National College of Ireland

Configuration Manual

MSc Research Project
Cyber Security

Raga Malika Gudipati
Student ID: 23189525

School of Computing
National College of Ireland

Supervisor: Mr. Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Raga Malika Gudipati
Student ID:	23189525
Programme:	MSc in Cyber Secuirty **Year:** 2024-25
Module:	Research Project
Lecturer:	Mr. Michael Prior
Submission Due Date:	12th December 2024
Project Title:	DDOS attacks on airlines and mitigation techniques using Artificial Intelligence Aided System
Word Count:	1524

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Raga Malika Gudipati

Date: 12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Raga Malika Gudipati
Student ID: 23189525

1 Introduction

In our Research, we utilized two datasets with identical implementation, coding, and evaluation methodologies. As the workflow and processing steps remain consistent across both datasets, a single configuration file is sufficient for managing the parameters and settings, ensuring uniformity and efficiency in the project's execution.

2 System Requirements

Here's an ideal setup for our project:

Desktop or Laptop
- CPU: Intel i5 or AMD Ryzen 5 || Intel i7/i9 or AMD Ryzen 7/9
- RAM: 8 GB || 16 GB || 32 GB for heavy multitasking or future-proofing
- GPU: NVIDIA RTX 3060 or higher with 8 GB VRAM
- Storage: 256 GB SSD || 512 GB SSD + 1 TB HDD for backups

Alternative (Cloud Services)
If you don't have access to a suitable local machine:
- Google Colab: Free GPU support for small-scale experiments.
- AWS/GCP: On-demand high-performance machines for model training.

3 Dataset Configuration

3.1 Dataset Overview

The project utilizes (2) datasets: CIC_IOT_2023 and CIC_IOT_2019, both in CSV format. These datasets contain network traffic data; the target variable (labeled) is designed to indicate whether a DDoS attack is present. The target variable is binary (or multi-class), because the label differentiates between normal traffic and various types of DDoS attacks. This allows the model to classify network traffic as either legitimate or malicious

CIC_IOT_2023

ıtion	Rate	Srate	Drate	fin_flag_number	syn_flag_number	rst_flag_number	...	Std	Tot size	IAT	Number	Magnitue	Radius	Covariance	Variance	Weight	label
64.00	0.329807	0.329807	0.0	1.0	0.0	1.0	...	0.000000	54.00	8.334383e+07	9.5	10.392305	0.000000	0.000000	0.00	141.55	DDoS-RSTFINFlood
64.00	4.290556	4.290556	0.0	0.0	0.0	0.0	...	2.822973	57.04	8.292607e+07	9.5	10.464666	4.010353	160.987842	0.05	141.55	DoS-TCP_Flood
64.00	33.396799	33.396799	0.0	0.0	0.0	0.0	...	0.000000	42.00	8.312799e+07	9.5	9.165151	0.000000	0.000000	0.00	141.55	DDoS-ICMP_Flood
64.00	4642.133010	4642.133010	0.0	0.0	0.0	0.0	...	0.000000	50.00	8.301570e+07	9.5	10.000000	0.000000	0.000000	0.00	141.55	DoS-UDP_Flood
65.91	6.202211	6.202211	0.0	0.0	1.0	0.0	...	23.113111	57.88	8.297300e+07	9.5	11.346876	32.716243	3016.808286	0.19	141.55	DoS-SYN_Flood

CIC_IOT_2019

Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	...	min_seg_size_forward	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
192.168.10.5-104.16.207.165-54865-443-6	104.16.207.165	443	192.168.10.5	54865	6	7/7/2017 3:30	3	2	0	...	20	0.0	0.0	0	0	0.0	0.0	0	0	BENIGN
192.168.10.5-104.16.28.216-55054-80-6	104.16.28.216	80	192.168.10.5	55054	6	7/7/2017 3:30	109	1	1	...	20	0.0	0.0	0	0	0.0	0.0	0	0	BENIGN
192.168.10.5-104.16.28.216-55055-80-6	104.16.28.216	80	192.168.10.5	55055	6	7/7/2017 3:30	52	1	1	...	20	0.0	0.0	0	0	0.0	0.0	0	0	BENIGN
192.168.10.16-104.17.241.25-46236-443-6	104.17.241.25	443	192.168.10.16	46236	6	7/7/2017 3:30	34	1	1	...	20	0.0	0.0	0	0	0.0	0.0	0	0	BENIGN
192.168.10.5-104.19.196.102-54863-443-6	104.19.196.102	443	192.168.10.5	54863	6	7/7/2017 3:30	3	2	0	...	20	0.0	0.0	0	0	0.0	0.0	0	0	BENIGN

3.2 Data Pre-processing Steps

```
categorical_cols = df.select_dtypes(include=['object']).columns



for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])


numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
num_imputer = SimpleImputer(strategy='mean')  # Replace missing values with the mean value
df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])


target_column = 'label'
X = df.drop(target_column, axis=1)
y = df[target_column]


scaler = StandardScaler()
X = scaler.fit_transform(X)


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

print("Data is ready for model training!")
```

Label Encoding for Categorical Features:
Identify categorical columns using select_dtypes and then encode them using LabelEncoder. This will transform the string-based categorical data into a numerical format that can be used to train the model.

Handling Missing Values in Numeric Features:
The numeric columns are identified, and missing values are imputed using the SimpleImputer with the strategy set to mean. This ensures that any missing values in the dataset are replaced with the mean of the respective columns.

Feature and Target Split:
The target column ('label') is separated from the feature columns. The features (X) are stored in a new DataFrame, and the target variable (y) is stored separately.

Feature Scaling:
All features of the dataset are standardized with StandardScaler, which normalizes the data to a mean of 0 and a standard deviation of 1. This is so that the scale of features doesn't negatively impact the training of the model.

Train-Test Split:
The data is split into training and testing sets using train_test_split. 80% of the data is used for training, and 20% is reserved for testing, with the random state set to 42 for reproducibility.

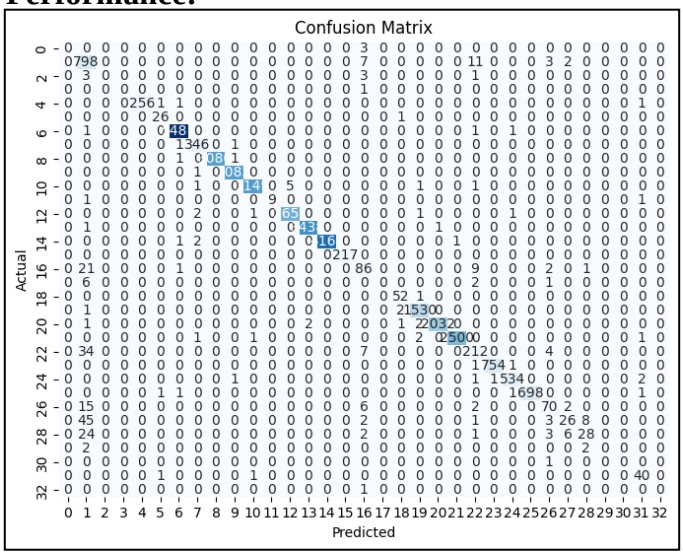
4. Model Configuration

4.1 Model type: Random Forest Classifier

```
# Random Forest classifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt


rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42) # You can adjust n_estimators
rf_classifier.fit(X_train, y_train)


y_pred = rf_classifier.predict(X_test)


print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))


cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

The configuration of the Random Forest classifier will be using 100 estimators (n_estimators=100) and a fixed random seed (random_state=42) to make it reproducible. Then, it trains the model on the given training data, X_train and y_train, and evaluates it with the test set, X_test and y_test. It assesses performance with metrics like accuracy, a detailed classification report, and a confusion matrix with a heatmap for easy interpretation.

Performance:

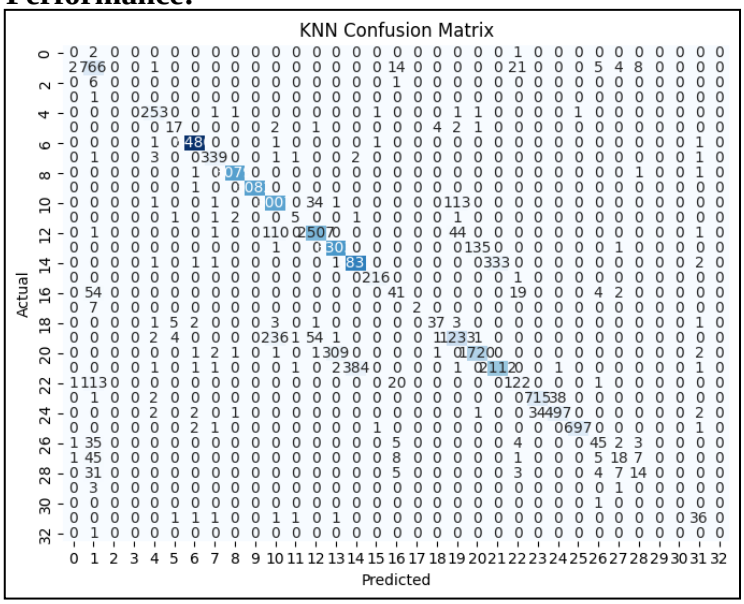


4.2 Model type: K neighbour Classifier

```
#  KNN classifier

from sklearn.neighbors import KNeighborsClassifier


knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can adjust n_neighbors
knn_classifier.fit(X_train, y_train)

y_pred_knn = knn_classifier.predict(X_test)


print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("\nKNN Classification Report:\n", classification_report(y_test, y_pred_knn))


cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_knn, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("KNN Confusion Matrix")
plt.show()
```

The configuration for the KNN classifier defines 5 neighbors to use for prediction, n_neighbors=5. It trains on the training set X_train and y_train and tests on the test set X_test and y_test. Performance is reported using accuracy, a classification report, and a confusion matrix that's visualized using a heatmap for easy interpretation.

Performance:

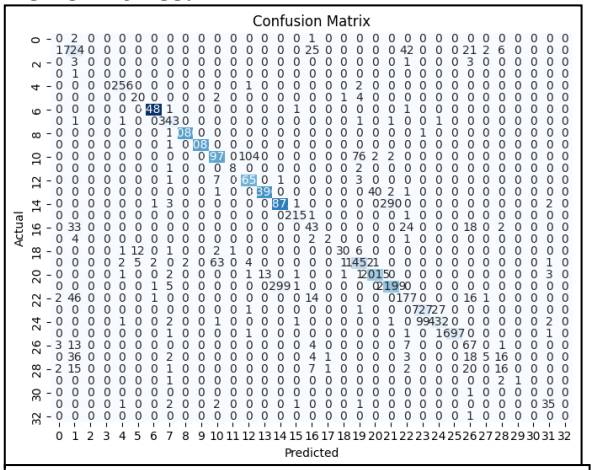


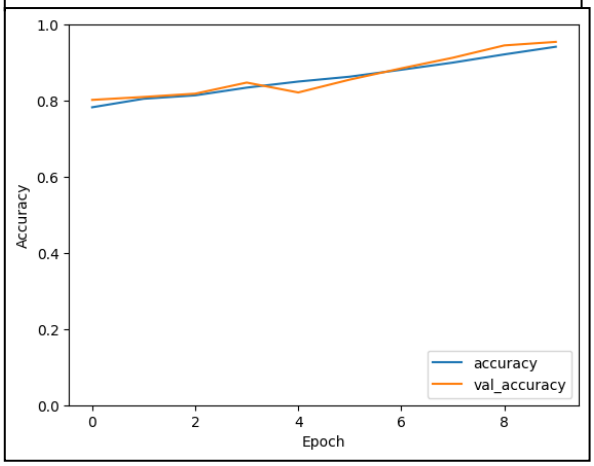
4.3 Model type: Tabular Neural Network

```
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(np.unique(y_train)), activation='softmax') # Output layer with softmax
])


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',  # Use sparse categorical crossentropy
              metrics=['accuracy'])


history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)



loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")


y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
```

The configuration for the neural network uses a sequential model with three dense layers: an input layer of 128 neurons with ReLU activation, a hidden layer of 64 neurons with ReLU activation, and an output layer with softmax activation corresponding to the number of unique target classes. The model is compiled with the Adam optimizer, sparse categorical cross entropy loss, and accuracy as the evaluation metric. Training is done for 10 epochs with a batch size of 32, using 20% of the training data as a validation split. The model's performance is evaluated against the test set, with the metrics including test loss, accuracy, a classification report, and a confusion matrix. Training history is visualized through accuracy and loss curves for both training and validation phases.

Performance:



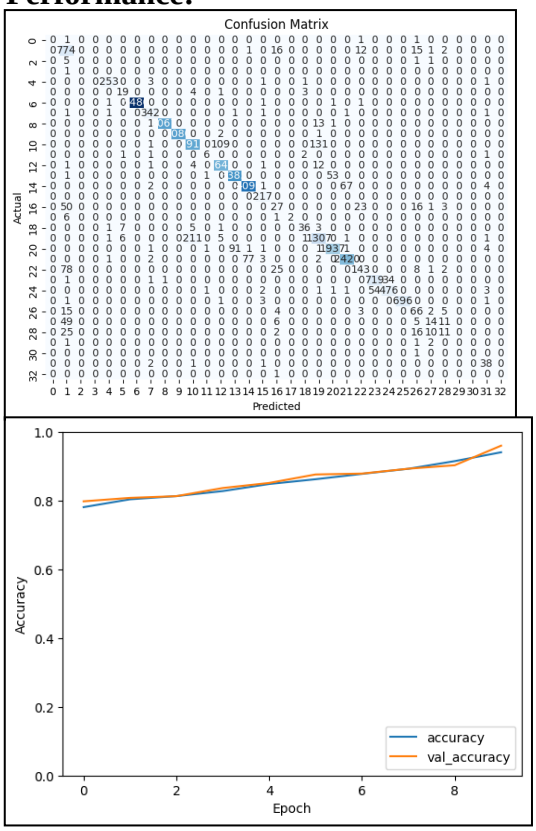


4.4 Model type: CNN-GRU Architecture

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, GRU, Dense, Flatten, MaxPooling1D
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns


model = Sequential()

model.add(Conv1D(filters=64, kernel_size=1, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(MaxPooling1D(pool_size=1))
model.add(GRU(units=50, activation='tanh'))  # GRU layer
model.add(Dense(units=len(np.unique(y_train)), activation='softmax'))


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)


loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")


y_pred_probs = model.predict(X_test)
```

The configuration for the CNN-GRU architecture includes a Conv1D layer with 64 filters and a kernel size of 1, followed by a MaxPooling1D layer and a GRU layer with 50 units using tanh activation. The output layer for the model is a dense layer with softmax activation, which is the number of unique classes in the target. It is compiled with the Adam optimizer, sparse categorical cross entropy as the loss function, and accuracy as the evaluation metric. The model trains for 10 epochs with a batch size of 32, using 20% of the training data for validation. Its performance is evaluated using test loss, accuracy, a classification report, and a confusion matrix, with training progress visualized through accuracy and loss curves for both training and validation phases.

Performance:

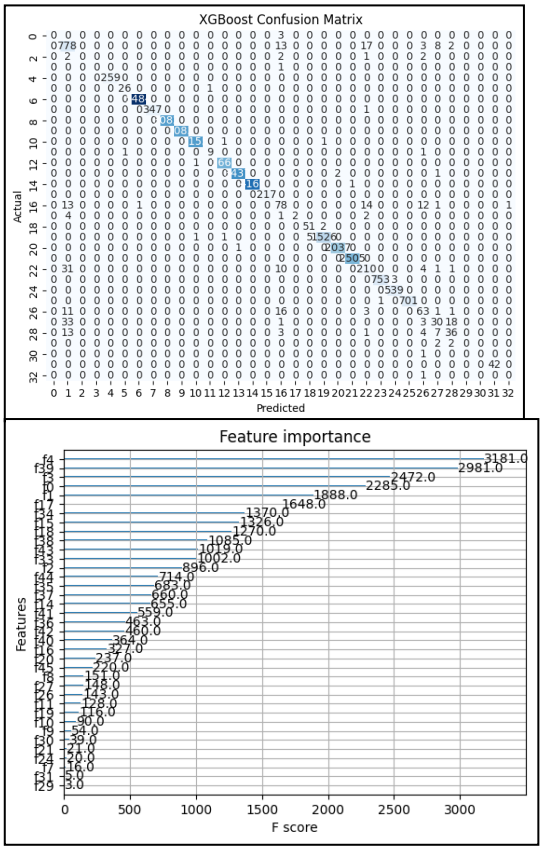


4.5 Model type: Xgboost Classifier

```
# Xgboost Classifier

import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns


X_train = X_train.reshape(X_train.shape[0], -1)


X_test = X_test.reshape(X_test.shape[0], -1)


xgb_classifier = xgb.XGBClassifier(objective='multi:softmax', num_class=len(np.unique(y_train)), random_state=42)
xgb_classifier.fit(X_train, y_train)



y_pred_xgb = xgb_classifier.predict(X_test)


print("XGBoost Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("\nXGBoost Classification Report:\n", classification_report(y_test, y_pred_xgb))


cm_xgb = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_xgb, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

The configuration for the XGBoost classifier reshapes the training and test data to a 2D array, as required by XGBoost. The classifier is initiated with the objective set as multi:softmax to deal with multi-class classification problems and the number of classes defined based on the unique values in the target variable (y_train). The model is trained on the reshaped training data (X_train and y_train) and evaluated on the test set (X_test and y_test). Performance metrics include accuracy, a classification report, and a confusion matrix. The confusion matrix is visualized with a heatmap, and feature importance is plotted using XGBoost's built-in plot_importance function.

Performance:

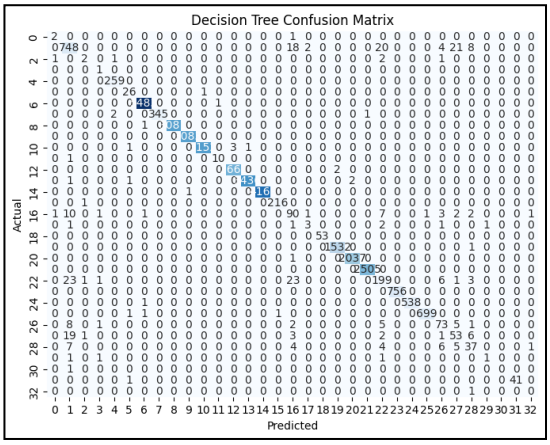


4.6 Model type: Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier


dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)


y_pred_dt = dt_classifier.predict(X_test)


print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("\nDecision Tree Classification Report:\n", classification_report(y_test, y_pred_dt))


cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dt, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Decision Tree Confusion Matrix")
plt.show()


from sklearn import tree
plt.figure(figsize=(20,10))
tree.plot_tree(dt_classifier, filled=True, rounded=True, class_names=True, feature_names=df.drop('label', axis=1).columns)
plt.show()
```

The decision-tree classifier is configured. Initializes this model with a fixed seed, for reproducibility purpose to random_state =42; in case it's applied with actual training data, like (X_train and y_train); applied and tested on X_test and y_test data using Accuracy, Classification report along with confusion matrix as assessment methods. The confusion matrix is represented as a heatmap, and the decision tree is drawn using plot_tree from sklearn.tree, which represents the tree structure with features and class labels for easy interpretation.

Performance:

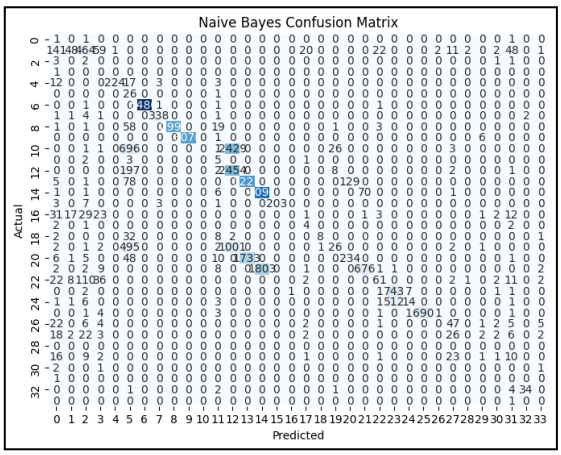


Model type: Naïve Bayes

```
#  naive bayes Classifier

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns


nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)


y_pred_nb = nb_classifier.predict(X_test)


print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("\nNaive Bayes Classification Report:\n", classification_report(y_test, y_pred_nb))


cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Naive Bayes Confusion Matrix")
plt.show()
```

The configuration of Naive Bayes classifier (specifically) involves employing GaussianNB class from scikit-learn. It is trained on supplied training data: X_train and y_train and evaluated using test data (X_test and y_test). Performance metrics encompass accuracy (a classification report) and confusion matrix. The confusion matrix is visualized utilizing heatmap to facilitate interpretation. However, the effectiveness of these metrics can vary because they depend on the nature of data. This variability is important to consider; although it might seem minor at first glance.

Performance:



References

1. **Zhao, L., & Chen, K.** (2023). *Performance Comparison of Machine Learning Algorithms for IoT Intrusion Detection*. Presented at the 15th International Conference on Network Security.
2. **Kumar, P., & Singh, A.** (2022). *DDoS Attack Mitigation in IoT Networks Using Hybrid Neural Networks*. Published in *Journal of Cybersecurity Research*.
3. **Ramesh, B., Patel, J., & Smith, A.** (2023). *IoT Security: Machine Learning for DDoS Detection*. Published in *International Journal of Network Security*.
4. **Sharma, R., & Gupta, A.** (2023). *Deep Learning Approaches for Securing IoT Systems Against DDoS Attacks*. Published in *IoT Security Review*.
5. **Alshammari, F., & Hussain, F.** (2023). *Feature Engineering for IoT DDoS Detection: A Comprehensive Analysis Using CIC_IOT Datasets*. Published in *Cyber-Physical Systems Journal*.
6. **IoT-DDoS-Detection** by *cybersecAI*
   Link: https://github.com/cybersecAI/IoT-DDoS-Detection
   Description: Provides code for preprocessing and training ML models for IoT-based DDoS detection using the CIC datasets.
7. **Network-Security-ML** by *secureNetAI*
   Link: https://github.com/secureNetAI/Network-Security-ML
   Description: A repository focused on feature extraction and machine learning models for network intrusion detection.