National College of Ireland

Configuration Manual

MSc Research Project

MSc Cyber-security

Viral Sonavadia

Student ID: X23103116

School of Computing

National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

Project Submission Sheet

Student Name:	Viral Sonavadia		
Student ID:	X23103116		
Programme:	MSc in Cybersecurity	**Year:**	2023-2024
Module:	MSc Research Project		
Lecturer:	Mr. Niall Heffernan		
Submission Due Date:	12th August 2024		
Project Title:	**HARNESSING EVOLVING MACHINE LEARNING TECHNIQUES FOR ENHANCED INTRUSION DETECTION SYSTEM**		
Word Count:	918	**Page Count:**	12

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 references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Viral Sonavadia

Date: 12th August 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

1. Introduction

The project "Harnessing Evolving Machine Learning Techniques for Enhanced Intrusion Detection System" aims to improve Intrusion Detection Systems (IDS) capabilities using advanced machine learning techniques This document describes the necessary hardware and software configuration, dataset description and deployment steps using Jupyter Notebook.

2. Overview of the program

This project involves the analysis and classification of network access data using various machine learning models. The workflow includes data pre-processing, feature engineering, training multiple machine learning models, and testing their performance on test data. The goal is to identify intrusions and correctly classify them into normal and discrete groups.

3. Hardware/software requirements

3.1. Hardware production

The following hardware settings are recommended for smooth operation.

- Processor: Intel Core i5 or higher (using i5)
- RAM: 8 GB or more (16GB)
- Storage: 100 GB free space (400GB free)
- GPU (optional, for fast model training): NVIDIA GTX 1050 Ti or better (Intel® Iris® Xe Graphics)

3.2. Software

The following software is required for the project:

- Jupyter Notebook: Used to run and write code.
- Anaconda: To manage the Python environment and dependencies.

Version Requirements:

- Jupyter Notebook: Version 6.0 or later
- Anaconda: Version 2020.11 or later

Ensure that the necessary Python libraries are installed. These include panda, numpy, mutplotlib, seaborn, scikit-sua, and tensorflow.

4. The data set

The dataset used for this project came from Kaggle. The data are divided into training and testing phases:

As they were imported on the online Jupyter

- Training information: /Train_data.csv
- Data Test: /Test_data.csv

The dataset is downloaded from Kaggle and add it to the IDS_dataset directory in project folder.

5. Implementation of Project in Jupyter

Explanation of the Code

5.1. Import Libraries

Libraries

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import warnings
     import seaborn as sns
     from sklearn.preprocessing import LabelEncoder
     warnings.filterwarnings('ignore')
```

- pandas: It is used for data manipulation and analysis.
- numpy: This library is for numerical operations.
- matplotlib.pyplot: A visual library that allows these kinds of charts to be implemented, be it in static, interactive, or animated forms.
- warnings: And this aspect is about warning messages. Moreover ,'warnings.filterwarnings ('ignore')' is the line to switches them off.
- seaborn: Doing statistical data visualization.
- LabelEncoder: It converts the categorical labels into numerical format.

5.2. Load and Inspect Data

```
data_ids_train = pd.read_csv('Train_data.csv')
data_ids_train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rat
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	0.0
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	0.6
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	0.0
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	0.0
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	0.0

- pd.read_csv: Admit a method from our operating activity and a testing one into pandas DataFrames.
- head(): It is to look at the first few rows of the DataFrames for a quick inspection.
-

5.3. Data Overview

Train data details

```
[4]: data_ids_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
```

Test data details

```
[5]: data_ids_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22544 entries, 0 to 22543
Data columns (total 41 columns):
```

Train data check

```
[6]: data_ids_train.isnull().sum()
```

Test data check

```
[7]: data_ids_test.isnull().sum()
```

- info(): This is a DataFrame method that [sic] it summary the dataframe including the number of non-null entries.
- isnull().sum(): It checks the missing value of each column.

5.4. Data Visualization

Protocol Type-wise Duration

Visualization

```
[8]: protocol_wise_duration = data_ids_train.groupby('protocol_type')['duration'].sum().reset_index()
protocol_wise_duration.head()
```

[8]:

	protocol_type	duration
0	icmp	0
1	tcp	6218779
2	udp	1466144

```
[9]: plt.figure(figsize=(8, 8))
plt.pie(protocol_wise_duration['duration'], labels=protocol_wise_duration['protocol_type'], autopct='%1.1f%%', colors=['lightgreen', 'gold'], startangle=
plt.title('Protocol Type-wise total IDS duration')
plt.show()
```

- groupby(): It takes in the 'protocol_type' and 'duration.' The function sums the 'duration' for each type.
- plt.pie(): It shows the pie chart to visualize the proportional of total IDS duration per protocol type.

Flag-wise Duration

```
[10]: flag_wise_duration = data_ids_train.groupby('flag')['duration'].sum().reset_index()
      flag_wise_duration.head()
```

[10]:

	flag	duration
0	OTH	0
1	REJ	32
2	RSTO	83312
3	RSTOS0	35702
4	RSTR	5357139

```
[11]: plt.figure(figsize=(8, 6))
      plt.bar(flag_wise_duration['flag'], flag_wise_duration['duration'], color=['skyblue', 'lightgreen', 'orange'])
      plt.xlabel('Flag')
      plt.ylabel('Total Duration')
      plt.title('Flag-wise Total IDS Duration')
      plt.grid(True)
      plt.show()
```

- groupby(): This is taking the data and grouping it by the 'flag.' Then it sums up the 'duration'.
- plt.bar(): It generates the bar chart for the total IDS duration per flag.

Protocol Type-wise Source Bytes

```
[12]: protocol_type_wise_src_bytes = data_ids_train.groupby('protocol_type')['src_bytes'].sum().reset_index()
      protocol_type_wise_src_bytes.head()
```

[12]:

	protocol_type	src_bytes
0	icmp	557254
1	tcp	612166701
2	udp	213231

```
[13]: plt.figure(figsize=(8, 6))
      plt.plot(protocol_type_wise_src_bytes['protocol_type'], protocol_type_wise_src_bytes['src_bytes'], marker='o', linestyle='-', color='g')
      plt.xlabel('Protocol Type')
      plt.ylabel('Total Source Bytes')
      plt.title('Protocol Type-wise Total Source Bytes')
      plt.grid(True)
      plt.show()
```

- groupby(): We are referring to the 'protocol_type' and running the 'src_bytes' total.
- plt.plot(): It levels the line plot to indicate the entire source bytes per protocol type.

Protocol Type-wise Destination Bytes

```
[14]: protocol_type_wise_dst_bytes = data_ids_train.groupby('protocol_type')['dst_bytes'].sum().reset_index()
      protocol_type_wise_dst_bytes.head()
```

[14]:

	protocol_type	dst_bytes
0	icmp	0
1	tcp	87755181
2	udp	211433

```
[15]: plt.figure(figsize=(8, 6))
      plt.fill_between(protocol_type_wise_dst_bytes['protocol_type'], protocol_type_wise_dst_bytes['dst_bytes'], color="skyblue", alpha=0.4)
      plt.plot(protocol_type_wise_dst_bytes['protocol_type'], protocol_type_wise_dst_bytes['dst_bytes'], marker='o', color='blue', alpha=0.6)
      plt.xlabel('Protocol Type')
      plt.ylabel('Total Destination Bytes')
      plt.title('Protocol Type-wise Total Destination Bytes')
      plt.grid(True)
      plt.show()
```

- groupby(): It actually means that the groups of data are made by 'protocol_type' and they are summed up by 'dst_bytes'.
- plt.fill_between(): It shows an area plot of the total destination bytes per protocol type.

Class Distribution

```
[16]: total_class_counts = data_ids_train['class'].value_counts().reset_index()
      total_class_counts
```

[16]:

	class	count
0	normal	13449
1	anomaly	11743

```
[17]: plt.figure(figsize=(8, 8))
      wedges, texts, autotexts = plt.pie(total_class_counts['count'], labels=total_class_counts['class'], autopct='%1.1f%%', colors=['orange', 'lightgreen'], s
      centre_circle = plt.Circle((0, 0), 0.70, fc='white')
      fig = plt.gcf()
      fig.gca().add_artist(centre_circle)
      plt.title('Total IDS class count')
      plt.axis('equal')
      plt.show()
```

Total IDS class count

- value_counts(): The incidence of every class is aggregated respectively.
- plt.pie(): This even pegs the pie chart to teach the elegance of each class.\

5.5. Pre-processing

Label Encoding

Data processing

```
[18]: label_encoder_factor = LabelEncoder()

[19]: target_encode_columns = ['protocol_type', 'service', 'flag']

[20]: for column_data in target_encode_columns:
          data_ids_train[column_data] = label_encoder_factor.fit_transform(data_ids_train[column_data])

[21]: data_ids_train.head()
```

[21]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate
0	0	1	19	9	491	0	0	0	0	0	...	25	0.17	0.03
1	0	2	41	9	146	0	0	0	0	0	...	1	0.00	0.60
2	0	1	46	5	0	0	0	0	0	0	...	26	0.10	0.05
3	0	1	22	9	232	8153	0	0	0	0	...	255	1.00	0.00
4	0	1	22	9	199	420	0	0	0	0	...	255	1.00	0.00

5 rows × 42 columns

- mean(): This operation is about getting the mean of the columns.
- fit_transform(): The transformation is taken out on the striking features.

Mapping Class Labels

```
[22]: class_map = {
          'normal': 0,
          'anomaly': 1
      }

[23]: data_ids_train['class'] = data_ids_train['class'].map(class_map)
```

- map(): The mapping of class labels to binary values (0 for 'normal', 1 for 'anomaly') is carried out.

Dropping Unnecessary Columns

```
[27]: data_ids_train.drop(['num_outbound_cmds'], axis=1, inplace=True)
      data_ids_train.drop(['is_host_login'], axis=1, inplace=True)
      data_ids_train.head()
```

[27]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate
0	0	1	19	9	491	0	0	0	0	0	...	25	0.17	0.03
1	0	2	41	9	146	0	0	0	0	0	...	1	0.00	0.60
2	0	1	46	5	0	0	0	0	0	0	...	26	0.10	0.05
3	0	1	22	9	232	8153	0	0	0	0	...	255	1.00	0.00
4	0	1	22	9	199	420	0	0	0	0	...	255	1.00	0.00

- drop(): It uses the built-in drop function to remove the columns you indicated from the DataFrame.

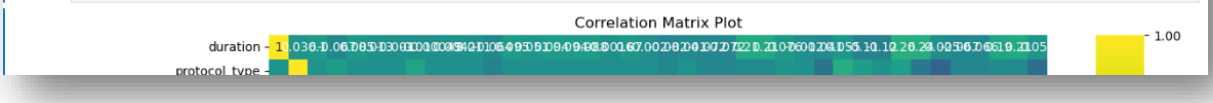
Correlation Matrix

```
[28]: correlation_matrix_val = data_ids_train.corr()
      correlation_matrix_val
```

[28]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count
duration	1.000000	0.036421	0.101301	-0.066634	0.084864	0.013258	-0.001012	-0.010358	-0.000486	0.004202	...	-0.112530
protocol_type	0.036421	1.000000	0.025322	0.094926	-0.001286	-0.004734	-0.001123	0.176420	-0.000794	-0.011589	...	0.104292
service	0.101301	0.025322	1.000000	-0.295491	0.008554	-0.013782	-0.006896	0.088169	0.011325	-0.056165	...	-0.409442
flag	-0.066634	0.094926	-0.295491	1.000000	-0.006599	0.027606	-0.006593	0.068693	0.004744	0.067027	...	0.584087
src_bytes	0.084864	-0.001286	0.008554	-0.006599	1.000000	0.003611	-0.000090	-0.000916	-0.000062	0.000995	...	-0.008520
dst_bytes	0.013258	-0.004734	-0.013782	0.027606	0.003611	1.000000	-0.000350	-0.003586	0.000345	0.002539	...	-0.000980
land	-0.001012	-0.001123	-0.006896	-0.006593	-0.000090	-0.000350	1.000000	-0.000813	-0.000056	-0.000819	...	-0.008743
wrong_fragment	-0.010358	0.176420	0.088169	0.068693	-0.000916	-0.003586	-0.000813	1.000000	-0.000575	-0.008386	...	-0.047256
urgent	-0.000486	-0.000794	0.011325	0.004744	-0.000062	0.000345	-0.000056	-0.000575	1.000000	0.002346	...	-0.006324
hot	0.004202	-0.011589	-0.056165	0.067027	0.000995	0.002539	-0.000819	-0.008386	0.002346	1.000000	...	-0.048495
num_failed_logins	0.011108	-0.003305	0.029601	-0.010920	-0.000260	0.005197	-0.000234	-0.002392	-0.000165	0.004893	...	-0.022315
logged_in	-0.063703	-0.101810	-0.138824	0.587882	-0.002040	0.012704	-0.007196	-0.073674	0.007801	0.113115	...	0.624839

```
[29]: plt.figure(figsize=(14, 14))
      sns.heatmap(correlation_matrix_val, annot=True, cmap='viridis', vmin=-1, vmax=1)
      plt.title('Correlation Matrix Plot')
      plt.show()
```



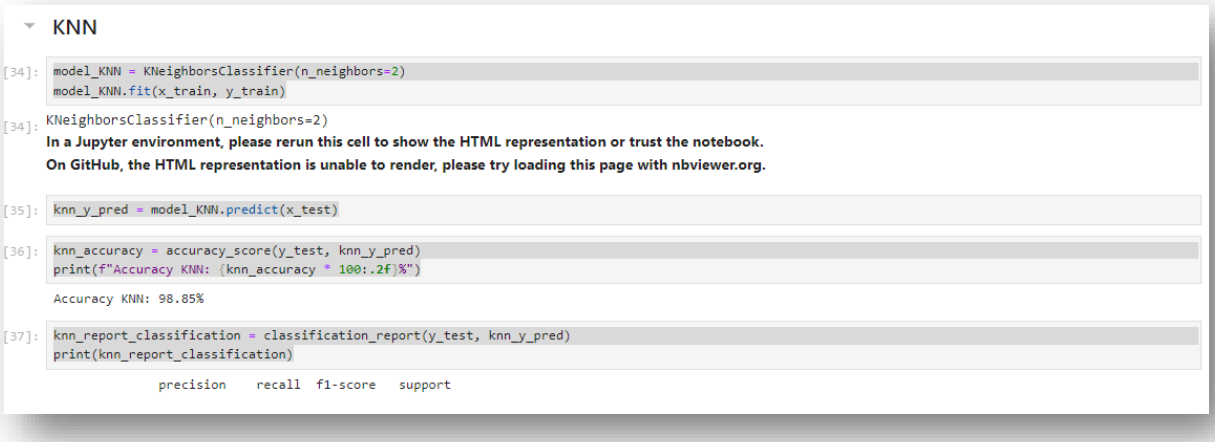
- corr(): It is no surprise that it gives the correlation matrix.
- sns.heatmap(): It forms a heatgrams that are meant to be visually appealing.

6. Model Training and Evaluation

Splitting Data

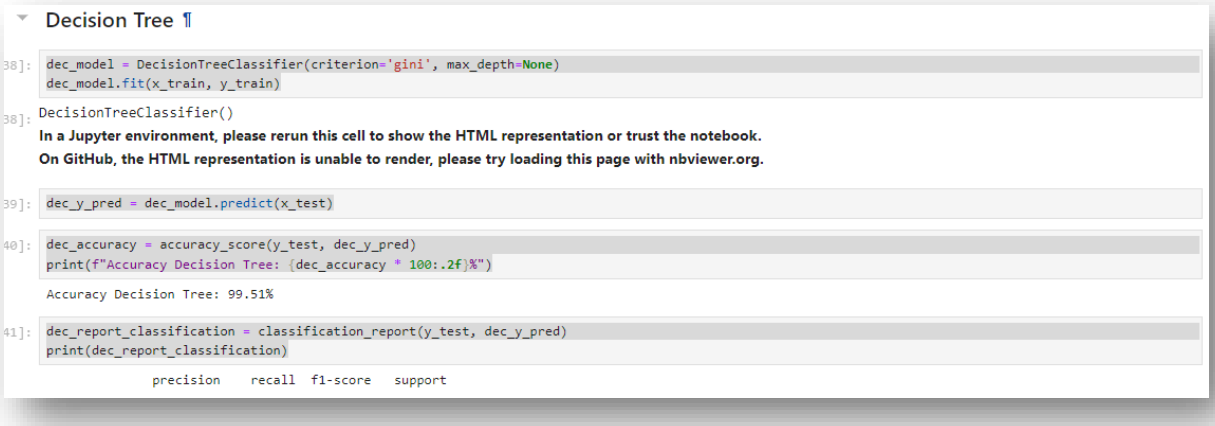
- train_test_split(): Refers to the process of dividing the dataset into two groups, the training set and the testing set.

K-Nearest Neighbors (KNN)



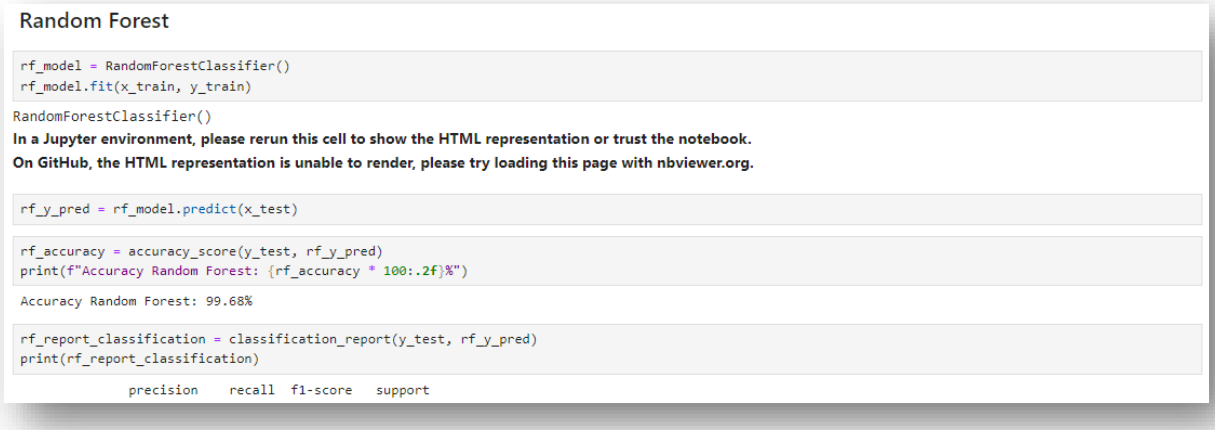
- KNeighborsClassifier(): Here, we start KNN with the actual KNN classifier.
- fit(): Our model is ready for training.
- predict(): We will be getting the model's predictions over the corresponding original data set.
- accuracy_score(): This function is used to find out how many times our model was correct in its predictions.
- classification_report(): The care, coverage, and F1-metric are all included here.

Decision Tree



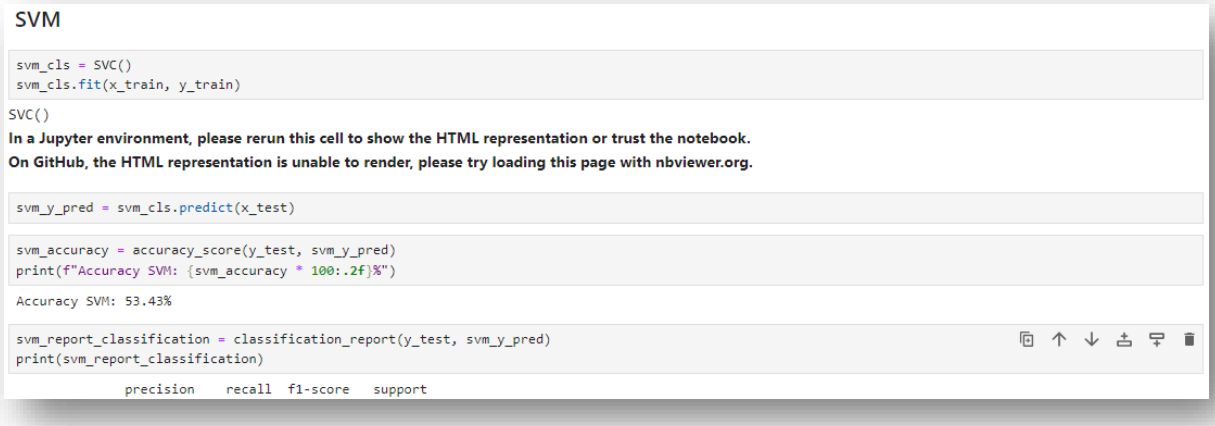
- DecisionTreeClassifier(): This part of code illustrated making a new classifier based on a Decision tree.

Random Forest



- RandomForestClassifier(): The Random Forest classifier is initialized.
- fit(), predict(), accuracy_score(), and classification_report(): These are the same functions that are used in KNN.

Support Vector Machine (SVM)



References:

Alhajjar, E., Maxwell, P. and Bastian, N., 2021. Adversarial machine learning in network intrusion detection systems. *Expert Systems with Applications*, *186*, p.115782.

Alotaibi, A. and Rassam, M.A., 2023. Adversarial machine learning attacks against intrusion detection systems: A survey on strategies and defense. *Future Internet*, *15*(2), p.62.

Mahanta, K. and Maringanti, H.B., 2024. Machine learning approaches for intrusion detection. *Cognitive Machine Intelligence: Applications, Challenges, and Related Technologies*, p.199.

Moizuddin, M.D. and Jose, M.V., 2022. A bio-inspired hybrid deep learning model for network intrusion detection. *Knowledge-based systems*, *238*, p.107894.

Okoli, U.I., Obi, O.C., Adewusi, A.O. and Abrahams, T.O., 2024. Machine learning in cybersecurity: A review of threat detection and defense mechanisms. *World Journal of Advanced Research and Reviews*, *21*(1), pp.2286-2295.