Configuration Manual

MSc Research Project
Cyber Security

Loudu Mary Gade
Student ID: 23188189

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

Configuration Manual

Loudu Mary Gade
Student ID: 23188189

1 Introduction

This configuration manual offers cognitive solution that explains how to implement and deploy a Network Intrusion Detection System (IDS) based on machine learning (ML) algorithms. This project aims to identify the ability to detect anomalies in NSL-KDD and UNSW-NB15 datasets which consist of normal and abnormal network profiles. This manual describes how to prepare the datasets for analysis, how to create feature vectors, to choose relevant features, how to train a machine learning model, and how to assess its performance. Majority of the concepts that has been studied under this project are Random Forest, XGBoost, Support Vector Machine and other classifiers like K- Nearest Neighbors and Decision Trees.Another important part of this work is to improve the feature selection step for the machine learning models by using Pearson Correlation Coefficient and Recursive Feature Elimination (RFE) for the purpose of increasing the accuracy of final model and reducing computational expense. They help filter in models only the most pertinent characteristics which increases the models' learning efficiency and predictive accuracy of the model.The manual is suitable for beginner and advanced users; it guides the users through executing the solution in local environments as well as Google Colab. At the end of this guide, the reader will be well equipped with knowledge and understanding of how to filter network traffic data for machine learning, train and have insights in the performance of machine learning to detect network intrusions effectively.

2 System Requirements

Architecture hierarchy of the Network Intrusion Detection System (IDS) is discussed below: These consist of normal and attacks, including network traffics, which were tested using NSL-KADD and UNSW-NB15 datasets found in the Data Layer. This data can be store either into local storage or to cloud storage systems The Preprocessing layer relates to how the data is cleaned, transformed, encoded for categorical features, normalized and how different features are chosen using methods like Pearson correlation or Recursive feature elimination and so on. This prepares the data to be used in the different developing machine learning models.This makes the Model Training Layer contain the development of the machine learning models such as; Random Forest, XGBoost, SVM, KNN, Decision Trees among others. These models are trained through the prepared data to recognize network anomalies and attacks.The Evaluation and Testing Layer adopts Accurate, precisely, Recall value, F1 Score & Confusion Matrix to test the models. This makes the models do well in unseen data.To be precise, the last layer, the Visualization Layer, seeks to present performance metrics in heat map, confusion matrix, and pie chart to justify the effectiveness of the models.

2.1 System Architecture

```
path = 'UNSW_NB15_training-set.csv'
path_test = 'UNSW_NB15_testing-set.csv'

# Reading the datasets into pandas DataFrames
data = pd.read_csv(path)
data_test = pd.read_csv(path_test)
```

```
file_path_20_percent = 'KDDTrain+_20Percent.txt'
file_path_full_training_set = 'KDDTrain+.txt'
file_path_test = 'KDDTest+.txt'

# Load the datasets
df = pd.read_csv(file_path_full_training_set)
test_df = pd.read_csv(file_path_test)
```

Figure 2.1 Code snippet Dataset Loading

2.2 Hardware Requirements

To run the IDS efficiently with machine learning models, some basic and general requirements of hardware are needed. The minimum recommended Processor (CPU) should be an Intel i5 and of course an Intel i7 or higher is desirable especially if you will be dealing with large files. The RAM users should allocate is of the minimum 8 GB, but 16 GB or more is preferable to speed up data computation and processing of big data such as NSL-KDD and UNSW-NB15 datasets. Storage is another component where at least 50GB of free disk space is needed for storing the datasets and model results. If more data or another model is desired, the recommended storage space is 100 GB or higher. Although, GPU is not compulsory in general machine learning algorithms like Random Forest, XGBoost, SVM etc., but a GPU say NVIDIA GTX series is useful in Neural Network or larger data set.

Component	Minimum Requirement	Recommended Requirement
Processor (CPU)	Intel i5 or equivalent	Intel i7 or equivalent for better performance
RAM	8 GB	16 GB or more for faster data processing
Storage	50 GB of free disk space	100 GB or more if saving additional data or models
Graphics Processing Unit (GPU)	Not necessary for traditional ML models	Dedicated GPU (NVIDIA GTX series or equivalent) for neural networks

2.3 Software Requirements

The following software components are required as the necessary components of the IDS system that will be implemented and run. Local: OS could be Windows 10/11 or Linux>Ubuntu 18.04 or later except macOS, or Google Colab running on the cloud could be used. Google Colab will be most ideal for the users that may not have excess local hardware resources to code with.The Programming Language to be used must be Python 3. x with preference being given to Python 3.8 and above. In achieving the machine learning tasks, many libraries that will be needed include Pandas for data manipulation, numpy for computations, specifically for the numerical calculations scikit-learn for the Models in Machine learning and its preprocessing, feature selection and evaluation, XGBoost for the implementation of XGBoost model, Data visualization and Structural graphics are done by using Matplotlib and seaborn Scikit-plot is mainly used advanced visualization including Confusion Matrix. Other useful libraries are itertools and os which help in working with iterators and managing file paths.Respoinsible: Others include itertools and os which assist in operating iterators and file paths.For the Development Environment, tools like Jupyter Notebook or Google Colab is is suitable in introducing interactivity into coding offline development can be done using integrated development environment like Pycharm, visual studio code or spyder among others.

Software Component	Requirement
Operating System	Windows 10/11, Linux (Ubuntu 18.04 or later), macOS, or Google Colab
Programming Language	Python 3.x (Recommended: Python 3.8 or higher)
Libraries/Frameworks	pandas, numpy, scikit-learn, xgboost, matplotlib, seaborn, scikit-plot, itertools, os
Development Environment	Jupyter Notebook, Google Colab (Recommended for cloud-based execution), or IDE like PyCharm, Visual Studio Code, Spyder

```python
# used for working with arrays
import numpy as np
# for creating and removing a directory (folder),
import os
from sklearn.metrics import r2_score
# for data analysis
import pandas as pd
# for graphs
import matplotlib.pyplot as plt
import seaborn as sns
# provides various functions that work on iterators to produce complex iterators
import itertools
# to generate random numbers
import random
# for creating static, animated, and interactive visualizations
import matplotlib.gridspec as gridspec
#provides a selection of efficient tools for machine learning and statistical modeling
#
from sklearn.preprocessing import LabelEncoder
# To divide data in training and testing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
import scikitplot.metrics as splt
```

Figure 2.3 Code snippet Used Libraries

2.4 Network and Internet Requirements

A constant network connection is important for downloading real datasets from archives such as UNSW and NSL-KDD or from previous papers. To my knowledge, Google Colab needs this ongoing internet connection for loading data sets and for terminal running the cloud code. You may also require additional access to third-party datasources/APIs in particular if utilising external threat intelligence is relevant.

3 Installation and Setup

The process of implementing the Network Intrusion Detection System (IDS) USING Machine Learning ranges from different steps whether it is by Google Colab or through the local environment. In Google Colab, you start with creating a new notebook, piping necessary libraries, and, if needed, to mount the Google Drive to work with datasets. Else, in a local setting, Windows/Linux/macOS environment, one requires to install Python itself along with necessary libraries such as pandas, scikit-learn, XGBoost and pip to install necessary libraries and datasets like NSL-KDD, UNSW-NB15 to download and store them. In setup phase Once this is done the data is preprocessed by handling missing values, encoding categorical variables, and normalizing the features. Afterwards, a number of machine learning algorithms are built and tested using accuracy and recall as the performance metrics and the output is viewed in the form of confusion matrix with the corresponding values for random forest, XGBoost and SVM. For cloud setups Google Colab is convenient if you need an environment, for more computational power you can use AWS or GCP. This setup thus permits effective recognition and identification of intended network irregularities and attack using machine learning.

3.1 Development Environment Setup:

training process for both Google Colab and local environments follows a well-defined pipeline, beginning with dataset collection and ending with model selection and optimization.foundation of the training process is laid with the collection of two widely recognized datasets: The datasets used in the experiments within this paper are the NSL-KDD and UNSW-NB15 datasets. These datasets are employed for analyzing and monitoring the network traffic and to detect intrusion. The NSL-KDD dataset is derived from the raw KDD Cup 1999 dataset comprising of instances of normal traffic and attacks. UNSW-NB15 is a relatively newer dataset obtained from real life network traffic and includes different types attack categor

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125972 entries, 0 to 125971
Data columns (total 43 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   duration                     125972 non-null  int64
 1   protocol_type                125972 non-null  object
 2   service                      125972 non-null  object
 3   flag                         125972 non-null  object
 4   src_bytes                    125972 non-null  int64
 5   dst_bytes                    125972 non-null  int64
 6   land                         125972 non-null  int64
 7   wrong_fragment               125972 non-null  int64
 8   urgent                       125972 non-null  int64
 9   hot                          125972 non-null  int64
 10  num_failed_logins            125972 non-null  int64
 11  logged_in                    125972 non-null  int64
 12  num_compromised              125972 non-null  int64
 13  root_shell                   125972 non-null  int64
 14  su_attempted                 125972 non-null  int64
 15  num_root                     125972 non-null  int64
 16  num_file_creations           125972 non-null  int64
 17  num_shells                   125972 non-null  int64
 18  num_access_files             125972 non-null  int64
 19  num_outbound_cmds            125972 non-null  int64
...
 41  attack                       125972 non-null  object
 42  level                        125972 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

```
dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175341 entries, 0 to 175340
Data columns (total 45 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   id                 175341 non-null  int64
 1   dur                175341 non-null  float64
 2   proto              175341 non-null  object
 3   service            175341 non-null  object
 4   state              175341 non-null  object
 5   spkts              175341 non-null  int64
 6   dpkts              175341 non-null  int64
 7   sbytes             175341 non-null  int64
 8   dbytes             175341 non-null  int64
 9   rate               175341 non-null  float64
 10  sttl               175341 non-null  int64
 11  dttl               175341 non-null  int64
 12  sload              175341 non-null  float64
 13  dload              175341 non-null  float64
 14  sloss              175341 non-null  int64
 15  dloss              175341 non-null  int64
 16  sinpkt             175341 non-null  float64
 17  dinpkt             175341 non-null  float64
 18  sjit               175341 non-null  float64
 19  djit               175341 non-null  float64
...
 43  attack_cat         175341 non-null  object
 44  label              175341 non-null  int64
dtypes: float64(11), int64(30), object(4)
memory usage: 60.2+ MB
```

Figure 3.1 Features

Once the datasets are collected, they are read from the source in the working environment using pandas whether in Google Colab or any other local environment. Then the datasets are loaded and follow preprocessing to get the data in the right format to be used in training. These preprocessing transformations applicable to missing values are Replace Numeric feature mean or median Replace Categorical feature mode The value imputation for missing observations is done numerically or categorically to use the average for the feature (numerically) or the most frequent value or category (categorically). The subsequent variables like protocol_type, service, and flag are categorical variables hence we can easily encode these categorical variables using Label Encoding or One-Hot Encoding. In addition, the features duration, src_bytes, and dst_bytes are scaled by either MinMaxScaler or StandardScaler in order to confine its values to one or zero. Last but not the least, the data has been divided into training set and testing set Usually 70 & 80 percent data has been used for training the model and the remaining 20 & 30 percent has been used to test how the model performs in front of the unseen data.

3.2 Defining Attack Class

```
def class_attack(attack_cat):
    if attack_cat == 'Fuzzers':
        attack_type = 1
    elif attack_cat == 'Analysis':
        attack_type = 2
    elif attack_cat == 'Backdoors':
        attack_type = 3
    elif attack_cat == 'Dos':
        attack_type = 4
    elif attack_cat == 'Exploits':
        attack_type = 5
    elif attack_cat == 'Generic':
        attack_type = 6
    elif attack_cat == 'Reconnaissance':
        attack_type = 7
    elif attack_cat == 'Shellcode':
        attack_type = 8
    elif attack_cat == 'Worms':
        attack_type = 9
    else:
        attack_type = 0
    return attack_type

attack_class = df.attack_cat.apply(class_attack)
df['attack_class'] = attack_class

df.head(100)
```

Figure 3.2 Code snippet Attack Class

3.3 Feature Selection UNSW

FEATURE SELECTION

```python
normal = df[df.attack_class == 0]
```
Python

```python
tnormal = dft[dft.attack_class == 0]
```
Python

```python
Exploit = df[df.attack_class == 5]
```
Python

```python
tExploit = dft[dft.attack_class == 5]
```
Python

```python
total_data = pd.concat([normal, Exploit], ignore_index=True)
```
Python

```python
test_total_data = pd.concat([tnormal, tExploit], ignore_index=True)
```
Python

```python
total_data
```
Python

```python
test_total_data
```
Python

```python
non_numeric_cols = total_data.select_dtypes(include=['object']).columns
print("Non-numeric columns:", non_numeric_cols)
```
Python

```
Non-numeric columns: Index(['proto', 'service', 'state', 'attack_cat'], dtype='object')
```

```python
for col in non_numeric_cols:
    total_data[col] = total_data[col].astype('category').cat.codes
```
Python

Figure 3.3 Code snippet Features Selection of UNSW dataset

```python
non_numeric_cols = total_data.select_dtypes(include=['object']).columns
print("Non-numeric columns:", non_numeric_cols)
```

```
Non-numeric columns: Index(['proto', 'service', 'state', 'attack_cat'], dtype='object')
```

```python
for col in non_numeric_cols:
    total_data[col] = total_data[col].astype('category').cat.codes
```

```python
corr = total_data.corr()
corr_y = abs(corr['attack_class'])
highest_corr = corr_y[corr_y > 0.1]
highest_corr.sort_values(ascending=True)
```

```python
total_data_numeric = total_data.select_dtypes(exclude=['object'])
corr = total_data_numeric.corr()
```

```python
corr= total_data.corr()
corr_y = abs(corr['attack_class'])
highest_corr = corr_y[corr_y > 0.1]
highest_corr.sort_values(ascending=True)
```

```python
# Convert non-numeric (categorical) columns to numeric codes
test_total_data = test_total_data.apply(lambda col: col.astype('category').cat.codes if col.dtype == 'object' else col)

# Calculate the correlation matrix
corrt = test_total_data.corr()
corr_yt = abs(corrt['attack_class'])

# Filter correlations greater than 0.1 and sort in ascending order
highest_corrt = corr_yt[corr_yt > 0.1]
highest_corrt = highest_corrt.sort_values(ascending=True)

# Display the results
highest_corrt
```

Figure 3.3 Code snippet Features Selection of UNSW dataset

3.4 Heatmap for the Attributes

```python
plt.figure(figsize=(15,10))
g=sns.heatmap(test_total_data[highest_corrt.index].corr(),annot=True,cmap="RdYlGn")
```

Python

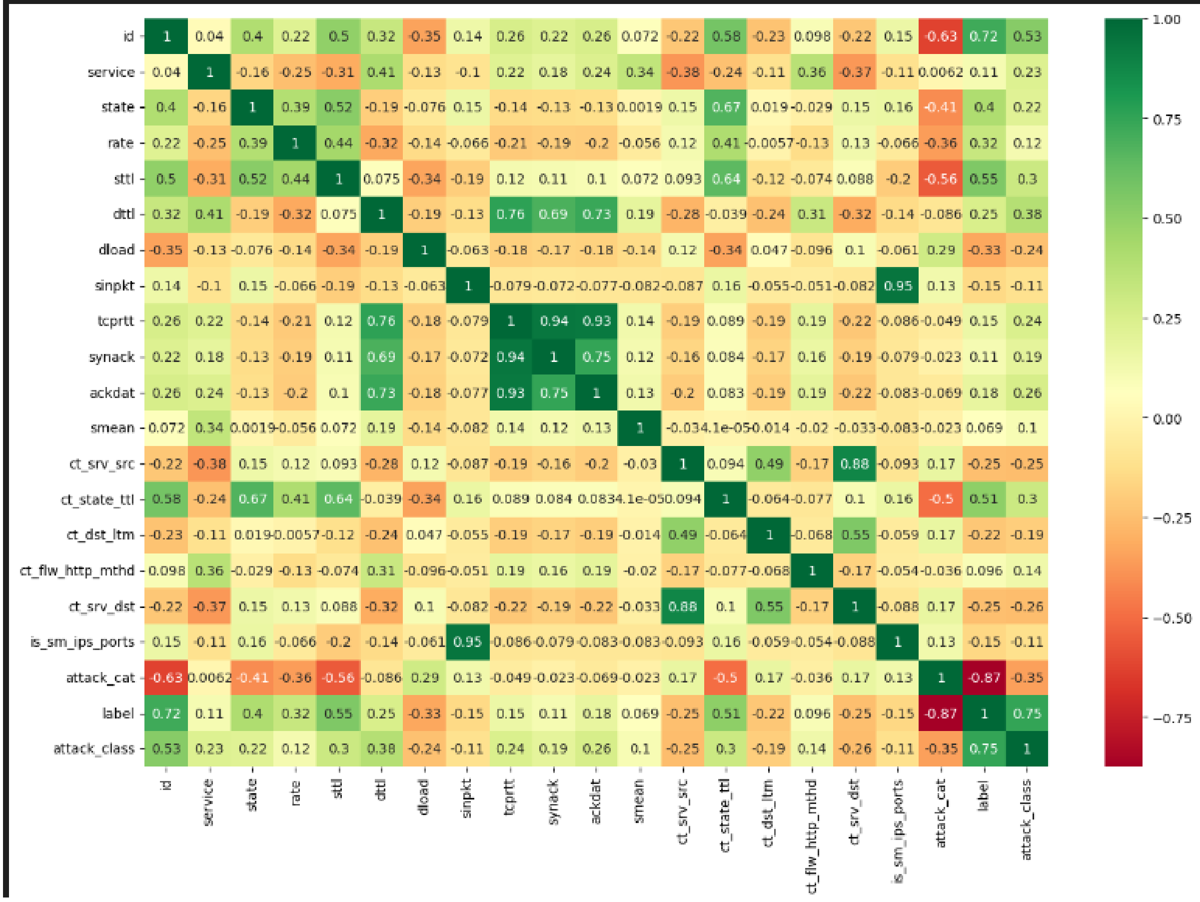


Figure 3.4 Code snippet Features Heatmap of UNSW dataset

3.5 Integer Attributes NSL-KDD

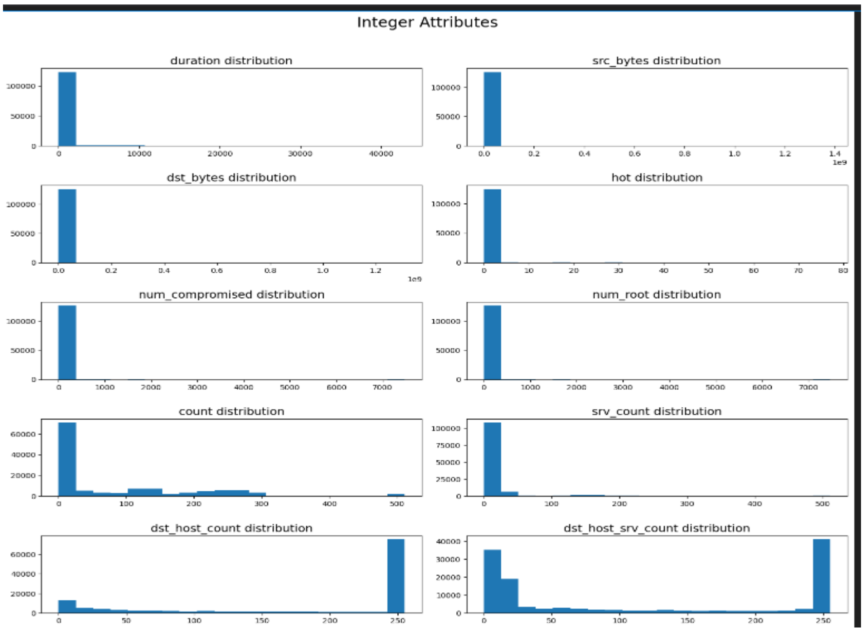


Figure 3.5 Code snippet Attributs

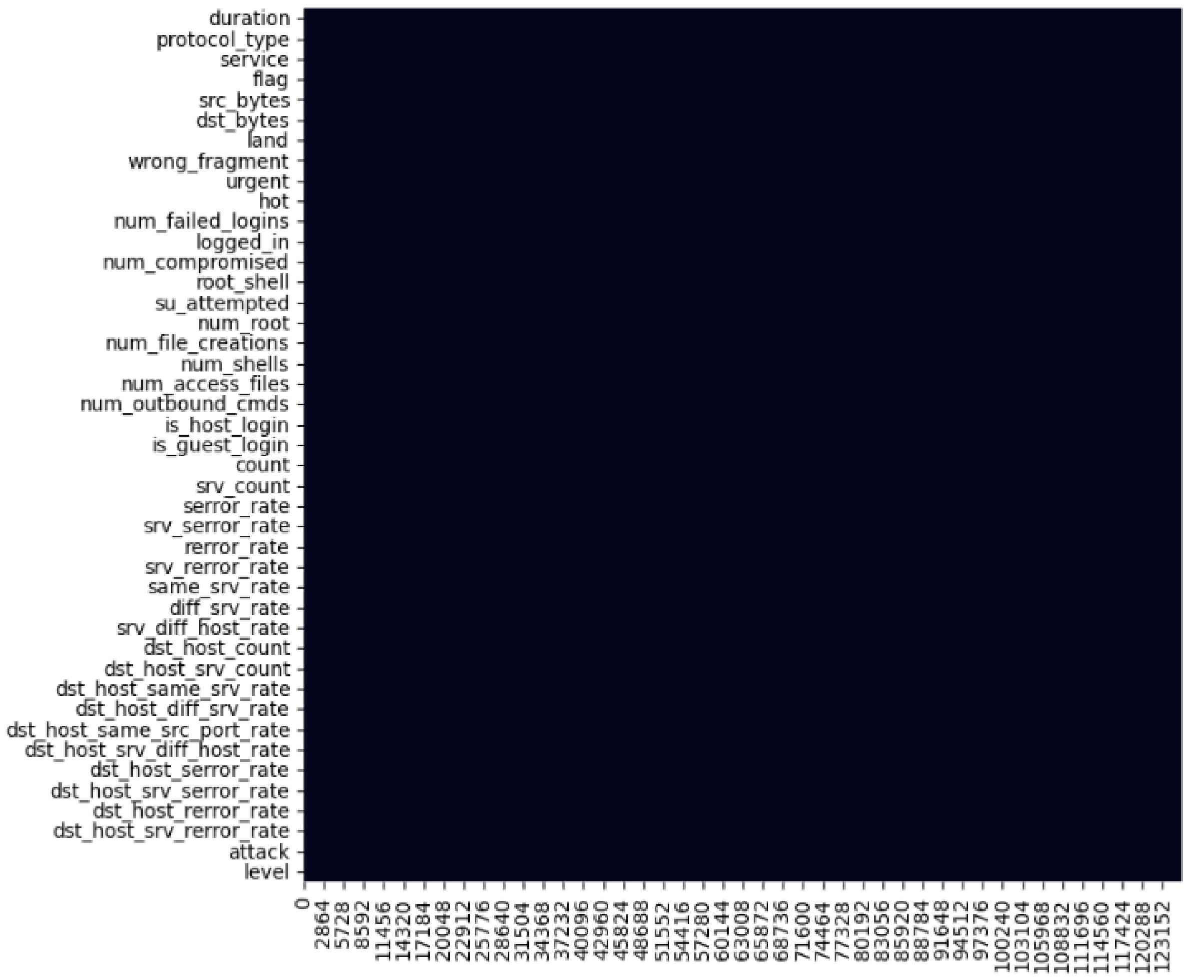


Figure 3.5 Code snippet Attributes distribution

3.6 Attacks ratio in Training And Testing Data

```
(df.attack_state == 1).sum()/len(df) #We have about 46% attacks in our Train dataset.

0.46542088871812784

(test_df.attack_state == 1).sum()/len(df) #We have about 10% attacks in our test dataset.

0.10186390626488426
```

Figure 3.6 Code snippet Attack Ratios for single dataset

```
sns.kdeplot(
    data=df, x="attack_state",
    fill=True, common_norm=False, palette="crest",
    alpha=.2, linewidth=10,
)
```

```
<ipython-input-21-14e10bd82136>:1: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.kdeplot(

<Axes: xlabel='attack_state', ylabel='Density'>
```

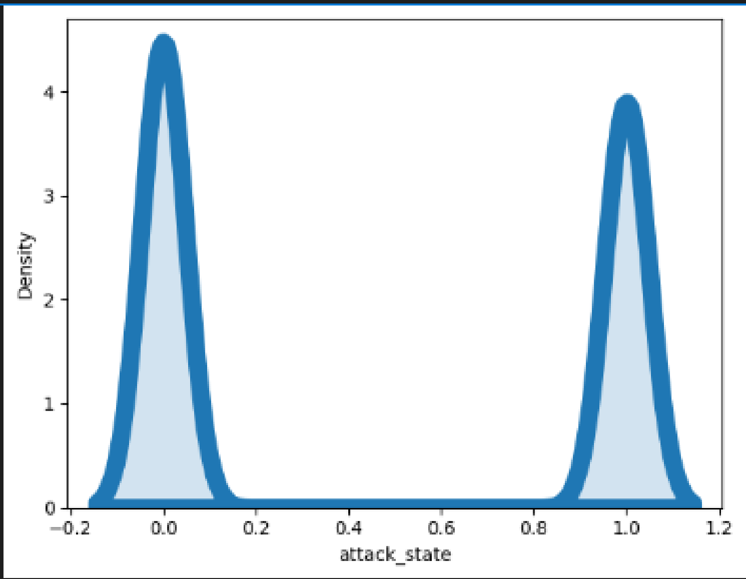


Figure 3.6 Code snippet Attack Ratios for single dataset

3.7 Attacks and Non-Attacks Ratio

```
my_labels = 'Number of Attacks','Number of Non-Attack Situations'
plt.pie(myData,labels=my_labels,autopct='%1.1f%%')
plt.title('Attack Rate')
plt.axis('equal')
plt.show()
```

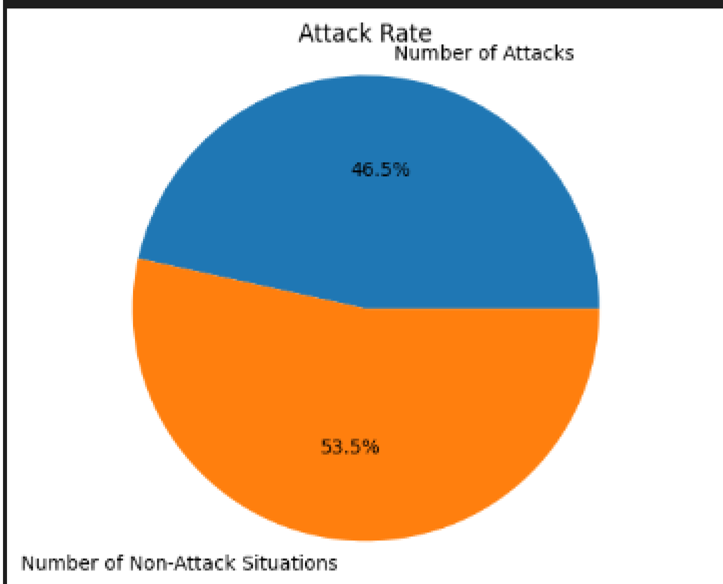


Figure 3.7 Diagram Attack Ratios for single dataset

3.8 Attacks

```python
DoS_attacks = ['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm']
Probe_attacks = ['ipsweep','mscan','nmap','portsweep','saint','satan']
U2R = ['buffer_overflow','loadmdoule','perl','ps','rootkit','sqlattack','xterm']
R2L = ['ftp_write','guess_passwd','http_tunnel','imap','multihop','named','phf','sendmail','snmpgetattack','snmpguess','spy','warezclient',

attack_labels = ['Normal','DoS','Probe','U2R','R2L']

def class_attack(attack):
    if attack in DoS_attacks:
        attack_type = 1
    elif attack in Probe_attacks:
        attack_type = 2
    elif attack in U2R:
        attack_type = 3
    elif attack in R2L:
        attack_type = 4
    else:
        attack_type = 0
    return attack_type


attack_class = df.attack.apply(class_attack)
df['attack_class'] = attack_class

test_attack_class = test_df.attack.apply(class_attack)
test_df['attack_class'] = test_attack_class

df.head()
```

Python

Figure 3.8 Code snippet Attack Classes

```python
Normal = (df.attack_class == 0).sum()/len(df)
print('Normal = ' , Normal)
DoSDDoS = (df.attack_class == 1).sum()/len(df)
print('DoS/DDoS = ' , DoSDDoS)
Probe = (df.attack_class == 2).sum()/len(df)
print("Probe = " , Probe )
U2R = (df.attack_class == 3).sum()/len(df)
print('U2R = ', U2R)
R2L = (df.attack_class == 4).sum()/len(df)
print('R2L = ' ,R2L)
```

```
Normal =  0.5346505572666942
DoS/DDoS =  0.3645810180040009
Probe =  0.09252849839646905
U2R =  0.00034134569586892325
R2L =  0.007898580636966945
```

```python
Normal = (test_df.attack_class == 0).sum()/len(test_df)
print('Normal = ' , Normal)
DoSDDoS = (test_df.attack_class == 1).sum()/len(test_df)
print('DoS/DDoS = ' , DoSDDoS)
Probe = (test_df.attack_class == 2).sum()/len(test_df)
print("Probe = " , Probe )
U2R = (test_df.attack_class == 3).sum()/len(test_df)
print('U2R = ', U2R)
R2L = (test_df.attack_class == 4).sum()/len(test_df)
print('R2L = ' ,R2L)
```

```
Normal =  0.43716453000931554
DoS/DDoS =  0.3308787650268376
Probe =  0.10739475668721998
U2R =  0.0028833784323293262
R2L =  0.12167856984429756
```

Figure 3.8 Code snippet Attack Classes

```python
#normal_class = df.loc[df.attack_class == 0].attack.value_counts()
DoSDDoS_class = df.loc[df.attack_class == 1].attack.value_counts()
probe_class = df.loc[df.attack_class == 2].attack.value_counts()


flag_axs = bake_pies([DoSDDoS_class , probe_class], ['DoS/DDoS','Probe'])
plt.show()
```

Python

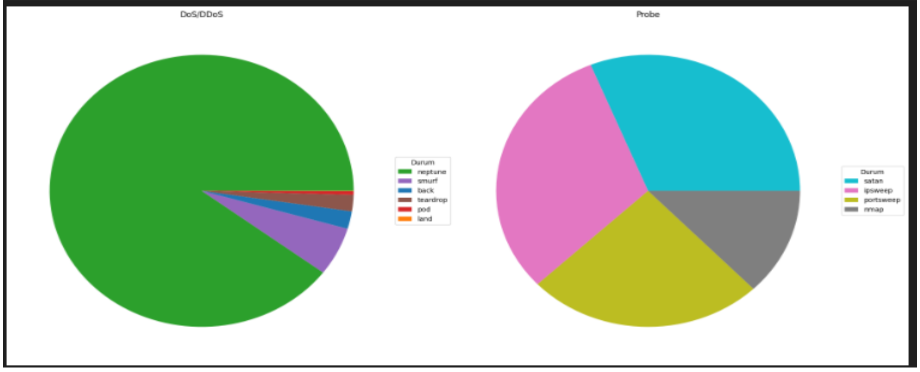


Figure 3.8 Diagram for Attack Classes

```python
U2R_class = df.loc[df.attack_class == 3].attack.value_counts()
R2L_class = df.loc[df.attack_class == 4].attack.value_counts()

flag_axs = bake_pies([U2R_class,R2L_class], ['U2R','R2L'])
plt.show()
```

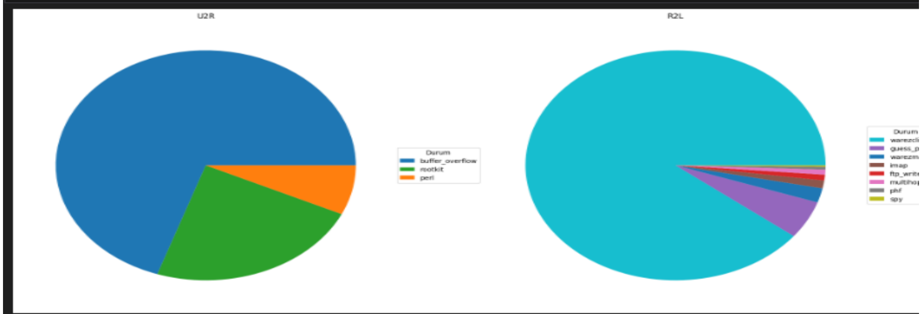


Figure 3.8 Diagram for Attack Classes

```python
Normal = (df.attack_class == 0).sum()
DoSDDoS = (df.attack_class == 1).sum()
Probe = (df.attack_class == 2).sum()
U2R = (df.attack_class == 3).sum()
R2L = (df.attack_class == 4).sum()
myData = [Normal , DoSDDoS,Probe,U2R,R2L]

my_labels = 'Normal','DoS/DDoS' ,'Probe' ,'U2R' , 'R2L'
plt.pie(myData,labels=my_labels ,autopct='%1.1f%%' , shadow = True)
plt.title('Attack Classes\n\n')
plt.axis('equal')
plt.show()
```

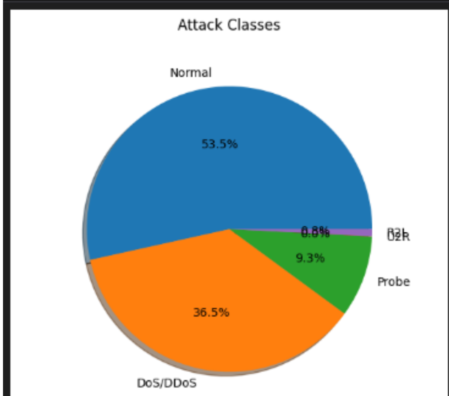


Figure 3.8 Diagram for Attack Classes

```python
icmp = attack_vs_protocol.icmp.sum()
tcp = attack_vs_protocol.tcp.sum()
udp = attack_vs_protocol.udp.sum()
myData = [icmp , tcp,udp]
my_labels = 'icmp','tcp' ,'udp'
plt.pie(myData,labels=my_labels ,autopct='%1.1f%%' , shadow = True)
plt.title('protocol type\n\n')
plt.axis('equal')
plt.show()
```

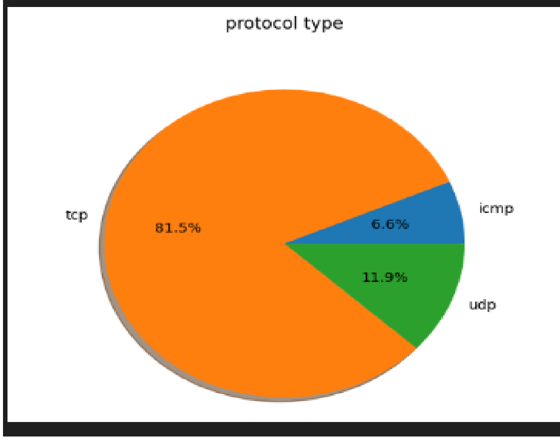


Figure 3.8 Diagram for Attack Classes

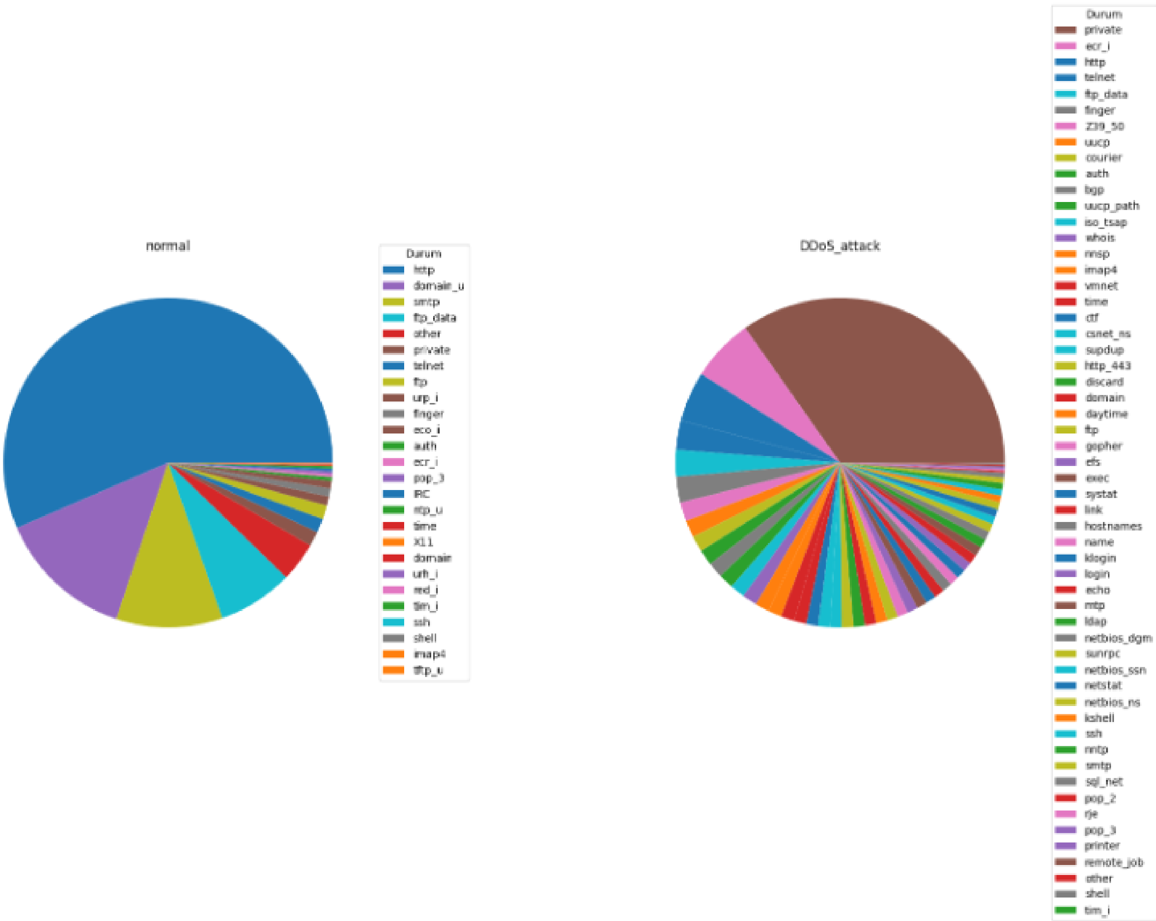


Figure 3.8 Diagram for Features representation for Attack Classes

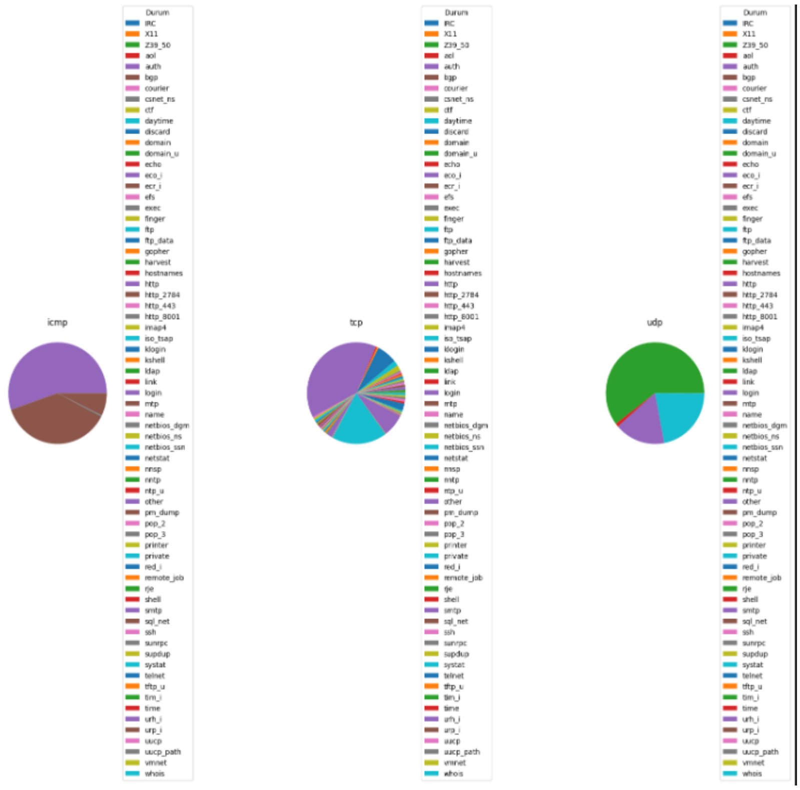


Figure 3.8 Diagram for Features representation for Attack Classes

3.9 Feature Selection

```python
normal = df[df.attack_class == 0]
```

```python
normal_test= test_df[test_df.attack_class == 0]
```

```python
DDoS = df[df.attack_class == 1]
```

```python
DDoS_test= test_df[test_df.attack_class == 1 ]
```

```python
total_data = pd.concat([normal, DDoS], ignore_index=True)
```

```python
total_data_test = pd.concat([normal_test, DDoS_test], ignore_index=True)
```

```python
total_data
```

```python
total_data_test
```

```python
corr= total_data.corr()
corr_y = abs(corr['attack_class'])
highest_corr = corr_y[corr_y > 0.1]
highest_corr.sort_values(ascending=True)
```

```python
corr= total_data_test.corr()
corr_y = abs(corr['attack_class'])
highest_corr_test = corr_y[corr_y >0.1]
highest_corr_test.sort_values(ascending=True)
```

Figure 3.9 Code Snippet for feature selection

```
corr= total_data.corr()
corr_y = abs(corr['attack_class'])
highest_corr = corr_y[corr_y > 0.1]
highest_corr.sort_values(ascending=True)
```

```
corr= total_data_test.corr()
corr_y = abs(corr['attack_class'])
highest_corr_test = corr_y[corr_y >0.1]
highest_corr_test.sort_values(ascending=True)
```

```
highest_corr_columns= highest_corr.index
```

```
highest_corr_test_columns= highest_corr_test.index
```

```
plt.figure(figsize=(15,10))
g=sns.heatmap(total_data[highest_corr.index].corr(),annot=True,cmap="RdYlGn")
```

Figure 3.9 Code Snippet for feature selection

3.10 Heat Map of Features

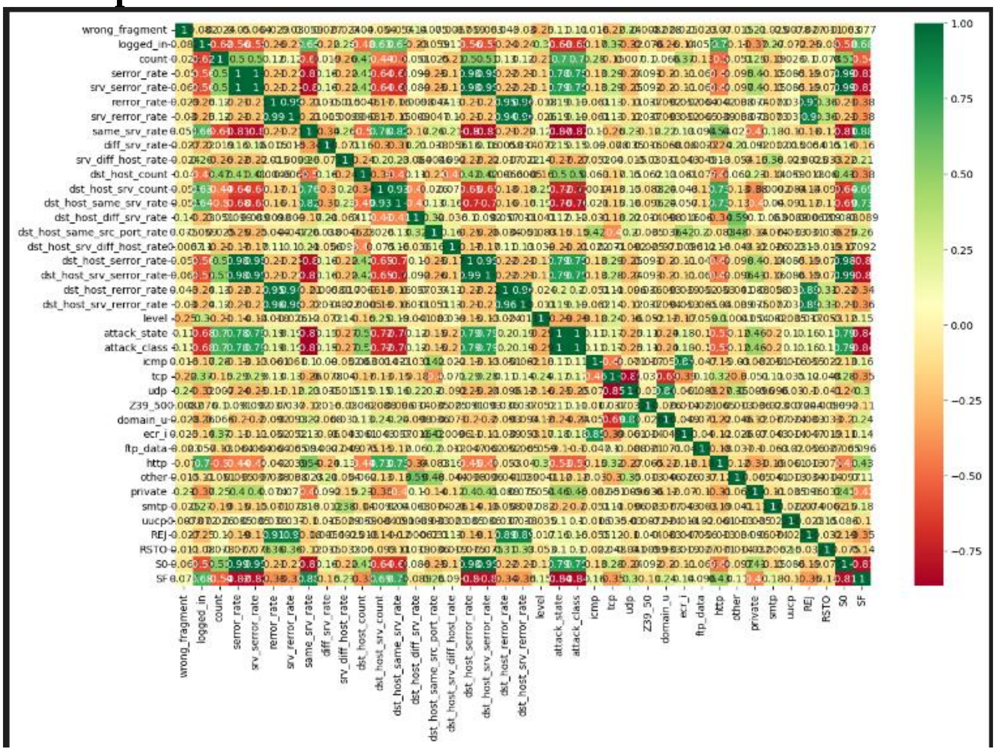


Figure 3.10 Diagram for Heatmap of KDD

- Since the data has been preprocessed, the next process is model selection. In this task, three machine learning models namely Random Forest, XGBoost, and Support Vector Machine (SVM) is selected. These models are chosen as a result of their ability to process a significant amount of information and high performance at detecting subtle features of network traffic. Random Forest is another type of classifier that uses the result of n decision trees in order to classify the data and it is also free of overfitting. XGBoost is a gradient boosting that works through building the model iteratively with a new model trained to minimize the error of the previous model, thereby making it perfect for training imbalanced datasets such as network traffic. The basic concept of SVM is particularly suitable for binary classification problems and it does not limit itself by the linearity or non-linearity of data.
- After the models have been selected, training follows.lection. For this task, three machine learning models—Random Forest, XGBoost, and Support Vector Machine (SVM)—are chosen. These models are selected due to their ability to handle large, complex datasets and their proficiency in capturing intricate patterns in network traffic. Random Forest is an ensemble method that combines multiple decision trees to make predictions and is known for avoiding overfitting. XGBoost is a gradient boosting framework that builds models sequentially, with each new model correcting the errors of the previous one, making it highly efficient for imbalanced datasets like network traffic. SVM is effective for binary classification problems and can handle both linear and non-linear data.
- As for the selection of the models, the training phase follows. During the training process of the model, this model identifies the preprocessed data and transfers the parameters through the features together with the corresponding attack class label. For instance, the Random Forest model creates n numbers of decision trees, and the XG Boost model develops the decision tree as well and builds the models to correct the previous model's errors. The other type of regularization though less common is done also in the selection of hyperparameters with a view of enhancing the existing models. Some goes to factors such as number of trees in Random Forest, learning rate in XGBoost, type of kernel in SVM. The parameters of the above models are mu, alpha, and lambda for the Original GMM; k and m) for the EM GMM; and c and gamma for the K-means GMM resp Libertarians defend utilitarianism on the basis of rights: Analyzing the parameters of the above models, mu, alpha and lambda are the parameters of the original GMM; k and m are that for the EM-GMM; c as well• In Google Colab, the training process has advantages in terms of using external resources of the cloud, including such important resources as GPUs, which make it possible to carry out training much faster, especially in critical situations that require the use of large amounts of data.ck class label. For instance, the Random Forest model generates n numbers of decision trees, and the XGBoost model develops the decision trees also and builds the models to rectify the mistakes of the previous models. Regularization is done also in the selection of hyperparameters with the intention of improving on the models. It extends to parameters like the number of decision trees in Random Forest, the rate of learning in XGBoost, kernel type in SVM. The parameters associated with the above models include mu, alpha, and lambda for the original GMM; k and m

for the EM GMM; and c and gamma for the K-means GMM models respectively Hyperparameters include items such as grid search or random search.

- In Google Colab, the training process benefits from the platform's cloud-based resources, including access to GPUs, which significantly accelerates the training process, especially when dealing with large datasets. After models are generated, the very next action is to assess the models, which we shall discuss below. The evaluation process is to check how well the trained models performed on a test set of data which the models had not seen at all. The multiple indices used to compare the results are accuracy, precisions, recollect rates, F1-scores, and confusion tables. Accuracy sums up the per cent of correct classification of all the records, while precision and recall see how a model does the right thing right in classifying positive records and avoiding both false Positive and false negative records. From the F1-score, it is clear that for getting the single value for the performance evaluation of the model, we use the harmonic mean of precision and recall. A confusion matrix is also utilised visually to examine if a model can differentiate between normal and distorted traffic.
- • The best model is that chosen from the evaluated models according to the measurements made during the evaluation process. Additionally, the choice is based on such a balance between accuracy, precision, recall, and F1-score as possible. For instance, if the number of false positives must be kept to a minimum, then a high precision model (e.g XGBoost) can be used. In case high recall is a goal, that is, when the main objective is to identify as many attacks as possible, then SVM might be chosen. The final assessment for the best models for the system guarantees the intricacies of the intrusion detection in consideration with the task.
- Apart from model selection, both hyperparameter optimization and cross-validation are essential steps in deriving the best out of models.the trade-offs between accuracy, precision, recall, and F1-score. For example, if minimizing false positives is crucial, a model with high precision (such as XGBoost) may be preferred. If detecting as many attacks as possible is the priority, a model with high recall (such as SVM) might be chosen. The final model selection ensures that the system meets the specific needs of the intrusion detection task.
- In addition to evaluating and selecting models, hyperparameter tuning and cross-validation play crucial roles in optimizing model performance. The second step is fine-tuning where different hyperparameters are tuned in a bid to get a better model; the methods applied here are the Grid Search and Random Search. Cross-validation is used to avoid overfitting, that is tester performance is checked and then compared with train data so that to ensure that the model performs perfectly on data that it has not seen before. Cross-validation divides the set of values into several folders, while several subsets are used for training sets and others for the validation sets. This makes it easier to give a better estimate of the performance of a given model and also minimizes chances of overfitting.

4 Final Testing and Validation

This paper examined the network intrusion detection system and checked the results with the models to see how well they fared on new data. During this phase, the test datasets that had not been used in any of the model training were invoked to check the capability of the same models in identifying anomalies. The evaluation measures applied included, Accuracy, Precision, Recall, F1 score, and the confusion matrix. These metrics gave clear picture about the nature of model both strengths and weaknesses which exist in each of them.

For instance, there was high accuracy and precision regarding the XGBoost model, a clear show of how its feature was capable of categorizing instances in the most efficient manner without watering down its true positive rate. Random forest focusing on big datasets yielded good performance and low Variance, whereas SVM model showed good recall and identify rare type of attack. A confusion matrix assisted in the depiction of the spread of the predicted results in an attempt to identify the regions that the models were accurate in their predictions as well as the regions that they were inclined to make errors. Comparing these results for all models was done and it revealed that image different algorithms possess certain strengths and relies on the selection of the model that typically suits the IDS.

The validation and testing phase asserted that the models selected were able to discern both known and previously undetected threats in the network traffic. The preprocessing techniques used in this study; feature selection and normalization, enhanced models' performance. All in all, the testing phase proved that the proposed concept of using the machine learning-based approach to address the problem of network intrusions is feasible.

4.1 Classification

```
from sklearn import tree
import matplotlib.pyplot as plt
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
dt_pred = clf.predict(X_test)
# print("Accuracy:",metrics.accuracy_score(y_test, dt_pred))
acc1 = accuracy_score(y_test ,dt_pred)
pre = precision_score(y_test ,dt_pred,average='weighted',labels=np.unique(dt_pred))
rec = recall_score(y_test ,dt_pred,average='weighted')
f1 = f1_score(y_test ,dt_pred,average='weighted')
print('ACCURACY : ',acc1)
print('PRECISON : ',pre)
print('RECALL : ',rec)
print('F1_SCORE : ',f1)
```

```
ACCURACY :  0.6830119595530993
PRECISON :  0.7006788981378727
RECALL :  0.6830119595530993
F1_SCORE :  0.6832067145867656
```

```
splt.plot_confusion_matrix(y_test, dt_pred)
```

```
<Axes: title={'center': 'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```

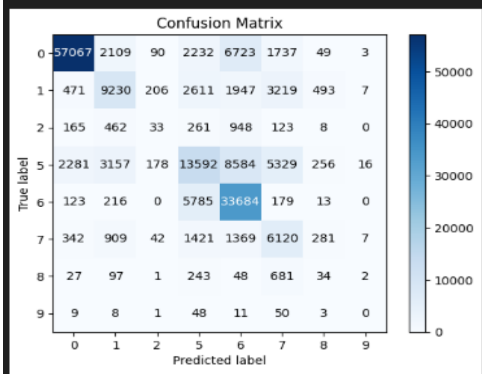


Figure 4.1 Code Snippet for Decision tree Classifier

	precision	recall	f1-score	support
0	0.94	0.82	0.87	70010
1	0.57	0.51	0.54	18184
2	0.06	0.02	0.03	2000
5	0.52	0.41	0.46	33393
6	0.63	0.84	0.72	40000
7	0.35	0.58	0.44	10491
8	0.03	0.03	0.03	1133
9	0.00	0.00	0.00	130
accuracy			0.68	175341
macro avg	0.39	0.40	0.39	175341
weighted avg	0.70	0.68	0.68	175341

Figure 4.1 Results Console

```
print(classification_report(y_test, dt_pred))
```

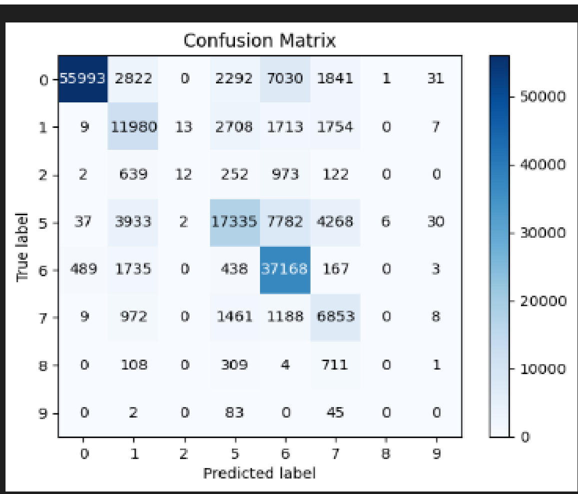
	precision	recall	f1-score	support
0	0.94	0.82	0.87	70010
1	0.57	0.51	0.54	18184
2	0.06	0.02	0.03	2000
5	0.52	0.41	0.46	33393
6	0.63	0.84	0.72	40000
7	0.35	0.58	0.44	10491
8	0.03	0.03	0.03	1133
9	0.00	0.00	0.00	130
accuracy			0.68	175341
macro avg	0.39	0.40	0.39	175341
weighted avg	0.70	0.68	0.68	175341

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X_train = mms.fit_transform(X_train)
X_test= mms.transform(X_test)
```

```
from sklearn.ensemble import GradientBoostingClassifier
gc = GradientBoostingClassifier()
gc.fit(X_train,y_train)
gb_pred = gc.predict(X_test)
#finding different scores
acc6 = accuracy_score(y_test ,gb_pred)
pre = precision_score(y_test ,gb_pred,average='weighted',labels=np.unique(gb_pred))
rec = recall_score(y_test ,gb_pred,average='weighted')
f1 = f1_score(y_test ,gb_pred,average='weighted')
print('ACCURACY : ',acc6)
print('PRECISON : ',pre)
print('CLASSIFIER RECALL : ',rec)
print('CLASSIFIER F1_SCORE : ',f1)
```

```
ACCURACY :  0.7376540569518824
PRECISON :  0.7669938462619407
CLASSIFIER RECALL :  0.7376540569518824
CLASSIFIER F1_SCORE :  0.7364663347764158
```

Figure 4.1 Code Snippet Results console




```
print(classification_report(y_test, gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.80      0.88     70010
           1       0.54      0.66      0.59     18184
           2       0.44      0.01      0.01      2000
           5       0.70      0.52      0.59     33393
           6       0.67      0.93      0.78     40000
           7       0.43      0.65      0.52     10491
           8       0.00      0.00      0.00      1133
           9       0.00      0.00      0.00       130

    accuracy                           0.74    175341
   macro avg       0.47      0.45      0.42    175341
weighted avg       0.77      0.74      0.74    175341
```

Figure 4.1 Code Snippet for DecisionTree Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 6)
knn = knn.fit(X_train , y_train)
knn_pred = knn.predict(X_test)
# print("Accuracy:",metrics.accuracy_score(y_test, knn_pred))
acc1 = accuracy_score(y_test ,knn_pred)
pre = precision_score(y_test ,knn_pred,average='weighted',labels=np.unique(knn_pred))
rec = recall_score(y_test ,knn_pred,average='weighted')
f1 = f1_score(y_test ,knn_pred,average='weighted')
print('ACCURACY : ',acc1)
print('PRECISON : ',pre)
print('RECALL : ',rec)
print('F1_SCORE : ',f1)
```

```
ACCURACY :  0.5588538904192403
PRECISON :  0.593781317695318
RECALL :  0.5588538904192403
F1_SCORE :  0.5690349306006097
```

```
splt.plot_confusion_matrix(y_test, knn_pred)
```

```
<Axes: title={'center': 'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```

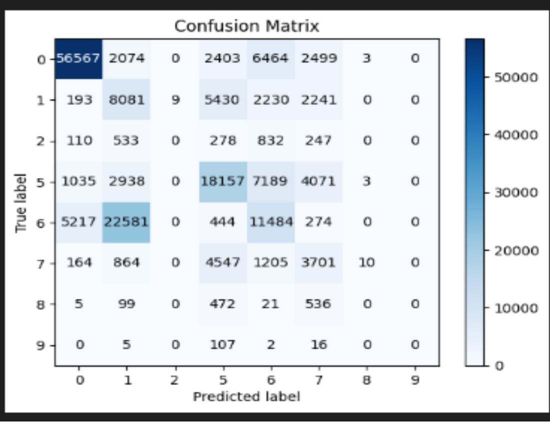


Figure 4.1 Code Snippet for KNN Classifier

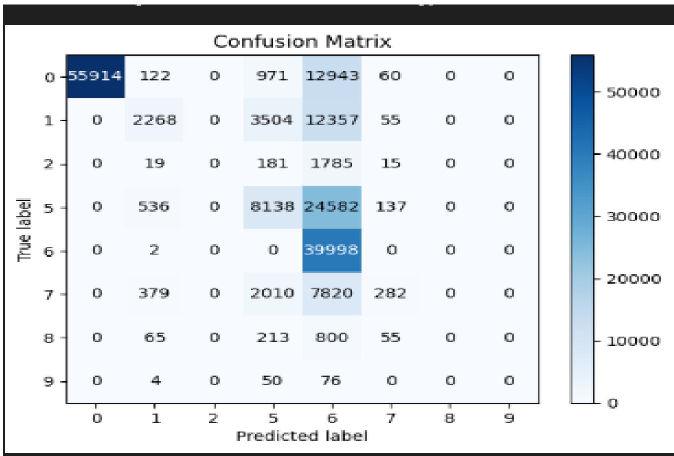
```
print(classification_report(y_test, knn_pred))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricW
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricW
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

           0       0.89      0.81      0.85     70010
           1       0.22      0.44      0.29     18184
           2       0.00      0.00      0.00      2000
           5       0.57      0.54      0.56     33393
           6       0.39      0.29      0.33     40000
           7       0.27      0.35      0.31     10491
           8       0.00      0.00      0.00      1133
           9       0.00      0.00      0.00       130

    accuracy                           0.56    175341
   macro avg       0.29      0.30      0.29    175341
weighted avg       0.59      0.56      0.57    175341

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricW
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression( solver='liblinear')
lr = lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
#finding different scores
acc1 = accuracy_score(y_test ,lr_pred)
pre = precision_score(y_test ,lr_pred,average='weighted',labels=np.unique(lr_pred))
rec = recall_score(y_test ,lr_pred,average='weighted')
f1 = f1_score(y_test ,lr_pred,average='weighted')
print('ACCURACY : ',acc1)
print('PRECISON : ',pre)
print('REGRESSION RECALL : ',rec)
print('REGRESSION F1_SCORE : ',f1)
```

```
ACCURACY :  0.6079582071506379
PRECISON :  0.7033652300575868
REGRESSION RECALL :  0.6079582071506379
REGRESSION F1_SCORE :  0.573405188096545
```

Figure 4.1 Code Snippet for LR Classifier

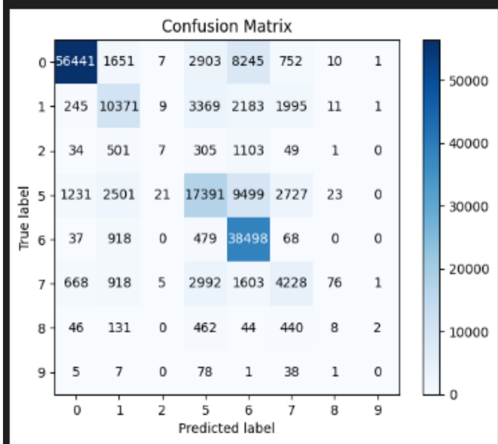



```
print(classification_report(y_test, lr_pred)) # A value of 1 here g
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatior
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatior
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
              precision    recall  f1-score   support

           0       1.00      0.80      0.89     70010
           1       0.67      0.12      0.21     18184
           2       0.00      0.00      0.00      2000
           5       0.54      0.24      0.34     33393
           6       0.40      1.00      0.57     40000
           7       0.47      0.03      0.05     10491
           8       0.00      0.00      0.00      1133
           9       0.00      0.00      0.00       130

    accuracy                           0.61    175341
   macro avg       0.38      0.27      0.26    175341
weighted avg       0.69      0.61      0.57    175341
```

Figure 4.1 Diagram for Confusion Metrix

```
<Axes: title={'center': 'Confusion Matrix'}, xlabel='Predicted label', ylabel='True
```




```
print(classification_report(y_test, rm_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     70010
           1       0.61      0.57      0.59     18184
           2       0.14      0.00      0.01      2000
           5       0.62      0.52      0.57     33393
           6       0.63      0.96      0.76     40000
           7       0.41      0.40      0.41     10491
           8       0.06      0.01      0.01      1133
           9       0.00      0.00      0.00       130

    accuracy                           0.72    175341
   macro avg       0.43      0.41      0.40    175341
weighted avg       0.74      0.72      0.72    175341
```

Figure 4.1 Diagram for Confusion Metrix

4.2 Evaluation Of NSL-KDD

DECISION TREE

```
from sklearn import tree
import matplotlib.pyplot as plt
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
dt_pred = clf.predict(X_test)
# print("Accuracy:",metrics.accuracy_score(y_test, dt_pred))
acc1 = accuracy_score(y_test ,dt_pred)
pre = precision_score(y_test ,dt_pred,average='weighted',labels=np.unique(dt_pred))
rec = recall_score(y_test ,dt_pred,average='weighted')
f1 = f1_score(y_test ,dt_pred,average='weighted')
r2 = r2_score(y_test, dt_pred)
print('R2_score : ' + str(r2))
print('ACCURACY : ',acc1)
print('PRECISON : ',pre)
print('RECALL : ',rec)
print('F1_SCORE : ',f1)
```

```
R2_score : 0.8140112498225854
ACCURACY :  0.8739741826731136
PRECISON :  0.9148103335815909
RECALL :  0.8739741826731136
F1_SCORE :  0.883696786169602
```

```
splt.plot_confusion_matrix(y_test, dt_pred)
```

```
<Axes: title={'center': 'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```

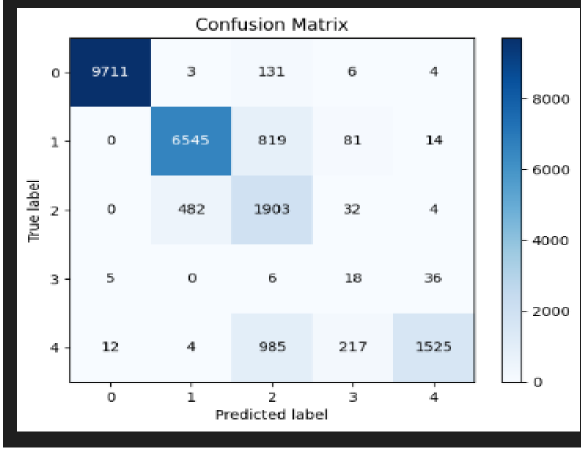


Figure 4.2 Evaluation for DecisionTree Classifier for NSL-KDD

```
print(classification_report(y_test, dt_pred)) # A value of 1 here gives attack_class == 1, that is, DDoS attacks.
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      9855
           1       0.93      0.88      0.90      7459
           2       0.50      0.79      0.61      2421
           3       0.05      0.28      0.09        65
           4       0.96      0.56      0.71      2743

    accuracy                           0.87     22543
   macro avg       0.69      0.70      0.66     22543
weighted avg       0.91      0.87      0.88     22543
```

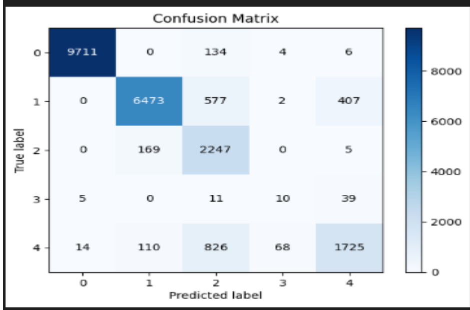
Figure 4.2 Evaluation for DecisionTree Classifier for NSL-KDD

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict and evaluate
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Model")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Model
Accuracy: 0.894557068713126
```

```
splt.plot_confusion_matrix(y_test, y_pred_rf)
```

```
<Axes: title={'center': 'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



True label \ Predicted label	0	1	2	3	4
0	9711	0	134	4	6
1	0	6473	577	2	407
2	0	169	2247	0	5
3	5	0	11	10	39
4	14	110	826	68	1725

```
print(classification_report(y_test, y_pred_rf))
```

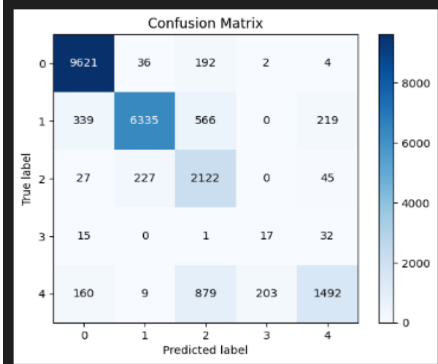
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      9855
           1       0.96      0.87      0.91      7459
           2       0.59      0.93      0.72      2421
           3       0.12      0.15      0.13        65
           4       0.79      0.63      0.70      2743

    accuracy                           0.89     22543
   macro avg       0.69      0.71      0.69     22543
weighted avg       0.91      0.89      0.90     22543
```

Figure 4.2 Evaluation for RandomForest Classifier for NSL-KDD

```
K-Nearest Neighbors Model
Accuracy: 0.8688728208313001
```

```
splt.plot_confusion_matrix(y_test, y_pred_knn)
```

```
<Axes: title={'center': 'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



True label \ Predicted label	0	1	2	3	4
0	9621	36	192	2	4
1	339	6335	566	0	219
2	27	227	2122	0	45
3	15	0	1	17	32
4	160	9	879	203	1492

```
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96      9855
           1       0.96      0.85      0.90      7459
           2       0.56      0.88      0.69      2421
           3       0.08      0.26      0.12        65
           4       0.83      0.54      0.66      2743

    accuracy                           0.87     22543
   macro avg       0.68      0.70      0.67     22543
weighted avg       0.89      0.87      0.87     22543
```

```
  warnings.warn(smsg, UserWarning)
XGBoost Model Performance:
R2 Score: 0.7273904047789977
Accuracy: 0.8903429002351062
Precision: 0.9116994816117503
Recall: 0.8903429002351062
F1 Score: 0.8948650293464091

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      9855
           1       0.97      0.86      0.91      7459
           2       0.57      0.91      0.70      2421
           3       0.13      0.12      0.13        65
           4       0.75      0.62      0.68      2743

    accuracy                           0.89     22543
   macro avg       0.69      0.70      0.68     22543
weighted avg       0.91      0.89      0.89     22543
```

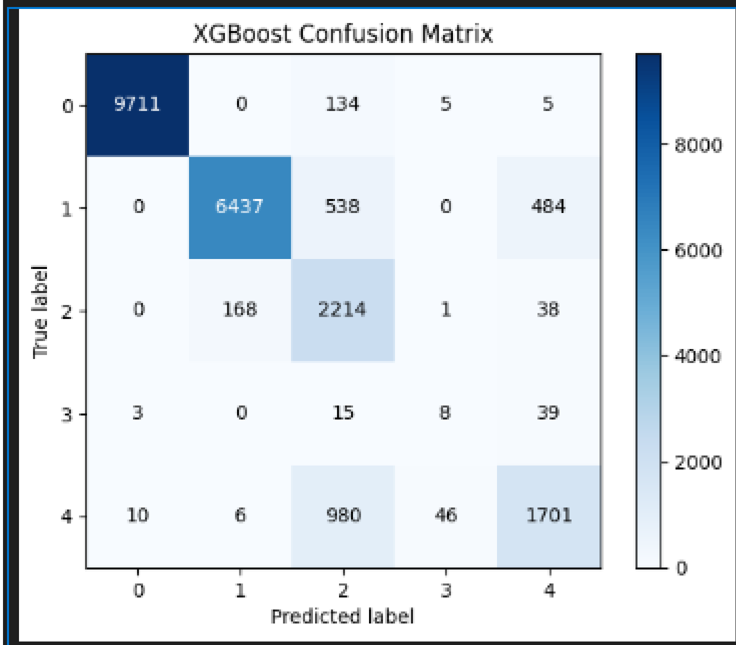


Figure 4.2 Evaluation for XGBoost for NSL-KDD

5 Conclusion and Future Work

This research work has shown how machine learning models such as XGBoost, Random Forest as well as SVM can be used to identify network intrusions using existing datasets such as NSL-KDD and UNSW-NB15. The findings revealed that these models should work well on large scale data, detect patterns in networks, and classify anomaly patterns with reasonable precision. It is worth to note that the processes of feature encoding, selection and normalization were crucial for improving the performances of the models. Hyper parameter optimization and validation also improved the generalization capabilities of the models from various input domains.• This system is one of the few that should emphasize the need for the use of machine learning when performing network intrusion detection.in detecting network intrusions using datasets like NSL-KDD and UNSW-NB15. The results showed that these models could handle large, complex datasets, identify patterns in network traffic, and classify anomalies with high accuracy. The preprocessing steps, including feature encoding, selection, and normalization, played a vital role in enhancing the models' efficiency. Hyperparameter tuning and cross-validation further ensured that the models performed reliably across diverse data distributions.

- The success of this system highlights the importance of employing machine learning for network intrusion detection. It also offers a solution to cyber threats that can help to identify a problem before it becomes a major issue in the network. There are also affordability issues; sometimes it might produce false positives; and it struggles to identify mutating patterns.
- As a future work, comparative analysis can be performed by incorporating higher level algorithms such as deep learning algorithms for better detection rates.s using datasets like NSL-KDD and UNSW-NB15. The results showed that these models could handle large, complex datasets, identify patterns in network traffic, and classify anomalies with high accuracy. The preprocessing steps, including feature encoding, selection, and normalization, played a vital role in enhancing the models' efficiency. Hyperparameter tuning and cross-validation further ensured that the models performed reliably across diverse data distributions.
- The success of this system highlights the importance of employing machine learning for network intrusion detection. It provides a proactive approach to cybersecurity, enabling early detection of threats and reducing the risk of network breaches. However, there are limitations, such as the potential for false positives in some scenarios and the challenges posed by evolving attack patterns.
- In the future, this work can be extended by exploring more advanced algorithms, such as deep learning models, to improve detection rates further. The appending of realtime data streams to the system can also assist to make the system more fluid and flexible in nature to new threats. Further, the incorporation of this system with cloud platforms could possibly take advantage of further flexibility compared to current applications and expand the areas of utilization. Further work on minimizing the number of false positives and enhancing the ability to better explain underlying machine learning algorithms will also be important to ensure that the implementation of IDSs becomes even more feasible and beneficial for detecting intrusions in realistic settings.

6 Glossary

- False Positive (FP): while the model provides an indication of an assault when traffic is normal.
- Feature encoding is basically conversion of non- quantitative (qualitative) data string or categories that are textual or categorical into a form that CAN be tackled by the machine learning algorithm.
- Feature Selection is the process by which only the relevant data points (features) are chosen from a dataset so as to enhance both the efficiency and performance of the

model.r data point that deviates from regular behaviour and frequently indicates a potential threat or attack in network traffic.

- assault Class: A number allocated to a certain sort of assault, such as DoS or Probe, to help machine learning models detect and describe it.
- A confusion matrix is a table that displays a model's performance by stating the number of correct and wrong predictions in each category.
- Cross-validation is a technique for determining how well a machine learning model works by dividing the dataset into smaller chunks and training and testing on each.
- A dataset is a collection of data used for training and testing machine learning models. The datasets utilised in this experiment are NSL-KDD and UNSW-NB15.
- False Positive (FP): while the model incorrectly predicts an assault while traffic is normal.
- Feature encoding is the process of transforming non-numerical (categorical) data, such as text or categories, into numerical values that the machine learning model can use.
- Feature Selection is the process of selecting only the most significant data points (features) from a dataset to improve the model's efficiency and accuracy.
- Tuning is the act of tweaking a machine learning model and its environment by changing the hyperparameters so as to give the best results
- An intrusion detection system (IDS) is a tool or system that detects and alerts to unauthorised network access or activity.
- Machine Learning (ML) is a sort of artificial intelligence in which computers learn from data to make decisions or predictions rather than being manually programmed for each task.
- Normalisation is a method of scaling data so that all values fall within a similar range, making it easier for the model to handle.
- Precision is a measure of how often the model's predictions of an assault are accurate. Consequently, high definition leads to reduction of false alarms.
- Accuracy is a measure of how many of the actual attacking events the learning model identifies.settings to improve its performance.
- An intrusion detection system (IDS) is a tool or system that detects and alerts to unauthorised network access or activity.
- Recall is a measure of how many actual attacks the model correctly identifies. Less missed attacks are noticed if the recall is high.
- SVM (Support Vector Machine): One of the machine learning approaches that are used in classifying data especially difficult and large data set.
- Confusion Matrix Metrics: Parameters that indicate the effectiveness of the model, these are the true positive which are the attacks identified as such, false positive which are the normal traffic that is labeled as an attack, the true negative is the normal traffic correctly identified and the false negative which are the attacks that were not detected.
- Overfitting: When a model learns the training data too well, and thus it becomes an overfitting model that does badly when tested on other data.

7 Acronyms

IDS: Intrusion Detection System
ML: Machine Learning

FP: False Positive
FN: False Negative
TP: True Positive
TN: True Negative
SVM: Support Vector Machine
NSL-KDD: Network Security Lab - Knowledge Discovery in Databases
UNSW-NB15: University of New South Wales - Network-Based 15 Dataset
ROC: Receiver Operating Characteristic
F1-Score: F1 Measure or Harmonic Mean of Precision and Recall
GPU: Graphics Processing Unit
CSV: Comma-Separated Values
API: Application Programming Interface
RF: Random Forest
XGBoost: Extreme Gradient Boosting
CSV: Comma-Separated Values
RFE: Recursive Feature Elimination