traffic. Random forest classifiers inherently handle imbalanced datasets well by assigning appropriate class weights or utilizing techniques like bootstrapping with replacement, ensuring that minority classes (intrusions) are adequately represented in the model.

• Parallelization and Scalability: Random forests can be parallelized, allowing for efficient training on large-scale network datasets. This scalability is beneficial for NIDS applications, where datasets can be extensive and real-time or near real time detection is essential.

• Interpretability and Explainability: Although random forests are considered black-box models to some extent, they still offer a degree of interpretability by providing insights into feature importance and decision paths. This interpretability is valuable for NIDS analysts to understand the rationale behind intrusion detection decisions and to refine the model based on domain knowledge.

Implementation Details

Random forest classifier

1. Data Preprocessing:

• Load the network traffic dataset, ensuring it's properly formatted.

• Handle missing values, categorical features, and feature scaling if necessary.

• Split the dataset into training and testing sets.

2. Model Training:

• Import the Random Forest classifier from a machine learning library like scikit-learn.

• Define the hyperparameters such as the number of trees, maximum depth, and minimum samples per leaf.

• Train the Random Forest model using the training data.

3. Model Evaluation:

• Evaluate the trained model using the testing data.

• Calculate metrics such as accuracy, precision, recall, and F1-score to assess the model's performance.

• Visualize feature importance to understand which network traffic attributes contribute most to intrusion detection

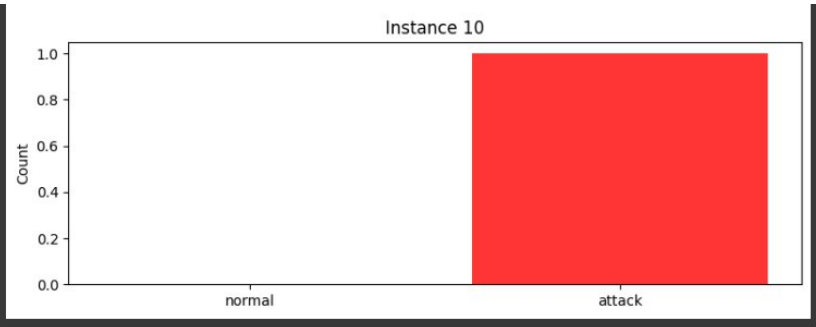
```
[ ] clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
[ ] y_pred = clf.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9997267344770001
Classification Report:
              precision    recall  f1-score   support

      attack       1.00      1.00      1.00     79452
      normal       1.00      1.00      1.00     19353

    accuracy                           1.00     98805
   macro avg       1.00      1.00      1.00     98805
weighted avg       1.00      1.00      1.00     98805
```

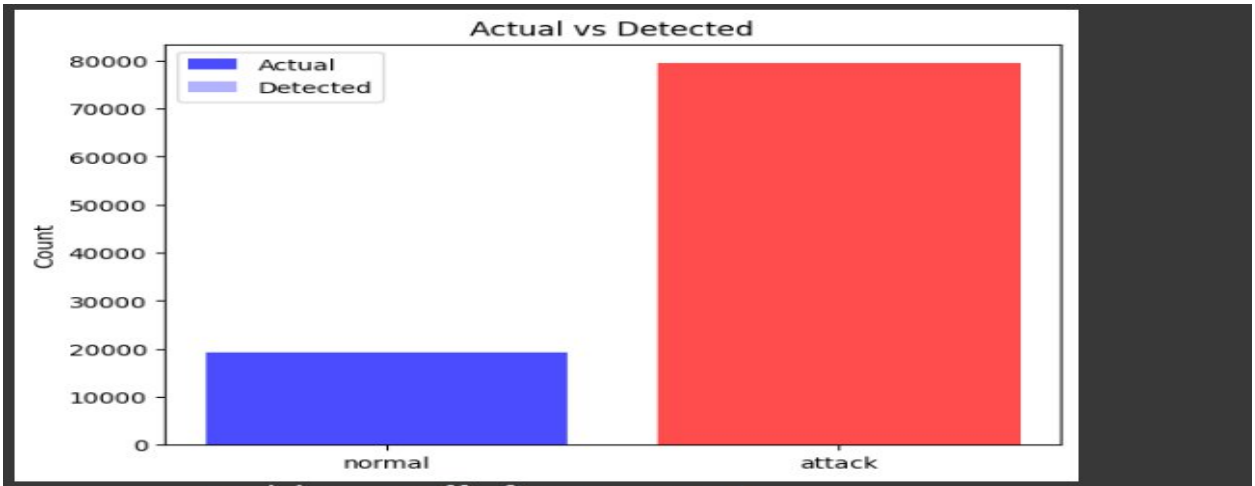
```
[ ] classes = ["normal", "attack"]
    fig, ax = plt.subplots()
    ax.bar(classes, [len(y_test) - np.sum(y_test == "attack"), np.sum(y_test == "attack")], color=['blue', 'red'], alpha=0.7)
    ax.bar(classes, [len(y_attack) - attack_detected, attack_detected], color=['blue', 'red'], alpha=0.3)
    ax.set_ylabel('Count')
    ax.set_title('Actual vs Detected')
    ax.legend(['Actual', 'Detected'])
    plt.show()
    print(classification_report(y_test, y_pred))

    fig, axes = plt.subplots(nrows=len(X_attack), ncols=1, figsize=(8, len(X_attack) * 3))
```




```
[ ] attack_samples = np.random.randint(0, len(X_test), 10)  # Simulate 10 attack instances
    X_attack = X_test.iloc[attack_samples]
    y_attack = y_test.iloc[attack_samples]
    y_attack_pred = clf.predict(X_attack)
```

```
[ ] for i in range(len(X_attack)):
        print(f"Instance {i+1}:")
        print(f"Actual label: {y_attack.iloc[i]}")
        print(f"Predicted label: {y_attack_pred[i]}")
        print("--------------------")
    attack_detected = np.sum(y_attack_pred == "attack")
    print(f"Attacks detected: {attack_detected}/{len(X_attack)}")
```

```
Instance 1:
Actual label: attack
Predicted label: attack
--------------------
Instance 2:
Actual label: attack
Predicted label: attack
```



	precision	recall	f1-score	support
attack	1.00	1.00	1.00	79452
normal	1.00	1.00	1.00	19353
accuracy			1.00	98805
macro avg	1.00	1.00	1.00	98805
weighted avg	1.00	1.00	1.00	98805

Implementing machine learning in a Network Intrusion Detection System (NIDS) involves several key steps. Here's a general guide on how to go about it:

1. Data Collection:

• Gather a comprehensive dataset that includes both normal and malicious network traffic.

• Ensure the dataset is diverse, representing various types of attacks and normal activities.

2. Data Preprocessing:

• Clean and preprocess the data to handle missing values, outliers, and irrelevant features.

• Transform categorical data into a suitable format for machine learning algorithms.

3. Feature Selection:

• Identify and select relevant features that contribute most to the detection of intrusions.

• Remove redundant or less informative features to enhance model efficiency.

4. Data Splitting:

• Split the dataset into training and testing sets to evaluate the model's performance accurately.

5. Choosing Algorithms:

• Select appropriate machine learning algorithms based on the nature of the problem.

• Common algorithms for NIDS include decision trees, random forests, support vector machines, and neural networks.

6. Model Training:

- Train the selected machine learning models using the training dataset.
- Fine-tune hyperparameters to optimize the model's performance.

7. Evaluation:

- Evaluate the model using the testing dataset to assess its accuracy, precision, recall, and other relevant metrics.
- Address any overfitting or underfitting issues.

8. Ensemble Techniques:

- Consider using ensemble techniques such as bagging or boosting to improve the model's robustness.

9. Feature Importance Analysis:

- Analyze the importance of different features in the model to understand the key indicators of intrusions.

10. Real-time Implementation:

- Integrate the trained model into the NIDS for real-time monitoring of network traffic.
- Ensure that the system can efficiently handle the computational requirements of the machine learning model.

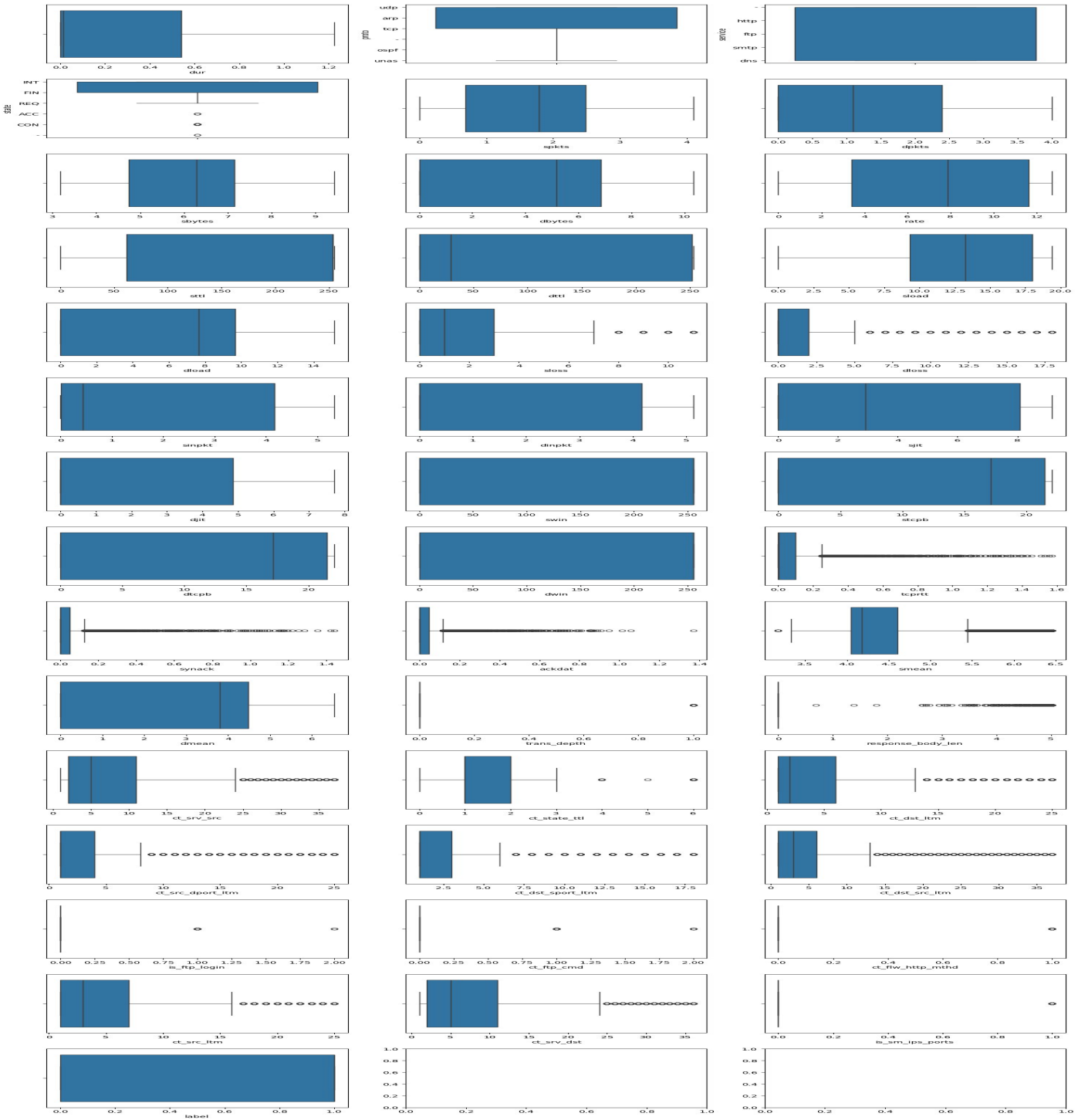
11. Continuous Monitoring and Updating:

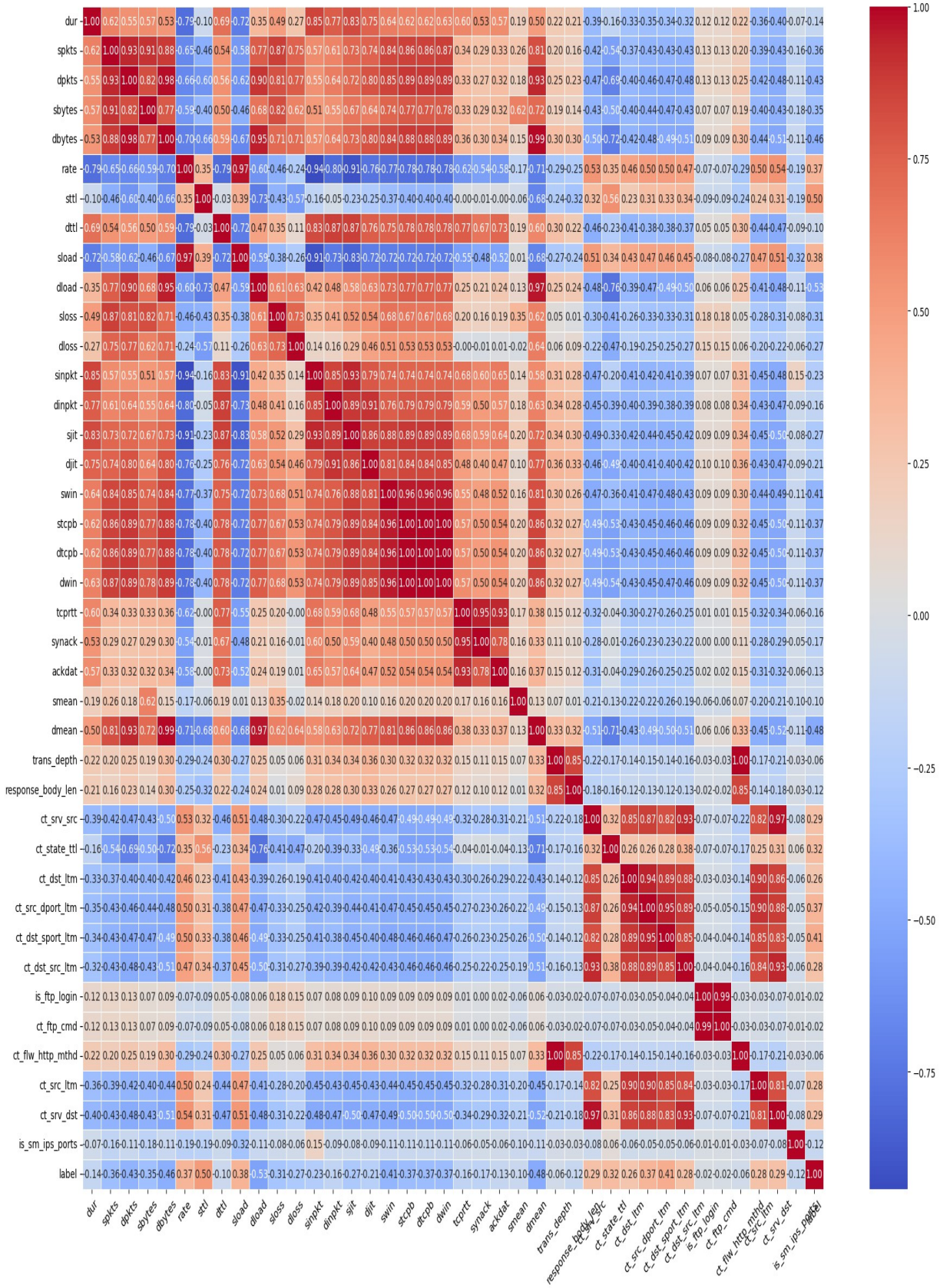
- Regularly update the NIDS with new data to adapt to evolving attack patterns.
- Implement mechanisms for continuous monitoring and improvement of the machine learning model.

12. Feedback Loop:

- Establish a feedback loop to gather information about the performance of the NIDS in real- world scenarios.
- Use this feedback to refine the model and enhance its accuracy over time.

By following these steps, you can successfully implement machine learning in a Network Intrusion Detection System, creating a robust and adaptive solution for identifying and mitigating security threats in network traffic.


dur
proto
udp
arp
tcp
-
ospf
unas
service
-
http
ftp
smtp
dns
state
INT
FIN
REQ
ACC
CON
-
spkts
dpkts
sbytes
dbytes
rate
sttl
dttl
sload
dload
sloss
dloss
sinpkt
dinpkt
sjit
djit
swin
stcpb
dtcpb
dwin
tcprtt
synack
ackdat
smean
dmean
trans_depth
response_body_len
ct_srv_src
ct_state_ttl
ct_dst_ltm
ct_src_dport_ltm
ct_dst_sport_ltm
ct_dst_src_ltm
is_ftp_login
ct_ftp_cmd
ct_flw_http_mthd
ct_src_ltm
ct_srv_dst
is_sm_ips_ports
label

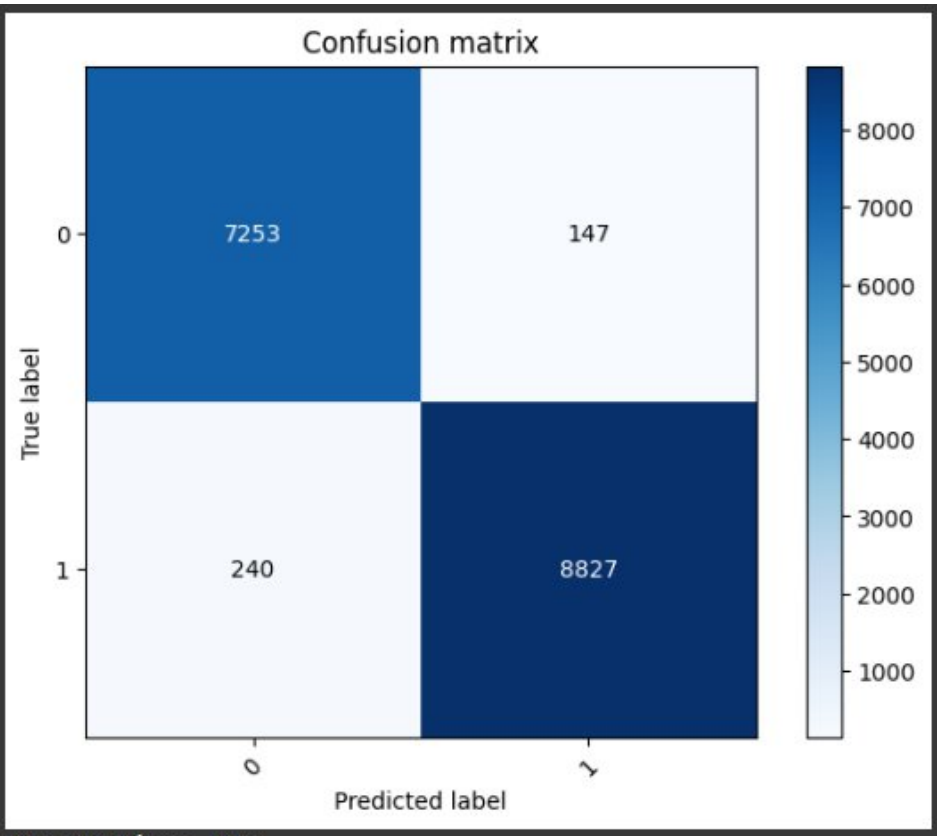

Correlation Matrix
dur
spkts
dpkts
sbytes
dbytes
rate
sttl
dttl
sload
dload
sloss
dloss
sinpkt
dinpkt
sjit
djit
swin
stcpb
dtcpb
dwin
tcprtt
synack
ackdat
smean
dmean
trans_depth
response_body_len
ct_srv_src
ct_state_ttl
ct_dst_ltm
ct_src_dport_ltm
ct_dst_sport_ltm
ct_dst_src_ltm
is_ftp_login
ct_ftp_cmd
ct_flw_http_mthd
ct_src_ltm
ct_srv_dst
is_sm_ips_ports
label
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

I

Results of Random Forest Classifier- Accuracy and Confusion Matrix

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

rf_classifier = RandomForestClassifier(
    n_estimators=180,
    criterion='gini',
    max_depth=40,
    random_state=1
)

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report_str)
```

```
Accuracy: 0.9765

Confusion Matrix:
[[7253  147]
 [ 240 8827]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      7400
           1       0.98      0.97      0.98      9067

    accuracy                           0.98     16467
   macro avg       0.98      0.98      0.98     16467
weighted avg       0.98      0.98      0.98     16467
```




```
True Negatives: 7253
False Positives: 147
False Negatives: 240
True Positives: 8827
```