Configuration Manual

MSc Research Project
MSc Cybersecurity

Chaitanya Londhe
Student ID: X19212518

School of Computing
National College of Ireland

Supervisor: Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chaitanya Anand Londhe
Student ID:	X19212518
Programme:	M.Sc. Cybersecurity **Year:** 2020-2021
Module:	Academic Internship
Lecturer:	Mr. Imran Khan
Submission Due Date:	16th August 2021
Project Title:	Applying Machine learning and Deep Learning Techniques for Improvement in Network Intrusion Detection System
Word Count:	2570 **Page Count:** 18

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: Chaitanya Londhe

Date: 16/08/2021

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

Chaitanya Londhe
Student ID: X19212518

1 Introduction

Due to recent expansion, and the advancement in growth of the Internet and digital technologies over the past decade, network security is a crucial field of research. It employs methods, such as antivirus software, firewalls, and intrusion detection systems to protect the integrity of the system and all its connected characteristics within the Internet. One of them is a threat detection component that allows the needed security through the constant surveillance of network traffic for disturbing or uneasy behavior which is Network-based intrusion detection. During the last 10 years, professionals have created several Machine Learning (ML) and Deep Learning (DL) techniques to improve the effectiveness of the Network Intrusion Detection System (NIDS) in recognizing malware assaults. There is indeed a significant amount of area for investigation into adding ML and DL approaches to NIDS to successfully identify perpetrators on the network. This research is therefore exploitable across NIDS.

2 Tools used for research implementation:

For a long time, Python is now the most important language for developers of machine learning and artificial intelligence. Python offers a broad variety of flexibility and functions for developers to increase not only their usability but also their development consistency. Accordingly, Python is utilized to implement this project as well. It has employed library package like Keras, Scikit-Learn, TensorFlow, etc. Python is a highly utilized language which uses mathematical formulas and maps to analyze data. In this project, the machine learning models are applied on the newly generated dataset. And the deep learning models are performed and then the evaluation of all the models was carried out. All these steps are carried out using python language in the Google Colab tool, since it has very user friendly interface and is very easy to use. The implementation steps are as follows:

3 Importing Libraries

3.1 Before starting with our implementation, the very first step is to import all the required libraries for model building. A library is basically a set of methods and functions that let us execute a lot of activities without writing a code for it.

3.2 In this project, numerous libraries are installed and imported like *pandas, sklearn, numpy, matplotlib, itertools,* etc. for using it for various purposes.

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import (precision_score, recall_score,f1_score, accuracy_score,mean_squared_error,mean_absolute_error)
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import Normalizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from math import *
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
import itertools
import io
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import classification_report
from sklearn.svm import OneClassSVM
from sklearn.pipeline import Pipeline
```

Fig 1: Importing libraries

4 Importing the Dataset

This dataset was initially produced for the analysis of DDoS data by the University of New Brunswick. The said dataset came from 2018, and will not be modified in the future, although fresh dataset versions are available. The dataset itself was derived on university logfiles, which reported several DoS assaults during the public timeframe. The Label column is the most essential part of the data when constructing machine-learning notebooks, as it indicates whether the packets that have been delivered are or are not malicious. In the dataset there are eighty columns, each of which represents an IDS logging system entry in place by the University of New Brunswick. The concepts 'intrusion' and 'detection system' make an IDS. Since its system categorizes traffic forward and behind, columns are available for both. All the variables in the dataset are numerical accept the Label variable which is categorical. A network connection is a sequence of packets that begin and terminate at a certain period during which the data travels from the source IP to the destination IP address where every connection is either labelled as benign or as malicious with just one particular form of assault in this dataset. Following are the steps for importing and processing of the dataset.

4.1 In this step, the dataset for NIDS consisting of the complete information about incoming and outgoing packets, is imported and since it is a CSV file, and is stored in a tabular format.
4.2 As the dataset is very complex and large, it is converted into a pickle so that it consumes less amount of memory.

```
class dataset:
    pass
sample_data = pd.read_csv("CSE-CIC-IDS2018.csv")
sample_data.to_pickle('CSE-CIC-IDS2018.pkl')
```

Fig 2: Importing the Dataset

5 Data Pre-processing

Pre-processing of the data is the primary step to be taken before starting with the process in the realm of machine learning. Data pre-processing is essentially used to convert and transform unprocessed and raw data to a much better and more comprehensible format. Real world data may generally be partial, irregular, incorrect, unstructured, and may be missing. Data pre-processing is being used to circumvent all this. It supports cleaning, formatting, organizing, and preparing raw data for implementation in the model of developing machine learning. Data Pre-processing cannot be carried out in a single process and is thus dispersed in many phases.

5.1 Further, that pickle is stored in a data frame. Then all the integer variables in that data frame are converted into float (continuous values) to avoid later disturbance in the implementation regarding and execution of different data types. Also, all the Na values are dropped (if any).[1]

[1] https://datatofish.com/integer-to-float-dataframe/

```python
df = pd.read_pickle('CSE-CIC-IDS2018.pkl')
df["Flow Pkts/s"] = pd.to_numeric(df["Flow Pkts/s"], errors='coerce')
df['Protocol'] = df['Protocol'].astype(float)
df['Dst Port'] = df['Dst Port'].astype(float)
df['Tot Fwd Pkts'] = df['Tot Fwd Pkts'].astype(float)
df['Tot Bwd Pkts'] = df['Tot Bwd Pkts'].astype(float)
df['TotLet Fwd Pkts'] = df['TotLen Fwd Pkts'].astype(float)
df['TotLen Bwd Pkts'] = df['TotLen Bwd Pkts'].astype(float)
df['Flow Duration'] = df['Flow Duration'].astype(float)

df.dropna(inplace=True)
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6537 entries, 0 to 6557
Data columns (total 81 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Dst Port           6537 non-null   float64
 1   Protocol           6537 non-null   float64
 2   Timestamp          6537 non-null   object
 3   Flow Duration      6537 non-null   float64
 4   Tot Fwd Pkts       6537 non-null   float64
 5   Tot Bwd Pkts       6537 non-null   float64
 6   TotLen Fwd Pkts    6537 non-null   float64
 7   TotLen Bwd Pkts    6537 non-null   float64
 8   Fwd Pkt Len Max    6537 non-null   float64
 9   Fwd Pkt Len Min    6537 non-null   float64
 10  Fwd Pkt Len Mean   6537 non-null   float64
 11  Fwd Pkt Len Std    6537 non-null   float64
 12  Bwd Pkt Len Max    6537 non-null   float64
 13  Bwd Pkt Len Min    6537 non-null   float64
 14  Bwd Pkt Len Mean   6537 non-null   float64
 15  Bwd Pkt Len Std    6537 non-null   float64
 16  Flow Byts/s        6537 non-null   float64
 17  Flow Pkts/s        6537 non-null   float64
 18  Flow IAT Mean      6537 non-null   float64
 19  Flow IAT Std       6537 non-null   float64
 20  Flow IAT Max       6537 non-null   float64
 21  Flow IAT Min       6537 non-null   float64
```

Fig 3: Converting int values into float

```python
df.drop('Flow Pkts/s', inplace=True, axis=1)
df.drop('Timestamp', inplace=True, axis=1)
df.drop('Flow Byts/s', inplace=True, axis=1)
df
```

	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	Bwd Pkt Len Max	Bwd Pkt Len Min	Bwd Pkt Len Mean	Bwd Pkt Len Std	Flow I Me
0	443.0	6.0	141385.0	9.0	7.0	553.0	3773.0	202.0	0.0	61.444444	87.534438	1460.0	0.0	539.000000	655.432936	9425.6666
1	49684.0	6.0	281.0	2.0	1.0	38.0	0.0	38.0	0.0	19.000000	26.870058	0.0	0.0	0.000000	0.000000	140.5000
2	443.0	6.0	279824.0	11.0	15.0	1086.0	10527.0	385.0	0.0	98.727273	129.392497	1460.0	0.0	701.800000	636.314186	11192.9600
3	443.0	6.0	132.0	2.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	132.0000
4	443.0	6.0	274016.0	9.0	13.0	1285.0	6141.0	517.0	0.0	142.777778	183.887722	1460.0	0.0	472.384615	611.180489	13048.3809
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6553	8080.0	6.0	10239.0	3.0	4.0	326.0	129.0	326.0	0.0	108.666667	188.216188	112.0	0.0	32.250000	53.767245	1706.5000
6554	8080.0	6.0	474.0	2.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	474.0000
6555	8080.0	6.0	10860.0	3.0	4.0	326.0	129.0	326.0	0.0	108.666667	188.216188	112.0	0.0	32.250000	53.767245	1810.0000
6556	8080.0	6.0	487.0	2.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	487.0000
6557	8080.0	6.0	11398.0	3.0	4.0	326.0	129.0	326.0	0.0	108.666667	188.216188	112.0	0.0	32.250000	53.767245	1899.6666

6537 rows × 78 columns

Fig 4: Drop unnecessary columns

5.2 Next, the ou tput variable, which is categorical, is then converted into a binary column into 0 and 1. So, the category ‘Benign’ is converted into a 0 and ‘Bot’ is converted into 1.

```
[8] ds = df.replace('Benign', 0)

dataset_m = ds.replace('Bot', 1)
dataset_m
```

URG Flag Cnt	CWE Flag Count	ECE Flag Cnt	Down/Up Ratio	Pkt Size Avg	Fwd Seg Size Avg	Bwd Seg Size Avg	Fwd Byts/b Avg	Fwd Pkts/b Avg	Fwd Blk Rate Avg	Bwd Byts/b Avg	Bwd Pkts/b Avg	Bwd Blk Rate Avg	Subflow Fwd Pkts	Subfl F By
0.0	0.0	1.0	0.0	270.375000	61.444444	539.000000	0.0	0.0	0.0	0.0	0.0	0.0	9.0	553
0.0	0.0	0.0	0.0	25.333333	19.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	2.0	38
0.0	0.0	1.0	1.0	446.653846	98.727273	701.800000	0.0	0.0	0.0	0.0	0.0	0.0	11.0	1086
0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0
0.0	0.0	1.0	1.0	337.545455	142.777778	472.384615	0.0	0.0	0.0	0.0	0.0	0.0	9.0	1285
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
0.0	0.0	1.0	1.0	65.000000	108.666667	32.250000	0.0	0.0	0.0	0.0	0.0	0.0	3.0	326
0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0
0.0	0.0	1.0	1.0	65.000000	108.666667	32.250000	0.0	0.0	0.0	0.0	0.0	0.0	3.0	326
0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0
0.0	0.0	1.0	1.0	65.000000	108.666667	32.250000	0.0	0.0	0.0	0.0	0.0	0.0	3.0	326

Fig 5: Converting output variable into 0s and 1s

5.3 After this conversion, the data is then standardized and rescaled to get a good shape of distribution of the dataset.

```
#Standardize Data
from sklearn.preprocessing import StandardScaler
from numpy import set_printoptions
scaler=StandardScaler()
rescaled_data=scaler.fit_transform(dataset_m)
set_printoptions(precision=3)
print(rescaled_data[0:5,:])
```

```
  -3.169e-02 -9.074e-02  3.540e+00  2.723e+00  3.434e+00  3.544e+00
   0.000e+00 -2.155e-01  1.074e+00  1.005e+00 -9.436e-01 -5.106e-02
   0.000e+00  1.074e+00 -8.121e-01  2.751e+00  1.782e-01  3.947e+00
   0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00
   9.175e-02  9.558e-02  6.536e-02  1.087e-01  4.509e-01 -1.589e-01
   7.103e-02  1.750e-01 -9.647e-02 -8.843e-02 -1.081e-01 -7.044e-02
  -2.256e-01 -1.099e-01 -2.300e-01 -2.150e-01 -1.909e+00  9.558e-02]
 [ 5.150e+00 -7.308e-02 -2.990e-01 -1.290e-01 -1.344e-01 -1.183e-01
  -6.378e-02 -6.564e-01 -9.360e-02 -5.826e-01 -6.270e-01 -4.233e-01
  -1.223e-01 -3.496e-01 -4.365e-01 -1.691e-01 -2.365e-01 -2.371e-01
  -2.666e-02 -2.976e-01 -1.972e-01 -2.267e-01 -2.351e-01 -5.245e-02
  -2.682e-01 -1.852e-01 -2.072e-01 -2.122e-01 -5.225e-02  4.641e+00
   0.000e+00  0.000e+00  0.000e+00 -1.199e-01 -1.704e-01 -5.794e-02
   6.860e-02 -9.074e-02 -5.570e-01 -3.089e-01 -4.868e-01 -3.264e-01
   0.000e+00  4.641e+00 -9.309e-01 -9.947e-01  1.060e+00 -5.106e-02
   0.000e+00 -9.309e-01 -8.121e-01 -2.756e-01 -5.826e-01 -3.496e-01
   0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00
  -1.290e-01 -1.183e-01 -1.344e-01 -6.378e-02 -8.340e-01 -1.755e-01
  -8.807e-02  1.750e-01 -9.647e-02 -8.843e-02 -1.081e-01 -7.044e-02
  -2.256e-01 -1.099e-01 -2.300e-01 -2.150e-01 -1.909e+00 -1.183e-01]
 [-9.347e-01 -7.308e-02 -2.897e-01  1.548e-01  3.317e-01  3.169e-01
   4.174e-01  1.064e+00 -9.360e-02  8.465e-01  4.271e-01  3.983e+00
  -1.223e-01  5.245e+00  4.341e+00 -1.636e-01 -2.282e-01 -2.264e-01
  -2.667e-02 -2.883e-01 -1.885e-01 -2.167e-01 -2.243e-01 -5.253e-02
  -2.588e-01 -1.789e-01 -1.963e-01 -1.984e-01 -5.225e-02 -2.155e-01
   0.000e+00  0.000e+00  0.000e+00  2.170e-01  4.479e-01 -1.000e-01
  -3.157e-02 -9.074e-02  3.540e+00  4.985e+00  4.227e+00  5.184e+00
   0.000e+00 -2.155e-01  1.074e+00  1.005e+00 -9.436e-01 -5.106e-02
   0.000e+00  1.074e+00  9.245e-01  4.929e+00  8.465e-01  5.245e+00
```

Fig 6: Standardizing the data[2]

---

[2] https://elearning.dbs.ie/pluginfile.php/1301058/mod_resource/content/1/Data%2BDescriptive%281%29%281%29.html

5.4 Here, the variable 'Label' is set as target and the rest of the variables in the dataset are set as features (input variables). And then the overview of that dataset is then printed to check if it consists of any number of missing values or not, which in this case is 0.

```
[12] Target=dataset_m['Label'] #output
     Target

0       0
1       0
2       0
3       0
4       0
       ..
6553    1
6554    1
6555    1
6556    1
6557    1
Name: Label, Length: 6537, dtype: int64
```

Fig 7: Setting the target variable

```
Features=dataset_m.loc[:, dataset_m.columns != 'Label']
Features
```

	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	Bwd Pkt Len Max	B P L M
0	443.0	6.0	141385.0	9.0	7.0	553.0	3773.0	202.0	0.0	61.444444	87.534438	1460.0	0
1	49684.0	6.0	281.0	2.0	1.0	38.0	0.0	38.0	0.0	19.000000	26.870058	0.0	0
2	443.0	6.0	279824.0	11.0	15.0	1086.0	10527.0	385.0	0.0	98.727273	129.392497	1460.0	0
3	443.0	6.0	132.0	2.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0
4	443.0	6.0	274016.0	9.0	13.0	1285.0	6141.0	517.0	0.0	142.777778	183.887722	1460.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	
6553	8080.0	6.0	10239.0	3.0	4.0	326.0	129.0	326.0	0.0	108.666667	188.216188	112.0	0
6554	8080.0	6.0	474.0	2.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0
6555	8080.0	6.0	10860.0	3.0	4.0	326.0	129.0	326.0	0.0	108.666667	188.216188	112.0	0
6556	8080.0	6.0	487.0	2.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0
6557	8080.0	6.0	11398.0	3.0	4.0	326.0	129.0	326.0	0.0	108.666667	188.216188	112.0	0

6537 rows × 77 columns

Fig 8: Setting the features

```python
def dataoverview(dataset_m, message):
  print(f'{message}:\n')
  print("Rows:", dataset_m.shape[0])
  print("\nNumber of features:", dataset_m.shape[1])
  print("\nFeatures:")
  print(dataset_m.columns.tolist())
  print("\nMissing values:", dataset_m.isnull().sum().values.sum())
  print("\nUnique values:")
  print(dataset_m.nunique())
```

```python
dataoverview(dataset_m, 'Overview of the Training dataset')
```

```
Overview of the Training dataset:

Rows: 6537

Number of features: 78

Features:
['Dst Port', 'Protocol', 'Flow Duration', 'Tot Fwd Pkts', 'Tot Bwd Pkts', 'TotLen Fwd Pkts', 'TotLe

Missing values: 0

Unique values:
Dst Port          198
Protocol            3
Flow Duration    3249
Tot Fwd Pkts       95
Tot Bwd Pkts      100
                 ...
Idle Std          441
Idle Max          138
Idle Min          305
Label               2
TotLet Fwd Pkts   398
Length: 78, dtype: int64
```

Fig 9: Dropping the Na values and getting an overview of the data

5.5 The dataset is further divided into two subsets by splitting it in the ratio of 80 and 20 for training and testing respectively. The size of the training data is set as 80% of the actual data randomly and the rest of the 20% of the actual data as the testing data, which means, every time the code is executed, the training data will split from any part of the data randomly (can be 80% of the upper part, can be 80% of the middle part, can be 80% of the lower part etc.).

```python
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(Features,Target,train_size=0.80,random_state=2)
X_train.info(verbose=True)
X_test.info(verbose=True)
```

```
 20  Fwd IAT Mean       1308 non-null   float64
 21  Fwd IAT Std        1308 non-null   float64
 22  Fwd IAT Max        1308 non-null   float64
 23  Fwd IAT Min        1308 non-null   float64
 24  Bwd IAT Tot        1308 non-null   float64
 25  Bwd IAT Mean       1308 non-null   float64
 26  Bwd IAT Std        1308 non-null   float64
 27  Bwd IAT Max        1308 non-null   float64
 28  Bwd IAT Min        1308 non-null   float64
 29  Fwd PSH Flags      1308 non-null   float64
 30  Bwd PSH Flags      1308 non-null   float64
 31  Fwd URG Flags      1308 non-null   float64
 32  Bwd URG Flags      1308 non-null   float64
 33  Fwd Header Len     1308 non-null   float64
 34  Bwd Header Len     1308 non-null   float64
 35  Fwd Pkts/s         1308 non-null   float64
 36  Bwd Pkts/s         1308 non-null   float64
 37  Pkt Len Min        1308 non-null   float64
 38  Pkt Len Max        1308 non-null   float64
 39  Pkt Len Mean       1308 non-null   float64
 40  Pkt Len Std        1308 non-null   float64
 41  Pkt Len Var        1308 non-null   float64
 42  FIN Flag Cnt       1308 non-null   float64
 43  SYN Flag Cnt       1308 non-null   float64
 44  RST Flag Cnt       1308 non-null   float64
```

Fig 10: Splitting the data into training and testing

5.6 During this step, the data is split in 4 parts: X-train, X-test, Y-tarin and Y-test as training part of the features, testing part of the features, training part of the output and testing part of the output variable respectively.

6 Feature Extraction and Selection

Standardization means that each attribute's dispersion is modified to a mean of zero and a standard deviation (unit variance). For a model based on dispersal of variables, it is important to standardize the attributes. Therefore, the standardization is performed for feature scaling before feeding the data to the model like KNN. Feature Selection is a procedure in which we may choose features from the dataset, either programmatically or manually, that can contribute the most to the prediction variable or output.

6.1 In this step, the RandomForestClassifier library is utilized for selecting the topmost 15 important features from the entire data set to make the further implementation much easier and faster. In this, the Random Feature Elimination (RFE), a feature selection method is applied for eliminating all the features that are not essential.

```
from sklearn.feature_selection import RFE
import itertools
rfc=RandomForestClassifier()

#create the RFE model and select 15 attibutes
rfe=RFE(rfc,n_features_to_select=15)
rfe=rfe.fit(X_train,Y_train)

#summarize the selection of the attributes
feature_map=[(i,v)for i,v in itertools.zip_longest(rfe.get_support(),X_train.columns)]
selected_features=[v for i,v in feature_map if i==True]
selected_features
```

```
['Dst Port',
 'Flow Duration',
 'Flow IAT Mean',
 'Flow IAT Std',
 'Flow IAT Max',
 'Fwd IAT Tot',
 'Fwd IAT Mean',
 'Fwd IAT Std',
 'Fwd IAT Max',
 'Fwd IAT Min',
 'Fwd PSH Flags',
 'Fwd Pkts/s',
 'Pkt Len Std',
 'SYN Flag Cnt',
 'Init Fwd Win Byts']
```

Fig 11: Feature extraction and selection

6.2 After the feature selection is done programmatically a new data frame is created that consist of only those selected features that are received from the RFE method.

```
dataset = dataset_m[['Dst Port',
 'Flow Duration',
 'Flow IAT Mean',
 'Flow IAT Std',
 'Flow IAT Max',
 'Fwd IAT Tot',
 'Fwd IAT Mean',
 'Fwd IAT Std',
 'Fwd IAT Max',
 'Fwd IAT Min',
 'Fwd PSH Flags',
 'Fwd Pkts/s',
 'Pkt Len Std',
 'SYN Flag Cnt',
 'Init Fwd Win Byts','Label']]
dataset
```

	Dst Port	Flow Duration	Flow IAT Mean	Flow IAT Std	Flow IAT Max	Fwd IAT Tot	Fwd IAT Mea
0	443.0	141385.0	9425.666667	19069.116850	73403.0	141385.0	17673.12
1	49684.0	281.0	140.500000	174.655375	264.0	281.0	281.00
2	443.0	279824.0	11192.960000	24379.448340	112589.0	279728.0	27972.80
3	443.0	132.0	132.000000	0.000000	132.0	132.0	132.00
4	443.0	274016.0	13048.380950	26311.627030	114077.0	273946.0	34243.25
...	...	...	...	...	...	...	.
3141	8080.0	458.0	458.000000	0.000000	458.0	458.0	458.00
3142	8080.0	21450.0	3575.000000	8382.901646	20684.0	446.0	223.00
3143	8080.0	482.0	482.000000	0.000000	482.0	482.0	482.00
3144	8080.0	9612.0	1602.000000	3555.189559	8852.0	454.0	227.00
3145	8080.0	515.0	515.000000	0.000000	515.0	515.0	515.00

3125 rows × 16 columns

Fig 12: Creating new and final data frame

6.3 Further, the steps 5.4, 5.5 and 5.6 are repeated to get a finalized dataset including separated target, features and a training and a testing part of new generated dataset for further model implementation with only 16 columns.

7 Machine Learning Models

Model fitting is an estimation about how a machine learning model is generalized to comparable data to the one it is trained on. The well-fitted model typically yields accurate findings. Model fitting is a key component of machine learning. If the model doesn't match our dataset appropriately, the results can't be true and can't depend on the results to be predictable. Model Evaluation is an essential aspect of the approach of machine learning model building. It helps to identify the best model for the selected dataset and how well the selected model works in the near future.

7.1 In this step, various machine learning models are executed. Initially, the K-Nearest neighbour model is implemented with the value of k as 3 and the number of neighbours as 5 by default as these values are giving the best accuracy and 0-2 false negatives approximately (approximation is said after every result as the training of the testing dataset is given a random state and can vary after every execution). This model gives 99% of accuracy approximately with same percentage of precision, recall, f1-score results.(Chudasma, no date)

```
# Load libraries
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

# Train KNeighborsClassifier Model
KNN_Classifier=KNeighborsClassifier(n_jobs=3)
KNN_Classifier.fit(X_train,Y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=3, n_neighbors=5, p=2,
                     weights='uniform')
```

Fig 13: KNN model

```
Model Accuracy for KNN :
 1.0
Confusion matrix :
 [[375   0]
 [  0 250]]
Outcome values :
 375 0 0 250
Classification report :
               precision    recall  f1-score   support

           1       1.00      1.00      1.00       375
           0       1.00      1.00      1.00       250

    accuracy                           1.00       625
   macro avg       1.00      1.00      1.00       625
weighted avg       1.00      1.00      1.00       625
```

Fig 14: KNN model accuracy and evaluation matrix

7.2 In the next step, the Decision Tree classifier model is executed with the default hyper parameters such as criterion as ‘gini’ and random_state as None, are passed to this model to get the accuracy. It demonstrates that the dataset we have altered is inaccurately labelled for splitting from the dataset. It is utilized with Classification and regression tree, and the value is more precise and less accurate than its best quality, entropy index; lower values suggest fewer impurities. This model gives a 99-100% of accuracy approximately with 0-1 number of false negatives.[3]

```
#Train Decision tree model
DTC_Classifier=tree.DecisionTreeClassifier()
DTC_Classifier.fit(X_train,Y_train)

DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Fig 15: DTC model

```
Model Accuracy for DTC :
 1.0
Confusion matrix :
 [[375   0]
 [  0 250]]
Outcome values :
 375 0 0 250
Classification report :
               precision    recall  f1-score   support

           1       1.00      1.00      1.00       375
           0       1.00      1.00      1.00       250

    accuracy                           1.00       625
   macro avg       1.00      1.00      1.00       625
weighted avg       1.00      1.00      1.00       625
```

Fig 16: DTC model accuracy and evaluation matrix

7.3 The next model used is Artificial Neural Network (ANN). The information processing technology is Artificial Neural Network. This model works like a human brain. It is generally organised in 3 layers, input layer, hidden layer, output layer. The input layer receives the input values for every observation which do not change the data. The hidden layer provides a transformation to an input value in the network and then connects with the output nodes also to other hidden layers, generally known as ‘weighted connections’. The output layer gets the link from the other two layers (input and hidden) and then it combines and converts the data to generate the output values. Here, random weights are assigned to the linkages initially. Then all the three layers are connected and assigned the required parameters. Data preparation is similar to the rest of the identification technology in the performance of ANN. The keras library is utilized to run this model. The epochs (the number of times an algorithm works through the complete training data) is set as 50 and then the model is executed. This model gives 99.49% of accuracy with no false

[3] https://datascience.foundation/sciencewhitepaper/understanding-decision-trees-with-python

negatives approximately. Here, the precision and the f1-score also give the result of 97-99%. [4]

```
import keras
from keras.models import Sequential
from keras.layers import Dense

#Initializing ANN
ANN_classifier= Sequential()

#Adding the input layer and the first hidden layer
ANN_classifier.add(Dense(units=8, kernel_initializer='uniform',activation='relu',input_dim=15))

#Adding second hidden layer
ANN_classifier.add(Dense(units=8,kernel_initializer='uniform',activation='relu'))

#Adding the output layer
ANN_classifier.add(Dense(units=1,kernel_initializer='uniform',activation='sigmoid'))

#Compiling the ANN
ANN_classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

#Fitting the ANN to the training set
ANN_classifier.fit(X_train,Y_train,batch_size=15,epochs=50)

y_pred = ANN_classifier.predict(X_test)
y_pred
```

```
Epoch 1/50
167/167 [==============================] - 14s 1ms/step - loss: 0.3451 - accuracy: 0.8795
Epoch 2/50
167/167 [==============================] - 0s 1ms/step - loss: 0.1151 - accuracy: 0.9954
Epoch 3/50
167/167 [==============================] - 0s 1ms/step - loss: 0.2449 - accuracy: 0.9909
Epoch 4/50
167/167 [==============================] - 0s 1ms/step - loss: 0.0766 - accuracy: 0.9978
Epoch 5/50
167/167 [==============================] - 0s 1ms/step - loss: 0.0568 - accuracy: 0.9980
Epoch 6/50
167/167 [==============================] - 0s 1ms/step - loss: 0.0425 - accuracy: 0.9973
```

Fig 17: ANN model

```
Model Accuracy for ANN :
 1.0
Confusion matrix :
 [[250   0]
 [  0 375]]
Outcome values :
 375 0 0 250
Classification report :
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       375
           0       1.00      1.00      1.00       250

    accuracy                           1.00       625
   macro avg       1.00      1.00      1.00       625
weighted avg       1.00      1.00      1.00       625
```

Fig 18: ANN model accuracy and evaluation matrix

[4] https://stackoverflow.com/questions/68185988/valueerror-input-0-of-layer-sequential-is-incompatible-with-the-layer-expected

8 Deep Learning Models

8.1 For deep learning approach, two models are implemented in this project Multi-Layer Perceptron (MLP), which is a type of Deep Neural Networks (DNN) and Convolutional Neural Network (CNN) to analyse the prediction of the Network Intrusion Detection dataset. Iinvestigation of the variations in accuracy while changing the number of parameters is done is this step. In these models the data pre-processing method is like that of the other models. The tenserflow library is utilized along with the other libraries and in this model and keras function is imported from the tensorflow library. This model gives the accuracy of 98.17% to 100% approximately and the validation of the accuracy is 99% true approximately. The accuracy and the loss of this model is visualised and as shown below.[5]

```
model = Sequential()
model.add(Dense(12, input_dim=15, activation= 'relu'))
model.add(Dense(8, activation= 'relu' ))
model.add(Dense(1, activation= 'sigmoid' ))
# Compile model
model.compile(loss='binary_crossentropy', optimizer= 'adam', metrics=['accuracy'])
# Fit the model
history=model.fit(train_features,train_label,epochs=50, batch_size=15)

scores=model.evaluate(train_features,train_label)
# evaluate the model
#scores = model.evaluate(test_features,test_label,verbose=3)
```

```
Epoch 1/50
140/140 [==============================] - 1s 2ms/step - loss: 1045.1581 - accuracy: 0.5203
Epoch 2/50
140/140 [==============================] - 0s 1ms/step - loss: 24.1061 - accuracy: 0.9808
Epoch 3/50
140/140 [==============================] - 0s 2ms/step - loss: 14.0120 - accuracy: 0.9826
Epoch 4/50
140/140 [==============================] - 0s 2ms/step - loss: 25.4054 - accuracy: 0.9456
Epoch 5/50
140/140 [==============================] - 0s 1ms/step - loss: 13.8023 - accuracy: 0.9866
Epoch 6/50
140/140 [==============================] - 0s 2ms/step - loss: 1024.8279 - accuracy: 0.9846
Epoch 7/50
140/140 [==============================] - 0s 1ms/step - loss: 20.8005 - accuracy: 0.9893
Epoch 8/50
140/140 [==============================] - 0s 1ms/step - loss: 8.0596 - accuracy: 0.9922
Epoch 9/50
140/140 [==============================] - 0s 1ms/step - loss: 6.3303 - accuracy: 0.9950
Epoch 10/50
140/140 [==============================] - 0s 1ms/step - loss: 10.5864 - accuracy: 0.9882
Epoch 11/50
140/140 [==============================] - 0s 1ms/step - loss: 7.8184 - accuracy: 0.9902
Epoch 12/50
140/140 [==============================] - 0s 1ms/step - loss: 8.4445 - accuracy: 0.9908
Epoch 13/50
140/140 [==============================] - 0s 1ms/step - loss: 13.5563 - accuracy: 0.9914
Epoch 14/50
140/140 [==============================] - 0s 1ms/step - loss: 7.2007 - accuracy: 0.9922
```

Fig 19: DNN model

5 https://elearning.dbs.ie/pluginfile.php/1301095/mod_resource/content/1/Deep%20Learning%20Tutorial.html

```
[135] #printing the traing accuracy
      print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

      accuracy: 99.86%

[136] # predications
      Pre_dict=model.predict_classes(test_features)
      #comparing the results

      acc_score=accuracy_score(test_label,Pre_dict)

      print("The testing accuracy of the Model {}".format(str(acc_score*100)))

      The testing accuracy of the Model 99.6124031007752
```

Fig 20: Training and Testing set accuracy

```
import keras
'''
#7 layer MLP
'''
# create model
model = Sequential()
model.add(Dense(12, input_dim=15, activation= 'relu'))
model.add(Dense(8, activation= 'relu' ))
model.add(Dense(8, activation= 'relu' ))
model.add(Dense(8, activation= 'relu' ))
model.add(Dense(8, activation= 'relu' ))
model.add(Dense(1, activation= 'sigmoid' ))
# Compile model
keras.optimizers.Adam(learning_rate=0.005, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(loss= 'binary_crossentropy' , optimizer= 'adam' , metrics=['accuracy'])
# Fit the model
history=model.fit(train_features,train_label, validation_data=(test_features,test_label) ,epochs=50, batch_size=15)
# evaluate the model
scores = model.evaluate(test_features,test_label,verbose=3)

140/140 [==============================] - 0s 2ms/step - loss: 0.3154 - accuracy: 0.9969 - val_loss: 0.9016 - val_accuracy: 0.9922
Epoch 22/50
140/140 [==============================] - 0s 2ms/step - loss: 0.6393 - accuracy: 0.9947 - val_loss: 0.5017 - val_accuracy: 0.9942
Epoch 23/50
140/140 [==============================] - 0s 2ms/step - loss: 0.3640 - accuracy: 0.9988 - val_loss: 0.6350 - val_accuracy: 0.9932
Epoch 24/50
140/140 [==============================] - 0s 2ms/step - loss: 0.1194 - accuracy: 0.9963 - val_loss: 0.7454 - val_accuracy: 0.9952
Epoch 25/50
140/140 [==============================] - 0s 2ms/step - loss: 0.5304 - accuracy: 0.9953 - val_loss: 0.7524 - val_accuracy: 0.9913
Epoch 26/50
140/140 [==============================] - 0s 2ms/step - loss: 0.2287 - accuracy: 0.9957 - val_loss: 0.7625 - val_accuracy: 0.9903
Epoch 27/50
140/140 [==============================] - 0s 3ms/step - loss: 1.2738 - accuracy: 0.9929 - val_loss: 0.7984 - val_accuracy: 0.9913
Epoch 28/50
140/140 [==============================] - 0s 3ms/step - loss: 0.0627 - accuracy: 0.9920 - val_loss: 0.6696 - val_accuracy: 0.9903
Epoch 29/50
140/140 [==============================] - 0s 3ms/step - loss: 0.0181 - accuracy: 0.9941 - val_loss: 0.6113 - val_accuracy: 0.9826
Epoch 30/50
140/140 [==============================] - 0s 2ms/step - loss: 0.1031 - accuracy: 0.9900 - val_loss: 0.6076 - val_accuracy: 0.9835
Epoch 31/50
140/140 [==============================] - 0s 2ms/step - loss: 0.0823 - accuracy: 0.9917 - val_loss: 0.6340 - val_accuracy: 0.9855
```

Fig 21: Validating the model

```
[139] print(np.mean(history.history['accuracy']))
      print(np.mean(history.history['val_accuracy']))

      0.9907214534282685
      0.9859302258491516
```

Fig 22: Accuracy and validation of the accuracy

```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

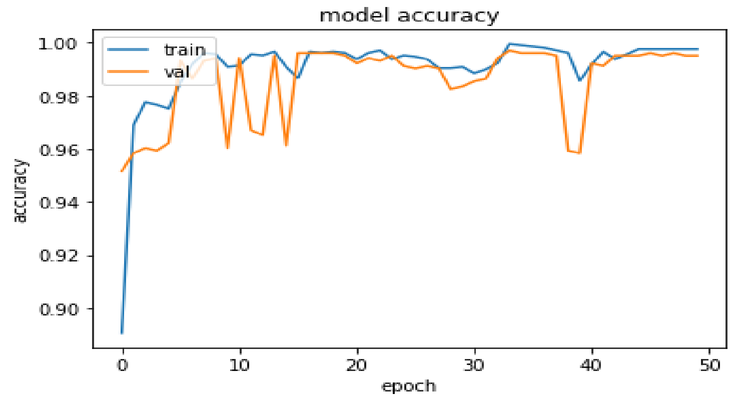

Fig 23: Visualisation of the actual and predicted accuracy

```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

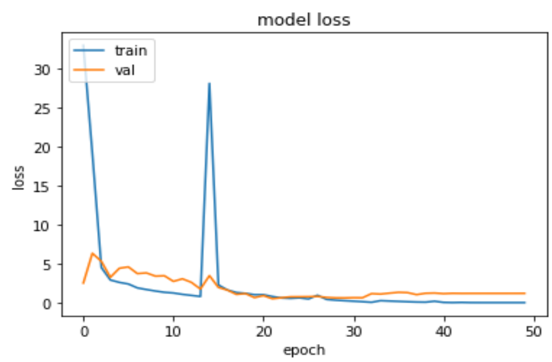

Fig 24: Visualisation of the actual and predicted loss

8.2 After the execution of DNN model the Convolutional Neural Network (CNN) model is implemented. The keras library is utilised in this model and the Conv1D, Flatten and MaxPooling1D functions ported from the library named keras.layers. Further, the features and the targets are assigned with x and y variable respectively. Further, the data frame features then converted into a numpy to apply the reshape attribute over it. Then, the dataset is split into training and testing part where the test size is set as 20% of the data and training size is set as 80% of the data. Then the model is executed, giving the accuracy 99.87% approximately with the loss of 22.13% approximately.[6]

[6] https://www.datatechnotes.com/2020/02/classification-example-with-keras-cnn.html

```
model = Sequential()
model.add(Conv1D(64, 2, activation="relu", input_shape=(15,1)))
model.add(Dense(16, activation="relu"))
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(2, activation = 'softmax'))
model.compile(loss = 'sparse_categorical_crossentropy',
     optimizer = "adam",
              metrics = ['accuracy'])
```

```
[152] model.summary()

Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 14, 64)            192
_________________________________________________________________
dense_15 (Dense)             (None, 14, 16)            1040
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 7, 16)             0
_________________________________________________________________
flatten (Flatten)            (None, 112)               0
_________________________________________________________________
dense_16 (Dense)             (None, 2)                 226
=================================================================
Total params: 1,458
Trainable params: 1,458
Non-trainable params: 0
_________________________________________________________________
```

Fig 25: Model summary

```
model.fit(X_train, Y_train, batch_size=16,epochs=50, verbose=0)

acc = model.evaluate(X_train, Y_train)
print("Loss:", acc[0], " Accuracy:", acc[1])

79/79 [==============================] - 0s 1ms/step - loss: 0.4873 - accuracy: 0.9900
Loss: 0.48732203245162964  Accuracy: 0.9900000095367432
```

Fig 26: Accuracy of actual and prdicted sets

9 Conclusion

As it can be observed here the some of the Machine learning Models are most of the time giving more accurate results than the Deep Learning models, while neural networks, the deep learning models are giving more accuracy than ANN in less computational time. So, it can be concluded that the Deep Learning models can give better accuracy than ANN, when it comes to neural networks, but the KNN and Decision Tree algorithms are the best fit models for this dataset (Results may vary by different dataset). Though in case of large and complex datasets. Deep Learning algorithms are much preferable for better accuracy and validation. The limitations of this research are as follows; Use of only one dataset is done and executed for all the models and Visualization of only one model is shown.

References

Chudasma, P. (no date) ‘Network Intrusion Detection System using Classification Techniques in Machine Learning’, p. 74.