National College of Ireland

Configuration Manual

MSc Research Project

MSc in Cyber Security (MSCCYB1)

Shivam Thakur

Student ID: x21220891

School of Computing

National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shivam Thakur
Student ID:	X21220891
Programme:	MSc in Cyber Security **Year:** 2022-2023
Module:	MSc Research Project – Configuration Manual
Lecturer:	Vikas Sahni
Submission Due Date:	14th August 2023
Project Title:	Configuration Manual
Word Count:	**837** **Page Count: 8**

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:	SHIVAM THAKUR
Date:	8th August 2023

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

Shivam Thakur
Student ID: x21220891

1 Introduction

This is a configuration manual which will help understand how to setup the project and replicate the results. All the hardware specifications, software specifications, code libraries that were used for the implementation of the artefact have been mentioned in this manual. The suggested proposal and the research conducted significantly relies upon this configuration manual.

2 Hardware Specifications

- Central Processing Unit (CPU) - AMD Ryzen 7 5800H with Radeon Graphics @ 3.20 GHz, x64 based processor.
- System Memory (RAM) – 16.0 GB (15.4 GB Usable)
- Operating System (OS) – Windows 11 Home, 64-bit Operating System,
- OS Version – 22H2
- OS Build - 22621.1992
- Storage – 1 TB Solid State Drive.
- Graphics Processing Unit (GPU) – Nvidia GeForce RTX 3070 Laptop GPU
- Laptop Manufacturer - Acer
- Laptop Model – Nitro AN515-45

3 Software Specifications

For the purposes of this project, the coding language used was Python and the Integrated Development Environment (IDE) used was Jupyter Notebook. For the purposes of machine learning, data processing, computation, visualization as well as analysis several packages of Python were used. In order to access both Python and Jupyter Notebook we installed the Anaconda Navigator which is a popular software package that comes with Python, Jupyter Notebook and several libraries.

- Anaconda Navigator (version 2.4.2) [1]
- Jupyter Notebook (version 6.5.4) – pre-packaged with Anaconda.
- Python – 3.11.3 – pre-packaged with Anaconda.

[1] https://www.anaconda.com/

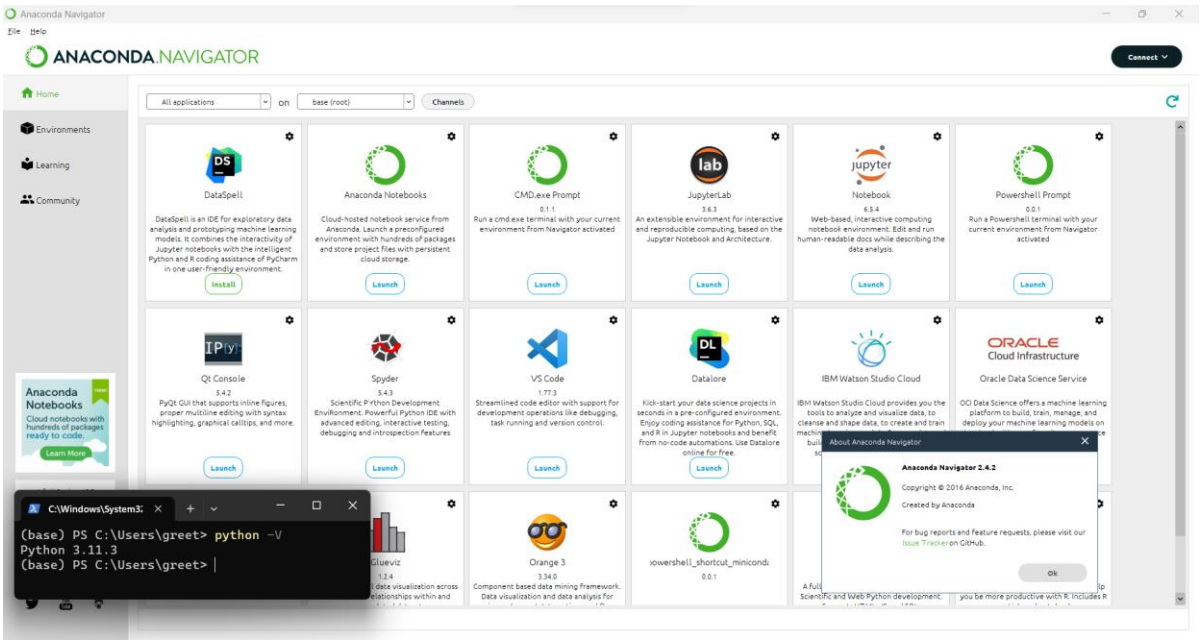

Figure 1 - Anaconda Navigator, Python and Jupyter Notebook

4 Dataset Download, Directory and Execution

The NSL-KDD Dataset [2] needs to be downloaded from the official website of University of New Brunswick and saved in a folder. The full file path must be noted so that it can be loaded into the notebook and code. Once the dataset is downloaded, the notebook MalwareDetection.ipynb must be opened and each block of code must be run.

Note: This line needs to be changed to represent the file path on your system:

```
In [4]: data_train = pd.read_csv("/Shivam/NCI Coursework/Thesis/NSL-KDD/KDDTrain+.txt") #Read the dataset
```

Figure 2 - Change this to the directory where dataset is stored.

5 Python Libraries Used

Anaconda Navigator comes installed with several Python libraries however, manual install maybe required for the below libraries:

- Numpy – 1.24.3
- Pandas – 1.5.3
- Matplotlib – 3.7.1
- Seaborn – 0.12.2
- Tensorflow – 2.13.0
- Xgboost – 1.7.6
- Scikit-learn – 1.2.2

[2] https://www.unb.ca/cic/datasets/nsl.html

```
In [1]: import numpy as np # NumPy Library is for numerical calculations.
        import pandas as pd # Pandas will help in manipulating data as a DataFrame.
        import warnings # This module will manage warnings.
        import matplotlib.pyplot as plt # This library will help in plotting graphs
        import seaborn as sns # Library for visualizations
        import tensorflow as tf # Tensorflow will help in ML
        from tensorflow.keras import regularizers # Imports regularizers from TF
        import xgboost as xgb # Imports XGBoost library
        from sklearn.decomposition import PCA # Principal Component Analysis
        from sklearn import tree # Decision Tree Algorithms
        from sklearn.naive_bayes import GaussianNB # naive Bayesian Gaussian classifier
        from sklearn.linear_model import LogisticRegression # Logistic Regression
        from sklearn.neighbors import KNeighborsClassifier # KNN
        from sklearn.tree import DecisionTreeClassifier # Decision Tree
        from sklearn.preprocessing import RobustScaler # Data Scaling
        from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor # RF
        from sklearn.model_selection import train_test_split # Split Train and Test sets
        from sklearn import svm # SVM
        from sklearn import metrics # Model Performance
        from datetime import datetime #Date and time runs
```

Figure 3 : Libraries Imported

6 Timestamping

Before starting the experiment, we will grab the time and store it as the start time of the experiment. We will repeat this at the end of the experiment to grab the time the experiment ended. We will calculate the difference as the runtime.

```
In [2]: now1 = datetime.now()
        start_time = now1.strftime("%H:%M:%S")
        start_time_formatted = now1.strptime(start_time, '%H:%M:%S')
```

Figure 4 : Start Time

```
In [23]: now2 = datetime.now()
         end_time = now2.strftime("%H:%M:%S")
         end_time_formatted = now2.strptime(end_time, '%H:%M:%S')

         diff = end_time_formatted - start_time_formatted

         print("The experiment was started today at {}.".format(start_time))
         print("The experiment was ended today at {}.".format(end_time))
         print('The runtime of the experiment was {} seconds.'.format(diff.total_seconds()))

         The experiment was started today at 01:48:45.
         The experiment was ended today at 01:50:16.
         The runtime of the experiment was 91.0 seconds.
```

Figure 5 : End Time and Runtime - Code + Output

7 Data Processing

In this phase we will refine the dataset. We will first import the dataset into the code and see what the first few values look like. We find out that there are no column labels, so we will assign them manually and then re-display to see if they have been applied. We will also perform data scaling on the dataset so that it can be fed into the AI models in a format that can be processed quickly and efficiently. Data splitting into test and train partitions also happens here

In [4]:
```
data_train = pd.read_csv("/Shivam/NCI Coursework/Thesis/NSL-KDD/KDDTrain+.txt") #Read the dataset
```

In [5]:
```
data_train.head() #Check the data
```

Out[5]:

	0	tcp	ftp_data	SF	491	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.18	2	2.1	0.00	0.00.1	0.00.2	0.00.3	1
0	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	1	0.0	0.0	0.0	0.0	0
1	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123	6	1.0	1.0	0.0	0.0	0
2	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5	5	0.2	0.2	0.0	0.0	1
3	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	30	32	0.0	0.0	0.0	0.0	1
4	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	121	19	0.0	0.0	1.0	1.0	0

In [6]:
```
columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot'
,'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creations'
,'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','serror_rate'
,'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_hos
,'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_r
,'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','outcome','level'])

# Making a list of columns.
```

In [7]:
```
# Assign name for columns
data_train.columns = columns
data_train.head()
```

Out[7]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell
0	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0
1	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0
2	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0
3	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0
4	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0

```
# Separation of data into training and test sets for classification
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Figure 6 : Dataset Processing

8 Model Analysis

Now the models will run and display the evaluation parameters, followed by the ensemble parameter calculations:

In [17]:
```
kernal_evals = dict()
def evaluate_classification(model, name, X_train, X_test, y_train, y_test):
    train_accuracy = metrics.accuracy_score(y_train, model.predict(X_train))
    test_accuracy = metrics.accuracy_score(y_test, model.predict(X_test))

    train_precision = metrics.precision_score(y_train, model.predict(X_train))
    test_precision = metrics.precision_score(y_test, model.predict(X_test))

    train_recall = metrics.recall_score(y_train, model.predict(X_train))
    test_recall = metrics.recall_score(y_test, model.predict(X_test))

    train_f1score = metrics.f1_score(y_train, model.predict(X_train))
    test_f1score = metrics.f1_score(y_test, model.predict(X_test))

    kernal_evals[str(name)] = [train_accuracy, test_accuracy, train_f1score, test_f1score, train_precision, test_precision, train
    print("Training Accuracy for " + str(name) + " is {:.2f} and Test Accuracy for ".format(train_accuracy*100) + str(name) + " i
          .format(test_accuracy*100))
    print("Training F1 score for " + str(name) + " is {:.2f} and Test F1 Score for ".format(train_f1score*100) + str(name) + " is
          .format(test_f1score*100))
    print("Training Precision for " + str(name) + " is {:.2f} and Test Precision for ".format(train_precision*100) + str(name) +
          .format(test_precision*100))
    print("Training Recall for " + str(name) + " is {:.2f} and Test Recall for ".format(train_recall*100) + str(name) + " is {:.2
          .format(test_recall*100))

    actual = y_test
    predicted = model.predict(X_test)
    confusion_matrix = metrics.confusion_matrix(actual, predicted)
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ['normal', 'attack'])

    fig, ax = plt.subplots(figsize=(5,5))
    ax.grid(False)
    cm_display.plot(ax=ax)
```

Figure 7 : Code for analysis

In [18]:
```
gnb = GaussianNB().fit(x_train, y_train)
evaluate_classification(gnb, "Gaussian Naive Bayes", x_train, x_test, y_train, y_test)
```

```
Training Accuracy for Gaussian Naive Bayes is 91.80 and Test Accuracy for Gaussian Naive Bayes is 91.61
Training F1 score for Gaussian Naive Bayes is 91.03 and Test F1 Score for Gaussian Naive Bayes is 90.89
Training Precision for Gaussian Naive Bayes is 92.63 and Test Precision for Gaussian Naive Bayes is 92.53
Training Recall for Gaussian Naive Bayes is 89.48 and Test Recall for Gaussian Naive Bayes is 89.30
```

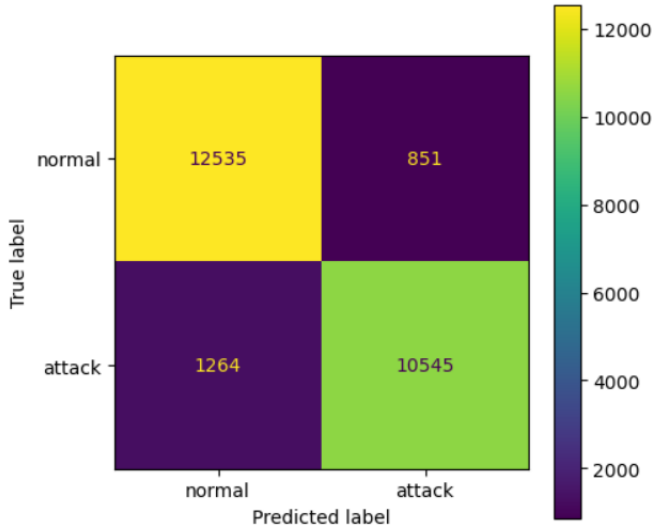

Figure 8 : Naive Bayes

In [19]:
```
knn = KNeighborsClassifier(n_neighbors=30).fit(x_train, y_train)
evaluate_classification(knn, "K Nearest Neighbours", x_train, x_test, y_train, y_test)
```

```
Training Accuracy for K Nearest Neighbours is 98.86 and Test Accuracy for K Nearest Neighbours is 98.82
Training F1 score for K Nearest Neighbours is 98.77 and Test F1 Score for K Nearest Neighbours is 98.73
Training Precision for K Nearest Neighbours is 99.06 and Test Precision for K Nearest Neighbours is 99.04
Training Recall for K Nearest Neighbours is 98.49 and Test Recall for K Nearest Neighbours is 98.43
```

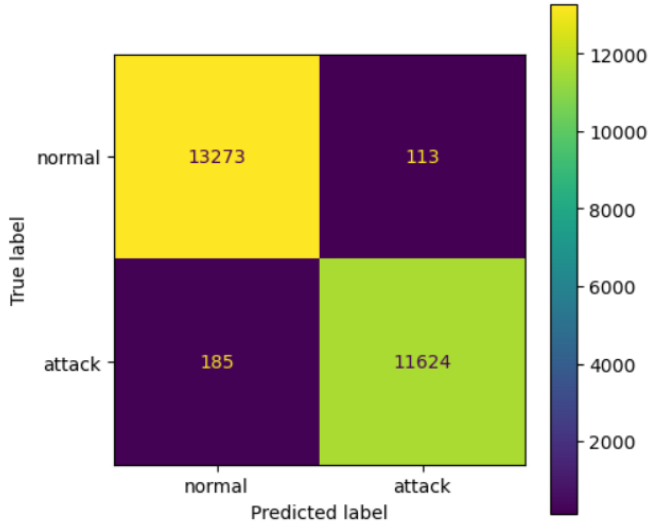

Figure 9 : KNN

```
In [20]: lr = LogisticRegression().fit(x_train, y_train)
         evaluate_classification(lr, "Logistic Regression", x_train, x_test, y_train, y_test)
```

```
Training Accuracy for Logistic Regression is 88.00 and Test Accuracy for Logistic Regression is 87.66
Training F1 score for Logistic Regression is 87.67 and Test F1 Score for Logistic Regression is 87.43
Training Precision for Logistic Regression is 83.89 and Test Precision for Logistic Regression is 83.66
Training Recall for Logistic Regression is 91.81 and Test Recall for Logistic Regression is 91.56
```

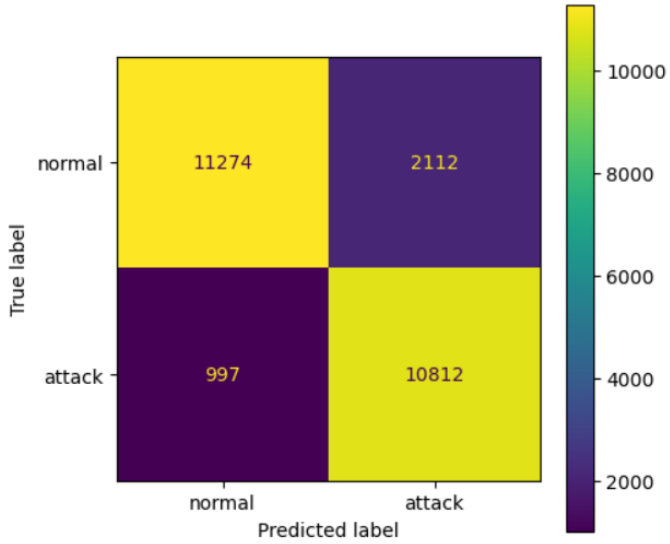

Figure 10 : LR

```
The final parameters for the ensemble are as follows:
Training Accuracy = 92.89
Testing Accuracy = 92.69
Training F1 score = 92.49
Testing F1 score = 92.35
Training Precision = 91.86
Testing Precision = 91.74
Training Recall = 93.26
Testing Recall = 93.10
```

Figure 11 : Ensemble