

Configuration Manual

MSc Research Project
Master of Science in Cyber Security

Shifan Anwar Sayyed
Student ID: 22193162

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Shifan Anwar Sayyed
Student ID: 22193162
Programme: Master of Science in Cyber Security **Year:** 2023 - 2024
Module: MSc Research Practicum
Lecturer: Jawad Salahuddin
Submission Due Date: 12th August 2024 14:00
Project Title: Anomaly Detection Method for OT/ICS Environment Using Ensemble Learning
Word Count: 1342 **Page Count:** 13

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: Shifan Anwar Sayyed

Date: 12th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Shifan Anwar Sayyed
Student ID: 22193162

1 Introduction

The research papers motive is to develop an Anomaly Detection System using Ensemble Learning methods. This configuration manual presents the information regarding the hardware equipment used for performing the experiment and the software application, languages and their libraries that are used for executing the machine learning code. This document also explains the machine learning code and in which sequence were they were executed.

2 Hardware Configuration

The main hardware equipment used for implementing the experiment and executing the machine learning code is an ASUS laptop.

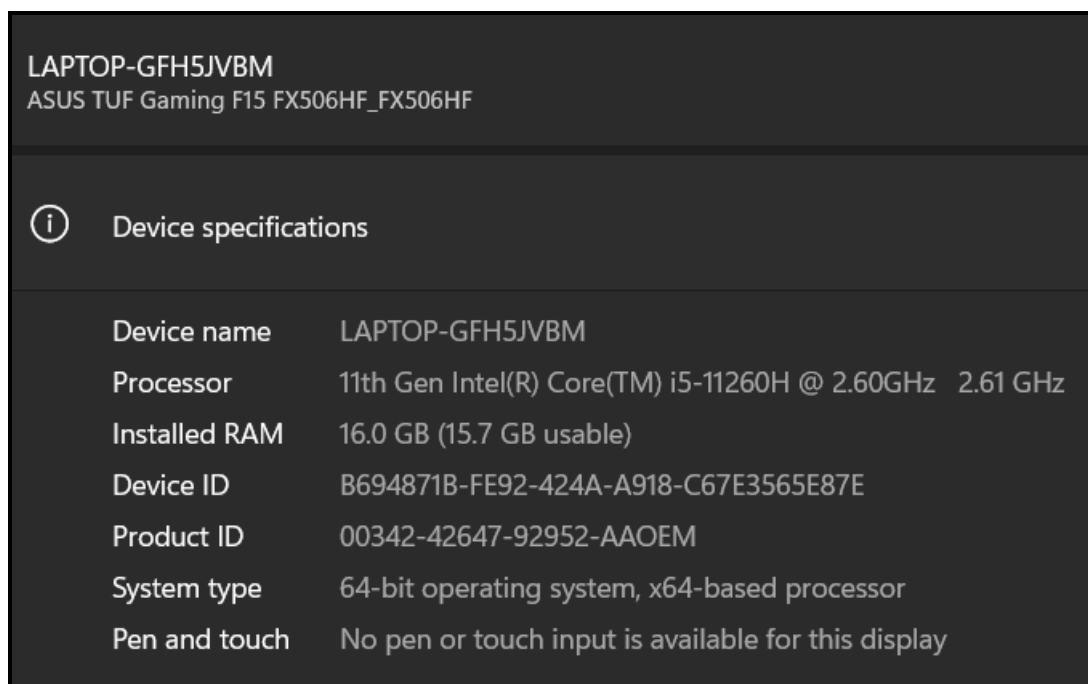
Laptop Model: ASUS TUF FX506HF-HN076W

Device name: LAPTOP-GFH5JVBM

Processor: 11th Gen Intel(R) Core (TM) i5-11260H @ 2.60GHz 2.61 GHz

Installed RAM: 16.0 GB (15.7 GB usable)

System type: 64-bit operating system, x64-based processor

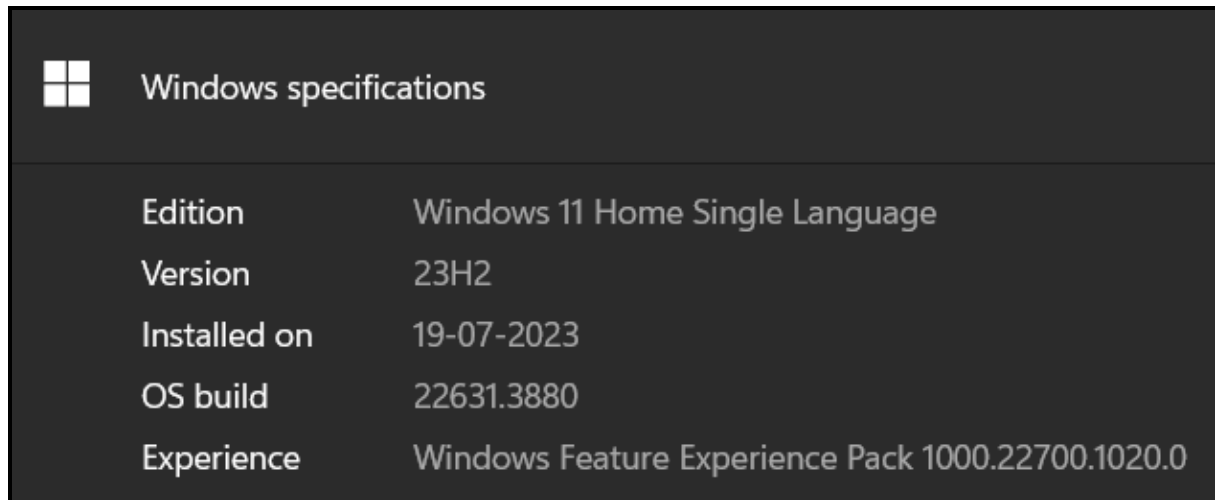


LAPTOP-GFH5JVBM ASUS TUF Gaming F15 FX506HF_FX506HF	
Device specifications	
Device name	LAPTOP-GFH5JVBM
Processor	11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz 2.61 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	B694871B-FE92-424A-A918-C67E3565E87E
Product ID	00342-42647-92952-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: ASUS Laptop Specification.

Edition: Windows 11 Home Single Language

Version: 22H2
Installed on: 19-07-2023
OS build: 22621.2428
Experience: Windows Feature Experience Pack 1000.22674.1000.0

A screenshot of the Windows 'About' page, specifically the 'Windows specifications' section. It features a dark background with white text. The title 'Windows specifications' is at the top left, preceded by the Windows logo. Below it, a table lists system details: Edition (Windows 11 Home Single Language), Version (23H2), Installed on (19-07-2023), OS build (22631.3880), and Experience (Windows Feature Experience Pack 1000.22700.1020.0).

Windows specifications	
Edition	Windows 11 Home Single Language
Version	23H2
Installed on	19-07-2023
OS build	22631.3880
Experience	Windows Feature Experience Pack 1000.22700.1020.0

Figure 2: Device Operating System Specification.

3 Software Specification

Anaconda Navigator was used as the Integrated Development Environment (IDE) for implementation of the research project and the programming language used is Python which comes already installed with Anaconda Navigator. Python was selected as it is the most commonly used for machine learning experiments and it has many libraires and packages for machine learning. Jupyter Notebook was used as the platform from within Anaconda Navigator to execute the machine learning code written in Python.

3.1 Software and Libraries Version

- Anaconda Navigator Version: 2.5.0
- Python Version: 3.11.5
- Jupyter Notebook Version: 6.5.4
- Pandas version: 2.0.3
- Numpy version: 1.24.3
- Scikit-learn version: 1.3.0

4 Implementation

4.1 Data Pre-processing

First, we import all the libraries required for performing data pre-processing. Figure 3. displays all the imported libraries for pre-processing.

```
#Importing required libraries.

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

Figure 3: Importing Required Pre-Processing Libraries.

Then we store the path of the original dataset into a variable.

```
#Storing the file path of the original dataset into a variable.
file_path = r'C:\Users\shifa\OneDrive\Desktop\CA\Sem - 3\Thesis\electra_s7comm\Original_electra_s7comm.csv'
```

Figure 4: Storing File Path of the Original Dataset.

Then we create a dictionary where we mention each label along with the count it will be under sampled to. Here we will be Under Sampling Normal Traffic to 50,00,000 and Combined Attack traffic to 50,00,000 datapoints.

```
#Creating a Dictionary for the count of labels.
target_counts = {
    'NORMAL': 5000000,
    'RESPONSE_ATTACK': 144892,
    'REPLAY_ATTACK': 26757,
    'COMMAND_ATTACK': 575122,
    'READ_ATTACK': 1063311,
    'WRITE_ATTACK': 1063306,
    'FALSE_ERROR_RESPONSE_ATTACK': 1063306,
    'MITM_UNALTERED': 1063306
}
```

Figure 5: Defining Dictionary for Target Count.

Then we create a dictionary to store the under sampled datapoints of each label. As the dataset is huge it can't be stored directly in a single dataframe hence we mention the chunk size in the variable chunk_size.

```
#Creating a Dictionary to store under sampled datapoints for each label.
sampled_data = {key: [] for key in target_counts.keys()}

#Initiating Chunk Size count. As the dataset is huge, we have to process it in chunks.
chunk_size = 2000000
reader = pd.read_csv(file_path, chunksize=chunk_size)
```

Figure 6: Defining Dictionary for Under Sampled Datapoints.

First, we create a For loop for processing the dataset in chunks of 2000000 then a nested For loop for looping through each label mentioned in the target_count variable. Then we filter the datapoints of the current chunk to the corresponding label. Then we calculate the number of under sampled datapoints for the current label and if the datapoints is less than the count mentioned in the target_count variable then we calculate the remaining required datapoints and then append it to the sample data.

```
#Process the dataset in chunks of 2000000 datapoints.
for chunk in reader:

    #Loop through each label mentioned in the target count variable.
    for label in target_counts.keys():

        #Filter the datapoints of the current chunk to the corresponding current label only.
        label_data = chunk[chunk['label'] == label]

        #Calculate the numbers of under sampled datapoints for the current label.
        current_count = sum([len(df) for df in sampled_data[label]])

        #Is True when the number of datapoints in the current chunk is less than the number of datapoints mentioned
        #in the target count variable.
        if current_count < target_counts[label]:

            #Calculate the remaining datapoints required for meeting the count mentioned in target count variable.
            sample_size = min(target_counts[label] - current_count, len(label_data))

            #Append the required remaining datapoints for the current label to sample data.
            sampled_data[label].append(label_data.sample(n=sample_size))
```

Figure 7: Under Sampling the Datapoints.

Once all the chunks are processed, we concat all the sample data for each of the label into a dataframe and then we randomize the sequence of the datapoints and finally export the under sampled dataframe to a CSV.

```
#Concating all the sample data for each label in a dataframe.
final_data = pd.concat([pd.concat(data) for data in sampled_data.values()])

#Shuffling dataset for randoming the datapoints sequence.
final_data = final_data.sample(frac=1).reset_index(drop=True)

#Exporting the Under Sampled Dataframe to CSV.
final_data.to_csv('UnderSampled_Balanced_Dataset.csv', index=False)
```

Figure 8: Concatenating all the under sampled labels.

Then we load the Under sampled and balanced dataset into a dataframe using the pd.read_csv() method.

```
#Loading the dataset into a dataframe.
df = pd.read_csv(r"C:\Users\shifa\OneDrive\Desktop\CA\Sem - 3\Thesis\electra_s7comm\UnderSampled_Balanced_Dataset.csv")
```

Figure 9: Loading Dataset into Dataframe.

Performing feature selection where we remove irrelevant feature and features with string values using the drop() method.

```
#Feature Selection. Removing irrelevant string columns.  
df.drop(columns=['smac', 'dmac', 'sip', 'dip'], inplace=True)
```

Figure 10: Feature Selection.

Performing label encoding where we map the label string values to numeric values. Here we map Normal traffic to 0 and Attack traffic to 1. After that we remove the original label column which has string label values.

```
#Label Encoding. Mapping 0 with Normal and 1 with Attack.  
df['encoded_label'] = np.where(df['label'] == 'NORMAL', 0, 1)  
  
#Removing the original label column with string values.  
df = df.drop(columns=['label'])
```

Figure 11: Label Encoding.

Here first we replace infinity values with NaN (Not a Number) values and then drop rows with NaN values.

```
#Handling Missing, Infinity and NaN values.  
df.replace([np.inf, -np.inf], np.nan, inplace=True)  
df.dropna(inplace=True)
```

Figure 12: Handling Missing and NaN values.

Next we perform standardization of the features. So first we separate the features and the label and then initiate StandardScaler(). Then we standardize the features and then convert them into a dataframe and at the last step we concat the original label with the standardized features.

```
#Separating Dataset into Features and Label.  
features = df.drop(columns=['encoded_label'])  
labels = df['encoded_label']  
  
#Initiating StandardScaler for Standardization.  
scaler = StandardScaler()  
  
#Standardizing Features.  
scaled_features = scaler.fit_transform(features)  
  
#Converting the stored standardized features into dataframe with original feature column names.  
scaled_df = pd.DataFrame(scaled_features, columns=features.columns)  
  
#Concating standardized features along with original label.  
standardized_df = pd.concat([scaled_df, labels.reset_index(drop=True)], axis=1)
```

Figure 13: Performing Standardization.

Finally, we export the pre-processed dataframe to a CSV.

```
#Exporting the Pre-Processed and Standardized Dataframe to CSV.
standardized_df.to_csv('Electra_S7Comm.csv', index=False)
```

Figure 14: Exporting the Pre-Processed Dataframe.

4.2 Ensemble Learning Code

First, we import all the libraries required for performing the experiment. Figure 3. Displays all the imported libraries for the experiment.

```
#Importing required Libraries.

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import VotingClassifier, StackingClassifier, BaggingClassifier, AdaBoostClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

Figure 15: Importing Required Libraries.

Then we load the pre-processed dataset into a dataframe using the `pd.read_csv()` method.

```
#Loading the dataset into a dataframe.
data = pd.read_csv(r"C:\Users\shifa\OneDrive\Desktop\CA\Sem - 3\Thesis\electra_s7comm\Final Dataset\Electra_S7Comm.csv")
```

Figure 16: Loading Dataset into Dataframe.

Next, we must create training and testing subset of the dataset. For that first we must separate the features and labels from the dataset and store them in variable. Where features are stored in variable `X` and label is stored in variable `y`.

Then using the **train_test_split** library we create training subset - **X_train** and **y_train** and testing subset **X_test** and **y_test**. For parameters, we keep the **test_size** which is the percentage of split between training and testing subsets as **70% training subset** and **30% testing subset** and **random_state** as **42** for reproducibility.

```
#Separating Dataset into Features and Label.
X = data.drop(columns=['encoded_label'])
y = data['encoded_label']

#Splitting the Dataset into Training and Testing subsets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```


Figure 17: Creating Training and Testing Subsets.

Here we **initiate all the 7 base classifiers**, for utilizing multiple threads of the CPU parallelly we keep the parameter of **n_jobs=-1** for the base classifiers which support it, **random_state=42** for reproducibility, **max_iter** for specifying the maximum number of iteration for optimization and **n_estimators** for specifying the number of trees.

```
#Initiating Base Classifiers.
lr = LogisticRegression(max_iter=2000, n_jobs=-1)
rf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
xgb = XGBClassifier(n_estimators=100, n_jobs=-1, random_state=42)
nb = GaussianNB()
dt = DecisionTreeClassifier(random_state=42)
knn = KNeighborsClassifier(n_jobs=-1)
mlp = MLPClassifier(max_iter=2000)
```

Figure 18: Initiating Base Classifiers.

Here we **train all the 7 base classifiers** on training subsets - **X_train** and **y_train**.

```
#Training Base Classifiers.
lr.fit(X_train, y_train)
rf.fit(X_train, y_train)
xgb.fit(X_train, y_train)
nb.fit(X_train, y_train)
dt.fit(X_train, y_train)
knn.fit(X_train, y_train)
mlp.fit(X_train, y_train)
```

Figure 19: Training Base Classifiers.

Here we **predict all the 7 base classifiers** on test subset - **X_test** which only contains the features.

```
#Predicting Base Classifiers.
lr_pred = lr.predict(X_test)
rf_pred = rf.predict(X_test)
xgb_pred = xgb.predict(X_test)
nb_pred = nb.predict(X_test)
dt_pred = dt.predict(X_test)
knn_pred = knn.predict(X_test)
mlp_pred = mlp.predict(X_test)
```

Figure 20: Predicting Base Classifiers.

For the evaluation part, we first initiate classification report which contains the evaluation metrics like **Precision, Recall and F-1 Score**. The parameter of **output_dict=True** converts the classification report in dictionary, this is done so later we can extract metrics of a specific class. Values of classification report are stored in variable named with each base classifier.

Then we extract the metrics from these variables of the positive class which is the attack class/1.

Then we extract the values of TP, TN, FP and FN for each base classifier via initiating the confusion matrix where the parameter `ravel()` is used for convert the values from confusion matrix into 1-Dimension array.

```

#Initiating Classification Report.
rf_report = classification_report(y_test, rf_pred, output_dict=True)
lr_report = classification_report(y_test, lr_pred, output_dict=True)
xgb_report = classification_report(y_test, xgb_pred, output_dict=True)
knn_report = classification_report(y_test, knn_pred, output_dict=True)
mlp_report = classification_report(y_test, mlp_pred, output_dict=True)
dt_report = classification_report(y_test, dt_pred, output_dict=True)
nb_report = classification_report(y_test, nb_pred, output_dict=True)

#Extracting Positive Class (Attack Class) metrics from Classification Report.
rf_metrics = rf_report['1']
lr_metrics = lr_report['1']
xgb_metrics = xgb_report['1']
knn_metrics = knn_report['1']
mlp_metrics = mlp_report['1']
dt_metrics = dt_report['1']
nb_metrics = nb_report['1']

#Extracting TP, TN, FP and FN values for each base classifier from confusion matrix.
tn_rf, fp_rf, fn_rf, tp_rf = confusion_matrix(y_test, rf_pred).ravel()
tn_lr, fp_lr, fn_lr, tp_lr = confusion_matrix(y_test, lr_pred).ravel()
tn_xgb, fp_xgb, fn_xgb, tp_xgb = confusion_matrix(y_test, xgb_pred).ravel()
tn_knn, fp_knn, fn_knn, tp_knn = confusion_matrix(y_test, knn_pred).ravel()
tn_mlp, fp_mlp, fn_mlp, tp_mlp = confusion_matrix(y_test, mlp_pred).ravel()
tn_dt, fp_dt, fn_dt, tp_dt = confusion_matrix(y_test, dt_pred).ravel()
tn_nb, fp_nb, fn_nb, tp_nb = confusion_matrix(y_test, nb_pred).ravel()

```

Figure 21: Evaluation of Base Classifiers Part - 1.

Then we calculate the Accuracy, False Positive Rate (FPR) and False Positive Rate (FNR) and extract the metric values of Precision, Recall and F-1 Score.

```

#Calculating Accuracy for each base classifier.
rf_accuracy = accuracy_score(y_test, rf_pred)
lr_accuracy = accuracy_score(y_test, lr_pred)
xgb_accuracy = accuracy_score(y_test, xgb_pred)
knn_accuracy = accuracy_score(y_test, knn_pred)
mlp_accuracy = accuracy_score(y_test, mlp_pred)
dt_accuracy = accuracy_score(y_test, dt_pred)
nb_accuracy = accuracy_score(y_test, nb_pred)

#Calculating Precision for each base classifier.
precision_lr = lr_metrics['precision']
precision_rf = rf_metrics['precision']
precision_xgb = xgb_metrics['precision']
precision_knn = knn_metrics['precision']
precision_mlp = mlp_metrics['precision']
precision_dt = dt_metrics['precision']
precision_nb = nb_metrics['precision']

#Calculating Recall for each base classifier.
recall_lr = lr_metrics['recall']
recall_rf = rf_metrics['recall']
recall_xgb = xgb_metrics['recall']
recall_knn = knn_metrics['recall']
recall_mlp = mlp_metrics['recall']
recall_dt = dt_metrics['recall']
recall_nb = nb_metrics['recall']

```

Figure 22: Evaluation of Base Classifiers Part – 2.

```

#Calculating F1-Score for each base classifier.
f1_score_lr = lr_metrics['f1-score']
f1_score_rf = rf_metrics['f1-score']
f1_score_xgb = xgb_metrics['f1-score']
f1_score_knn = knn_metrics['f1-score']
f1_score_mlp = mlp_metrics['f1-score']
f1_score_dt = dt_metrics['f1-score']
f1_score_nb = nb_metrics['f1-score']

#Calculating False Positive Rate(FPR) for each base classifier.
fpr_lr = fp_lr / (fp_lr + tn_lr)
fpr_rf = fp_rf / (fp_rf + tn_rf)
fpr_xgb = fp_xgb / (fp_xgb + tn_xgb)
fpr_knn = fp_knn / (fp_knn + tn_knn)
fpr_mlp = fp_mlp / (fp_mlp + tn_mlp)
fpr_dt = fp_dt / (fp_dt + tn_dt)
fpr_nb = fp_nb / (fp_nb + tn_nb)

#Calculating False Negative Rate(FNR) for each base classifier.
fnr_rf = fn_rf / (fn_rf + tp_rf)
fnr_lr = fn_lr / (fn_lr + tp_lr)
fnr_xgb = fn_xgb / (fn_xgb + tp_xgb)
fnr_knn = fn_knn / (fn_knn + tp_knn)
fnr_mlp = fn_mlp / (fn_mlp + tp_mlp)
fnr_dt = fn_dt / (fn_dt + tp_dt)
fnr_nb = fn_nb / (fn_nb + tp_nb)

```

Figure 23: Evaluation of Base Classifiers Part – 3.

Then a dataframe is created to arrange and display the output metrics for each base classifier.

```

#Creating a list for displaying output of each base classifier.
data_base = {
    'Algorithms': ['Random Forest', 'Logistic Regression', 'XGBOOST', 'KNN', 'MLP', 'DT', 'NB'],
    'Accuracy': [rf_accuracy, lr_accuracy, xgb_accuracy, knn_accuracy, mlp_accuracy, dt_accuracy, nb_accuracy],
    'Precision': [precision_rf, precision_lr, precision_xgb, precision_knn, precision_mlp, precision_dt, precision_nb],
    'Recall': [recall_rf, recall_lr, recall_xgb, recall_knn, recall_mlp, recall_dt, recall_nb],
    'F1-Score': [f1_score_rf, f1_score_lr, f1_score_xgb, f1_score_knn, f1_score_mlp, f1_score_dt, f1_score_nb],
    'FPR': [fpr_rf, fpr_lr, fpr_xgb, fpr_knn, fpr_mlp, fpr_dt, fpr_nb],
    'FNR': [fnr_rf, fnr_lr, fnr_xgb, fnr_knn, fnr_mlp, fnr_dt, fnr_nb],
}
df_metrics_base = pd.DataFrame(data_base)

print("Base Classifiers Metrics:")
print(df_metrics_base)

```

Figure 24: Evaluation of Base Classifiers Part – 4.

Then we initiate, train and predict the ensemble technique Voting with parameter of **voting=hard** for majority voting and **n_jobs=-1** for multiple threads usage.

```
#Initiating, Training and Predicting Voting Ensemble Technique.
voting_bc = VotingClassifier(estimators=[
    ('dt', dt),
    ('knn', knn),
    ('xgb', xgb),
    ('rf', rf)], voting='hard', n_jobs=-1)

voting_bc.fit(X_train, y_train)
voting_pred = voting_bc.predict(X_test)
```

Figure 25: Initiating, Training and Predicting Voting Ensemble Technique.

Next we initiate, train and predict the ensemble technique Stacking with parameter of **final_estimator=LogisticRegression()** for specifying the meta-classifier algorithm and **n_jobs=-1** for multiple threads usage.

```
#Initiating, Training and Predicting Stacking Ensemble Technique.
stacking_bc = StackingClassifier(estimators=[
    ('dt', dt),
    ('knn', knn),
    ('xgb', xgb),
    ('rf', rf)], final_estimator=LogisticRegression(), n_jobs=-1)
stacking_bc.fit(X_train, y_train)
stacking_pred = stacking_bc.predict(X_test)
```

Figure 26: Initiating, Training and Predicting Stacking Ensemble Technique.

Next we initiate, train and predict the ensemble technique Bagging with parameter of **base_estimator=DecisionTreeClassifier(random_state=42)** for specifying the base model algorithm, **random_state=42** for reproducibility and **n_jobs=-1** for multiple threads usage.

```
#Initiating, Training and Predicting Bagging Ensemble Technique.
bagging_bc = BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=42),
                              n_estimators=100, random_state=42, n_jobs=-1)
bagging_bc.fit(X_train, y_train)
bagging_pred = bagging_bc.predict(X_test)
```

Figure 27: Initiating, Training and Predicting Bagging Ensemble Technique.

Then we initiate, train and predict the ensemble technique Boosting with parameter of **base_estimator=DecisionTreeClassifier(random_state=42)** for specifying the base model algorithm, **random_state=42** for reproducibility, **n_estimators=100** for specifying the number of trees and **n_jobs=-1** for multiple threads usage.

```
#Initiating, Training and Predicting Boosting Ensemble Technique.
boosting_bc = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(random_state=42), n_estimators=100, random_state=42)
boosting_bc.fit(X_train, y_train)
boosting_pred = boosting_bc.predict(X_test)
```

Figure 28: Initiating, Training and Predicting Boosting Ensemble Technique.

Then we carry out the same evaluation process as carried out for base classifiers, only this time for Ensemble Techniques.

```
#Initiating Classification Report.
voting_report = classification_report(y_test, voting_pred, output_dict=True)
stacking_report = classification_report(y_test, stacking_pred, output_dict=True)
bagging_report = classification_report(y_test, bagging_pred, output_dict=True)
boosting_report = classification_report(y_test, boosting_pred, output_dict=True)

#Extracting Positive Class (Attack Class) metrics from Classification Report.
voting_metrics = voting_report['1']
stacking_metrics = stacking_report['1']
bagging_metrics = bagging_report['1']
boosting_metrics = boosting_report['1']

#Extracting TP, TN, FP and FN values for each base classifier from confusion matrix.
tn_voting, fp_voting, fn_voting, tp_voting = confusion_matrix(y_test, voting_pred).ravel()
tn_stacking, fp_stacking, fn_stacking, tp_stacking = confusion_matrix(y_test, stacking_pred).ravel()
tn_bagging, fp_bagging, fn_bagging, tp_bagging = confusion_matrix(y_test, bagging_pred).ravel()
tn_boosting, fp_boosting, fn_boosting, tp_boosting = confusion_matrix(y_test, boosting_pred).ravel()
```

Figure 29: Evaluation of Ensemble Techniques Part – 1.

```
#Calculating Accuracy for each Ensemble Technique.
voting_accuracy = accuracy_score(y_test, voting_pred)
stacking_accuracy = accuracy_score(y_test, stacking_pred)
bagging_accuracy = accuracy_score(y_test, bagging_pred)
boosting_accuracy = accuracy_score(y_test, boosting_pred)

#Calculating Precision for each Ensemble Technique.
precision_voting = voting_metrics['precision']
precision_stacking = stacking_metrics['precision']
precision_bagging = bagging_metrics['precision']
precision_boosting = boosting_metrics['precision']

#Calculating Recall for each Ensemble Technique.
recall_voting = voting_metrics['recall']
recall_stacking = stacking_metrics['recall']
recall_bagging = bagging_metrics['recall']
recall_boosting = boosting_metrics['recall']

#Calculating F1-Score for each Ensemble Technique.
f1_score_voting = voting_metrics['f1-score']
f1_score_stacking = stacking_metrics['f1-score']
f1_score_bagging = bagging_metrics['f1-score']
f1_score_boosting = boosting_metrics['f1-score']
```

Figure 30: Evaluation of Ensemble Techniques Part – 2.

```

#Calculating False Positive Rate(FPR) for each Ensemble Technique.
fpr_voting = fp_voting / (fp_voting + tn_voting)
fpr_stacking = fp_stacking / (fp_stacking + tn_stacking)
fpr_bagging = fp_bagging / (fp_bagging + tn_bagging)
fpr_boosting = fp_boosting / (fp_boosting + tn_boosting)

#Calculating False Negative Rate(FNR) for each Ensemble Technique.
fnr_voting = fn_voting / (fn_voting + tp_voting)
fnr_stacking = fn_stacking / (fn_stacking + tp_stacking)
fnr_bagging = fn_bagging / (fn_bagging + tp_bagging)
fnr_boosting = fn_boosting / (fn_boosting + tp_boosting)

#Creating a list for displaying output of each Ensemble Technique.
ensemble_data_base = {
    'Algorithms': ['Voting', 'Stacking', 'Bagging', 'Boosting'],
    'Accuracy': [voting_accuracy, stacking_accuracy, bagging_accuracy, boosting_accuracy],
    'Precision': [precision_voting, precision_stacking, precision_bagging, precision_boosting],
    'Recall': [recall_voting, recall_stacking, recall_bagging, recall_boosting],
    'F1-Score': [f1_score_voting, f1_score_stacking, f1_score_bagging, f1_score_boosting],
    'FPR': [fpr_voting, fpr_stacking, fpr_bagging, fpr_boosting],
    'FNR': [fnr_voting, fnr_stacking, fnr_bagging, fnr_boosting]
}

df_metrics_base_ensemble = pd.DataFrame(ensemble_data_base)

print("Ensemble Classifiers Metrics:")
print(df_metrics_base_ensemble)

```

Figure 31: Evaluation of Ensemble Techniques Part – 3.

After executing the above code you will get the output for Ensemble Techniques.