National College of Ireland

Configuration Manual

MSc Research Project
Master of Science in Cyber Security

Achu Abraham George
Student ID: X23127872

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Achu Abraham George……

Student ID: ……X23127872……..……

Programme: ……MS in Cyber Security…… **Year:** …2024-2025..

Module: ……Research Project……..……

Supervisor: …… Diego Lugones

Submission Due Date: 29/01/2025 ……..……

Project Title: ……Attack Detection and Prevention Using Machine Learning and Deep Learning……..……

Word Count: ……1075…… **Page Count**……11……..……..

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: …Achu Abraham George……

Date: …28/01/2025……

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

Attack Detection and Prevention using Machine Learning and Deep Learning Techniques

Achu Abraham George

X23127872

1. Introduction

This configuration manual provides a comprehensive guide to the hardware and software requirements, as well as the implementation process, for the research project titled: ”Attack detection and prevention using machine learning and deep learning algorithms”. This manual describes the steps in preparing the datasets with the four feature scaling and selection transformation steps as well as in building the subsets for analysis and training the ML models and in validating the models.

It also discusses the work done in combining K-Nearest Neighbors (KNN) and Multilayer Perceptron (MLP) models whereby these models were run on the Internet Firewall dataset. Python codes are inserted to illustrate some parts of the developmental procedures; hence, this paper can act as a template for replicating or expanding the study.

Also, the manual explains the practical time utilization of the developed system with a green and clean user interface to predict network traffic outcomes along with their probable possibility. The last section of the paper presents the quantitative performance results of the models that were evaluated in this study with the focus on achieved validation accuracies equaled 99.83% for MLP and 95.29% for KNN classification algorithms, and reflects on how these practical findings can support the development of adaptive and intelligent cybersecurity systems.

2. System Configuration

The brief of the system hardware and software that were used while conducting the research project are as follows:

2.1 Hardware Configuration:

- Operating System: Windows 11
- Processor: 12th Gen Intel® Core™ i7-1255U 1.70GHz
- RAM: 16.0 GB
- System Type: 64-bit Operating System, x64-based processor

2.2 Software Configuration:

Software/Tools	Version	Description
Anaconda Navigator	2.6.3	Graphical User Interface for working with Anaconda environments, installation of packages and for launching data science applications.
Jupyter Notebook	6.5.4	Web application that allows the creation of documents with live code and equations plus handling interactive elements and visualizations.
Python	3.8	In this project, python is used to develop models
Pandas	2.2.2	An integrated data manipulation and analysis library, that had DataFrames for structured data handling.
Numpy	1.26.4	Library for numerical computations, large multi-dimensional arrays and matrices..
Matplotlib	3.9.2	Library used for construction of static, animated and interactive graphs and charts.
Scikit-learn	1.5.1	A brilliant machine learning library to perform classification, regression, clustering and preprocessing functions.
Seaborn	0.13.2	Python based visualization library that aims at providing a high-level interface for statistical graphics based on the Matplotlib toolkit.
TensorFlow	2.18.0	Applies to training and development of machine learning and deep learning schemes and models. TensorFlow can compute on both CPU and GPU and has options for model deployment on multiple environments: mobile and web.

3. Download and Implementation

This section describes how to implement the project on any Windows system. The following are the steps.

a. Download and install Anaconda Navigator Software:

The Anaconda Navigator is the Graphical User Interface where users can install, run, update, even start up Python/R environments and Data Science tools in particular use for Jupyter Notebooks, Spyder and many more more without needing to type any command script.

b. Create a new environment

Based on the best practices, it is recommended that different projects be run in different environments.

- Open **Anaconda Prompt**
- To create the environment with a particular python version:
  conda create --name myproject python=3.9
- To activate the created environment:
  conda activate myproject

c. Install Jupyter Notebook

conda install -c conda-forge notebook

d. Install required libraries in environment
conda install -c conda-forge jupyter
notebook=6.5.4 numpy=1.26.4 pandas=2.2.2
matplotlib=3.9.2 seaborn=0.13.2 scikit-learn=1.5.1

conda install -c conda-forge tensorflow=2.18.0

e. Add Kernel to Jupyter Notebook
python -m ipykernel install --user --name=myproject --display-name "Python (myproject)"

4. Dataset

The dataset is sourced from Kaggle. The dataset employed in this research is known as Internet Firewall DataSet with 12 attributes and 65532 records of ISP traffic. They are numerical, containing 11 variables such as Source Port, Bytes Sent, Elapsed Time in sec, and Categorical containing only one variable, Action, as label.

Dataset link: https://www.kaggle.com/datasets/tunguz/internet-firewall-data-set

5. Configuration and Execution

5.1 Importing Libraries

To implement the model, essential libraries are imported.

```
import os
import sys
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

# Ensure correct working directory
print("Current Working Directory:", os.getcwd())


# Now import the module

# Other imports
import pandas as pd
import numpy as np
import matplotlib
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
import random
from sklearn.model_selection import train_test_split
```

Fig 1. Importing libraries

5.2 Data Preprocessing

For the analysis and machine learning, transforming raw data into a clean and usable format.

1. Checked for null values. There is no null values available in the dataset.

Check for Null Values ¶

```
df.isnull().sum()
```

```
Source Port             0
Destination Port        0
NAT Source Port         0
NAT Destination Port    0
Bytes                   0
Bytes Sent              0
Bytes Received          0
Packets                 0
Elapsed Time (sec)      0
pkts_sent               0
pkts_received           0
Action                  0
dtype: int64
```

Fig 2. Checking for Null values

2. Checked for Infinite Values. There is no infinite values available in the dataset.

Check for Infinite Values

```
for feature in df.columns:
    if feature == 'Action':
        continue
    num_infinite = np.isinf(df[feature]).sum()
    if num_infinite > 0:
        print(f"Feature: {feature:<30} | Infinite Values: {num_infinite}")
    else:
        print(f"Feature: {feature:<30} | 0 Infinite Values.")
```

```
Feature: Source Port                   | 0 Infinite Values.
Feature: Destination Port              | 0 Infinite Values.
Feature: NAT Source Port               | 0 Infinite Values.
Feature: NAT Destination Port          | 0 Infinite Values.
Feature: Bytes                         | 0 Infinite Values.
Feature: Bytes Sent                    | 0 Infinite Values.
Feature: Bytes Received                | 0 Infinite Values.
Feature: Packets                       | 0 Infinite Values.
Feature: Elapsed Time (sec)            | 0 Infinite Values.
Feature: pkts_sent                     | 0 Infinite Values.
Feature: pkts_received                 | 0 Infinite Values.
```

Fig 3. Checking for infinite values

3. Plotted graph for analyse Action Feature and got to know that it was highly imbalanced.

Ploting Graph for Analyse Action Feature

```
with plt.style.context("fivethirtyeight"):
    ax = df['Action'].value_counts().plot.bar(
        fontsize=14,
        figsize=(10, 5)
    )
    plt.title("Analyzing Action Feature Using Bar Chart", pad=10)
    plt.xlabel("Class Action")
    plt.ylabel("Number of Records")
    plt.xticks(rotation=90)

    # Each bar with its height counts
    for p in ax.patches:
        ax.annotate(
            str(int(p.get_height())),
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='bottom', fontsize=10, rotation=90
        )

    plt.show()
```

Fig. 4 Plotting graph for Analyse Action features

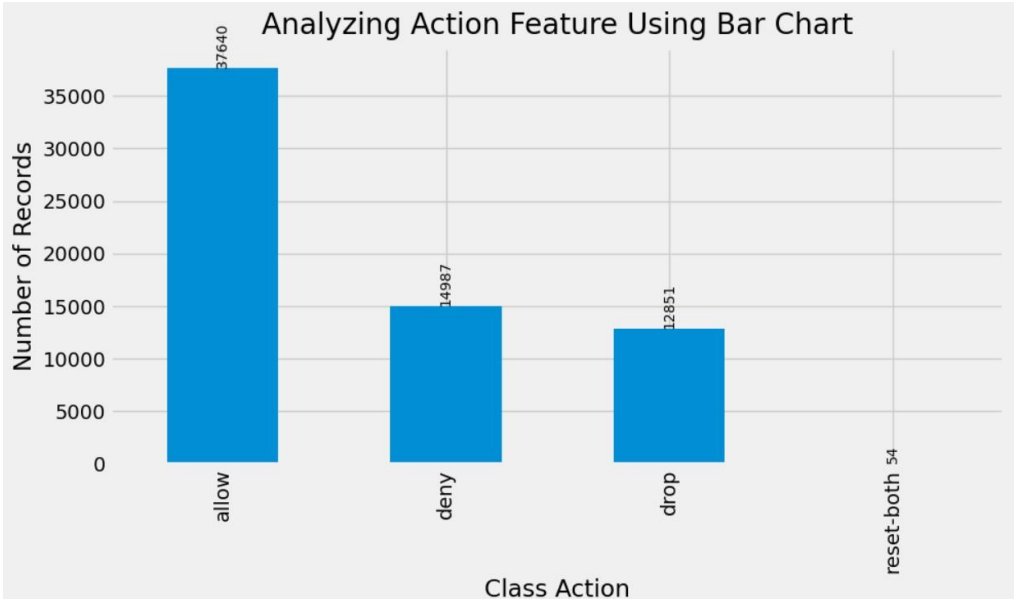


Fig 5. Bar graph of imbalanced Action features

4. Action- Balanced and plotted

The Action : Allow, Deny and Drop were balanced to 12000 records

```
[38]: valid_actions = df['Action'].value_counts()[df['Action'].value_counts() >= 12000].index

      filtered_df = df[df['Action'].isin(valid_actions)]

      filtered_df = filtered_df.groupby('Action').head(12000)

      filtered_df.head()
```

[38]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received	Action
0	57222	53	54587	53	177	94	83	2	30	1	1	allow
1	56258	3389	56258	3389	4768	1600	3168	19	17	10	9	allow
2	6881	50321	43265	50321	238	118	120	2	1199	1	1	allow
3	50553	3389	50553	3389	3327	1438	1889	15	17	8	7	allow
4	50002	443	45848	443	25358	6778	18580	31	16	13	18	allow

```
[39]: filtered_df.shape
```

```
[39]: (36000, 12)
```

```
[40]: label_actions = filtered_df['Action'].value_counts()
      for action, count in label_actions.items():
          print(f"{action:<30} : {count}")
```

```
allow                          : 12000
drop                           : 12000
deny                           : 12000
```

Fig 5. Balancing the class Action to 12000 records

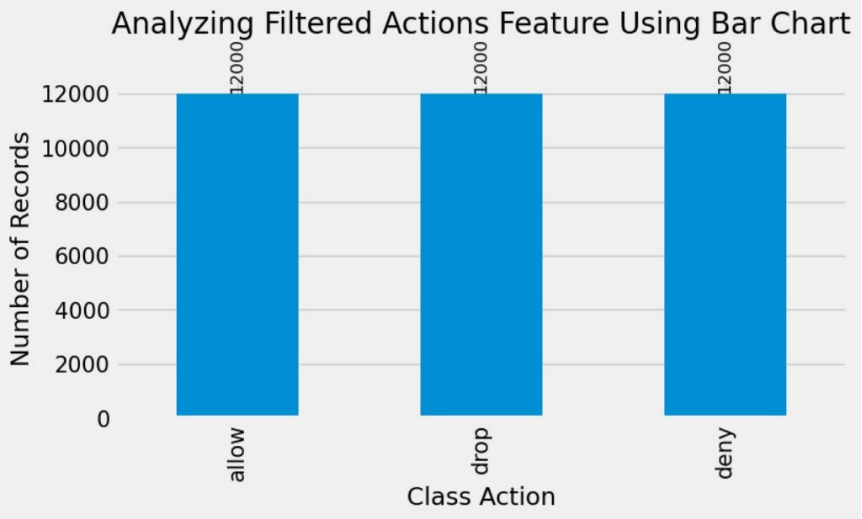


Fig 6. Bar graph of balanced Action

5. Class Action Encoding

Converted the categorical target variable to numerical. That is Allow: 0, Deny: 1, Drop: 2

```
[17]: class_dict = {}
      for idx, action in enumerate(sorted(filtered_df['Action'].unique())):
          class_dict[action] = idx
      print(class_dict)
```

```
{'allow': 0, 'deny': 1, 'drop': 2}
```

Fig 7. Class Action Encode

6. Saved Preprocessing Model and saved Normalized Data

   The preprocessing model is saved as pickle file and normalized data is saved as csv file.

Fig 8. Saving preprocessing model and normalized data

7. Performed Data Shuffling and Data Splitting

   The normalized data is shuffled and split into x_train, x_test, y_train, y_test

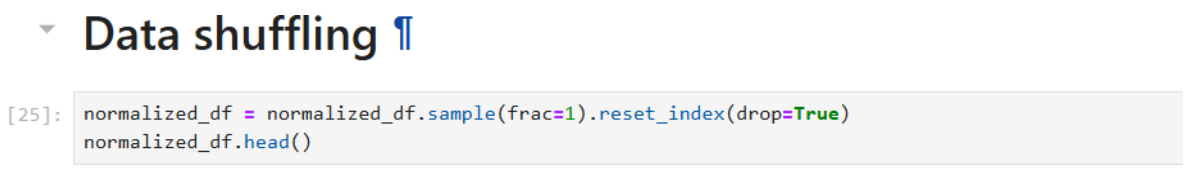


Fig 9. Shuffling the normalized data

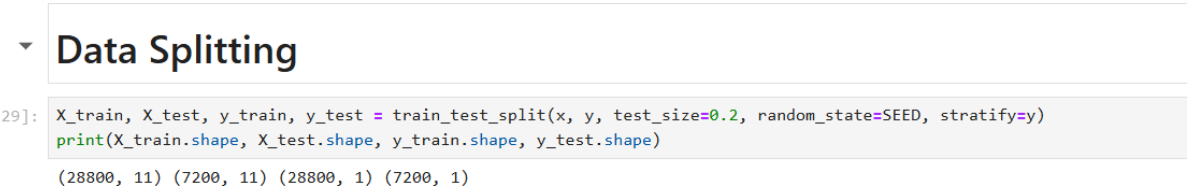


Fig 10. Splitting Data

8. Saved Train and Test Data

   The Train and Test data is saved as csv file.

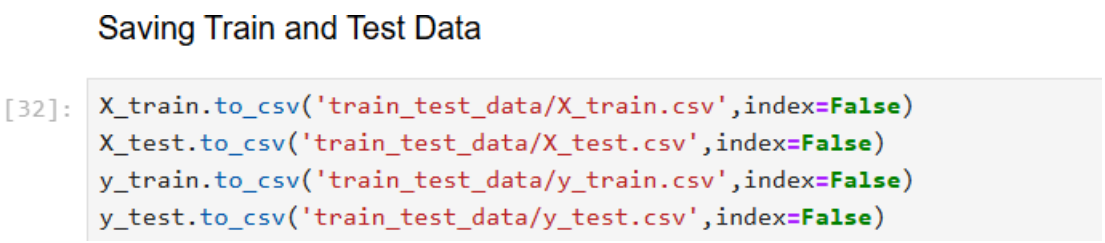


Fig 11. Saving Train and Test Data

5.3 Model Training (KNN and MLP)

1. Import Test and Train Dataset

   Here, importing x_train, x_test, y_train, y_test for the model training.

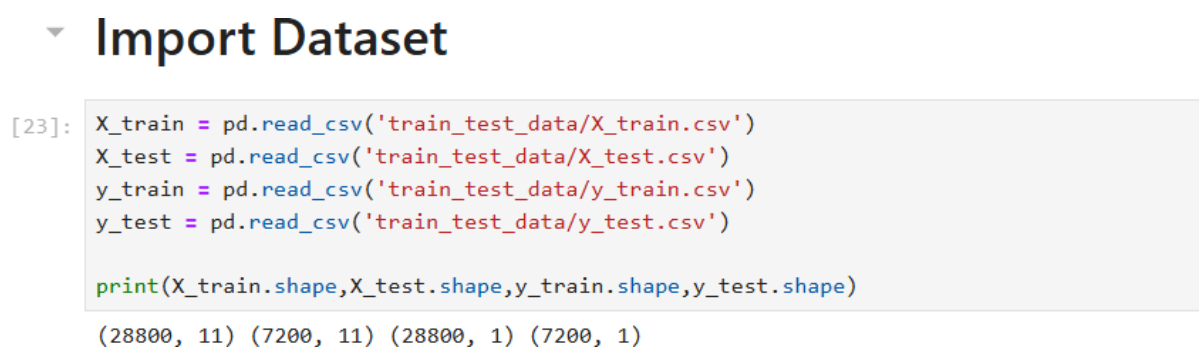


Fig 12. Importing Test and Train Dataset

2. Imported KNN Classifier and Trained KNN Model

Fig 13. Importing KNN Classifier and Training KNN Model

3. Result Analysis of KNN

   Result of KNN algorithm is analysed. Validation accuracy of KneighborsClassifier model is 95.29%.

Result Analysis

Model Accuracy for KNeighborsClassifier

[33]:
```
knc_model_accuracy=accuracy_score(y_true=y_true,y_pred=knc_prediction)
print("Validation accuracy of KNeighborsClassifier model is {:.2f}%".format(knc_model_accuracy*100))
```

```
Validation accuracy of KNeighborsClassifier model is 95.29%
```

Classification Report for KNeighborsClassifier

[34]:
```
print(classification_report(y_true=y_true, y_pred=knc_prediction, target_names=class_labels))
```

```
              precision    recall  f1-score   support

       Allow       1.00      0.91      0.95      2400
        Deny       0.97      0.95      0.96      2400
        Drop       0.90      1.00      0.95      2400

    accuracy                           0.95      7200
   macro avg       0.96      0.95      0.95      7200
weighted avg       0.96      0.95      0.95      7200
```

Fig 14. KNN Model accuracy and Classification Report

4. Saved KNeighbors Classifier Model

The model is saved as pickle file.

Saving the KNeighbors Classifier model

[36]:
```
with open(file="models/KNeighborsClassifier_model.pkl", mode="wb") as file:
    pickle.dump(obj=knc_model, file=file)
```

Fig 15. Saving Model

5. Imported Keras Modules for Algorithm: MLP

Importing Keras Modules

[38]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, BatchNormalization
from tensorflow.keras.regularizers import L2
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

Fig 16. Importing Keras Modules for MLP

6. Model Trained

Model Training

```
[41]: EPOCHS = 15

history = model.fit(
    x=X_train,
    y=y_train,
    batch_size=32,
    epochs=EPOCHS,
    validation_data=(X_test, y_test),
    callbacks=[
        ReduceLROnPlateau(
            monitor='val_accuracy',
            min_lr=0,
            patience=2
        )
    ],
    verbose=1
)
```

Fig 17. Training MLP Model

7. Analysed Result

The validation accuracy of DeepNeuralNetwork model is 99.85%.

Result Analysis

Model Accuracy for Deep Neural Network

```
[51]: model_accuracy = accuracy_score(
    y_true=true_labels,
    y_pred=predicted_labels
)

print(f"Validation accuracy of DeepNeuralNetwork model is {model_accuracy*100:.2f}%")
```

```
Validation accuracy of DeepNeuralNetwork model is 99.85%
```

classification report for Deep Neural Network

```
[52]: print(classification_report(y_true=true_labels, y_pred=predicted_labels, target_names=class_labels))
```

```
              precision    recall  f1-score   support

       Allow       1.00      1.00      1.00      2400
        Deny       1.00      1.00      1.00      2400
        Drop       1.00      1.00      1.00      2400

    accuracy                           1.00      7200
   macro avg       1.00      1.00      1.00      7200
weighted avg       1.00      1.00      1.00      7200
```

Fig. 18 Model Accuracy and Classification report of Deep Neural Network

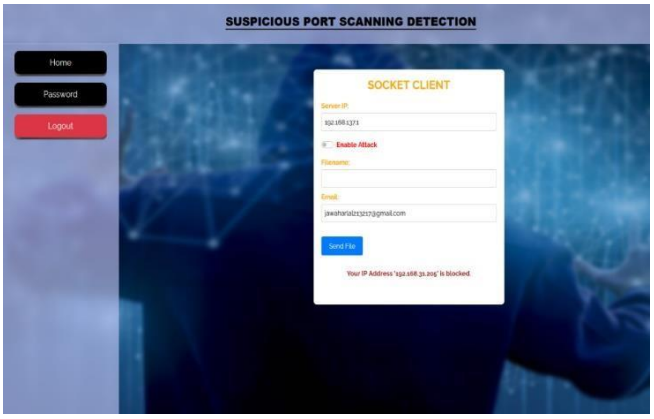
8. Model saved

Save Trained Model

```
[54]: model.save("models/MultilayerPerceptronModel.h5")
```

Fig 19. Saving Trained Model

9. Attack Detection and Prevention Including Detection Time and Response Time from Front-End



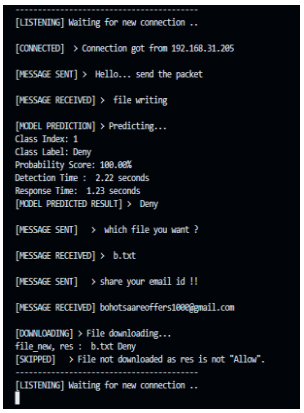


Fig 20. Attack detection and Prevention.