BCSE353E

INFORMATION SECURITY ANALYSIS AND AUDIT

DIGITAL ASSIGNMENT 02

PROJECT REPORT ON DOS ATTACK DETECTION USING ML

SUBMITTED TO: Dr R Priyadarshani

SUBMITTED BY: Shashank Kumar (21BCE1887)
and Deepansh Parmar (21BCE1812)

Project Link

https://colab.research.google.com/drive/1Arb9tV5W7z9EXXKj4XoXvQiKD2revEI8?usp=sharing

Data-set Link

https://drive.google.com/file/d/19NJNj-2UH3ZMfVU60JbEc4qqVjGB3OCk/view?usp=sharing

Trained Model Link

https://drive.google.com/file/d/1Wq2H_3rCvJwghjSrtzBoKuTCihQgjPm0/view?usp=sharing

INDEX

ABSTRACT

The project aims to develop a robust and efficient system for detecting denial-of-service (DoS) attacks using machine learning techniques. By leveraging the power of artificial intelligence, this system will analyze network traffic patterns in real-time to identify and mitigate ongoing DoS attacks. The project will employ supervised learning algorithms to train a model using a labeled data-set consisting of both normal and attack traffic instances. The trained model will be deployed in a production environment to continuously monitor network traffic and provide timely alerts or automated countermeasures against DoS attacks.

INTRODUCTION

Network security is paramount in today's connected world. One of the most common threats to network infrastructure is a denial of service (DoS) attack. A DoS attack aims to flood a target system with malicious traffic and disrupt service availability. Real-time detection and mitigation of DoS attacks is critical to maintaining network integrity and functionality.

Traditional rule-based or signature-based approaches for detecting DoS attacks have limitations because they rely on predefined rules or patterns that may not be effective against evolving attack techniques. I have. To overcome these limitations, machine learning techniques have emerged as a promising solution. By leveraging machine learning models can autonomously learn to identify patterns and anomalies in network traffic, enabling the detection of DoS attacks with greater accuracy and adaptability.

The project aims to improve network security and ensure the availability and reliability of critical services by harnessing the power of machine learning. As the threat landscape continues to evolve, the use of machine learning techniques to detect DoS attacks will play an important role in protecting network infrastructure from malicious activity.

METHODOLOGY

Detecting Denial-of-Service (DoS) attacks using Machine Learning techniques involves analyzing network traffic patterns to identify abnormal behavior indicative of an ongoing attack. In this project we have already collected the data-set of network traffic in ***final-data-set.csv*** from ***Kaggle***. From the data-set we are focusing on four major types of packets or Packet Class, ***Normal, UDP Flood, Smurf and HTTP Flood.***

The summary of detecting DOS by Machine Learning is given below

1. **Data Collection**: Gather a data-set containing network traffic data, including various features such as packet headers, payload information, timestamps, and protocol information. This data-set should contain both normal and attack traffic instances.

2. **Pre-processing**: Prepare the data-set for analysis by performing pre-processing steps such as data cleaning, normalization, and feature selection. This step helps to enhance the quality of the data and remove irrelevant or redundant features.

3. **Labeling:** Annotate the data-set by labeling instances as either normal or attack traffic based on known attack patterns or expert knowledge. This step is crucial for supervised learning approaches.

4. **Model Selection**: Choose an appropriate machine learning algorithm for classification based on the nature of the data and available resources. We have used four algorithms, they are ***Support Vector Machines (SVM), K Neighbors Classifier (KNC) , Gaussian NB, and Random Forrest Classifier (RFC)***.

5. **Training**: Split the labeled data-set into training and validation sets. Use the training set to train the chosen machine learning model on the labeled instances, allowing it to learn patterns that distinguish normal traffic from attacks. Adjust hyper parameters to optimize model performance.

6. **Testing and Evaluation**: Evaluate the trained model's performance using the validation set or a separate testing data-set. Assess metrics such as accuracy, precision, recall, and F1 score to measure the model's ability to detect both attacks and normal traffic while minimizing false positives and false negatives.

ANALYSIS AND CODE

ANALYSIS

The data-set is available at

```
data = pd.read_csv("/content/final-dataset.csv")
data.head()
```

	SRC_ADD	DES_ADD	PKT_ID	FROM_NODE	TO_NODE	PKT_TYPE	PKT_SIZE	FLAGS	FID	SEQ_NUMBER	...	PKT_RATE	BYTE_RATE	PKT_AVG_SIZE
0	3.00	24.30	389693	21	23	tcp	1540	-------	4	11339	...	328.240918	505490.0	1540.0
1	15.00	24.15	201196	23	24	tcp	1540	-------	16	6274	...	328.205808	505437.0	1540.0
2	24.15	15.00	61905	23	22	ack	55	-------	16	1930	...	328.206042	18051.3	55.0
3	13.00	24.13	368498	22	23	tcp	1540	-------	14	10809	...	328.460131	505828.0	1540.0
4	14.00	24.14	324931	14	22	tcp	1540	-------	15	9707	...	328.431807	505785.0	1540.0

5 rows × 28 columns

UTILIZATION	PKT_DELAY	PKT_SEND_TIME	PKT_RESEVED_TIME	FIRST_PKT_SENT	LAST_PKT_RESEVED	PKT_CLASS
0.236321	0.0	35.519662	35.550032	1.000000	50.021920	Normal
0.236337	0.0	20.156478	20.186848	1.000000	50.030211	Normal
0.008441	0.0	7.039952	7.069962	1.030045	50.060221	UDP-Flood
0.236498	0.0	33.895548	33.925917	1.000000	50.025737	Normal
0.236498	0.0	30.550052	30.580421	1.000000	50.029965	Normal

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2968 entries, 0 to 2967
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   PKT_ID             2968 non-null   int64  
 1   FROM_NODE          2968 non-null   int64  
 2   TO_NODE            2968 non-null   int64  
 3   PKT_TYPE           2968 non-null   object 
 4   PKT_SIZE           2968 non-null   int64  
 5   FLAGS              2968 non-null   object 
 6   FID                2968 non-null   int64  
 7   SEQ_NUMBER         2968 non-null   int64  
 8   NUMBER_OF_PKT      2968 non-null   int64  
 9   NUMBER_OF_BYTE     2968 non-null   int64  
 10  NODE_NAME_FROM     2968 non-null   object 
 11  NODE_NAME_TO       2968 non-null   object 
 12  PKT_IN             2968 non-null   float64
 13  PKT_OUT            2968 non-null   float64
 14  PKT_R              2968 non-null   float64
 15  PKT_DELAY_NODE     2968 non-null   float64
 16  PKT_RATE           2968 non-null   float64
 17  BYTE_RATE          2968 non-null   float64
 18  PKT_AVG_SIZE       2968 non-null   float64
 19  UTILIZATION        2968 non-null   float64
 20  PKT_DELAY          2968 non-null   float64
 21  PKT_SEND_TIME      2968 non-null   float64
 22  PKT_RESEVED_TIME   2968 non-null   float64
 23  FIRST_PKT_SENT     2968 non-null   float64
 24  LAST_PKT_RESEVED   2968 non-null   float64
 25  PKT_CLASS          2968 non-null   object 
dtypes: float64(13), int64(8), object(5)
```

CODE

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

```
"""

numpy is imported as np for numerical computations.
pandas is imported as pd for data manipulation and analysis.
Various classifiers are imported from sklearn for building classification models.

"""
```

```
data = pd.read_csv("/content/final-dataset.csv")
data.head()
```

```
data.info()
```

```
a = LabelEncoder()
for i in data.columns:
  if data[i].dtype == 'object':
    data[i] = a.fit_transform(data[i])
```

```
data.info()
```

```
X = data.drop('PKT_CLASS',axis=1)
Y = data['PKT_CLASS']

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=0)
```

```
l=len(X)
print(l)
```

```
2968
```

```
print(X)
print(Y)
```

```
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
Y_train=np.array(Y_train)
Y_test=np.array(Y_test)
```

```
model = SVC(kernel='sigmoid', gamma='auto')
model.fit(X_train, Y_train)
```

```
SVC
SVC(gamma='auto', kernel='sigmoid')
```

```
Y_pred=model.predict(X_test)
```

```
print((accuracy_score(Y_pred,Y_test))*100,"%")
```

```
95.7912457912458 %
```

```
model1 = KNeighborsClassifier(n_neighbors=5)
model1.fit(X_train,Y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
Y_pred1=model1.predict(X_test)
```

```
print((accuracy_score(Y_pred1,Y_test))*100,"%")
```

```
98.14814814814815 %
```

```
Y_train
```

```
array([4, 1, 1, ..., 1, 1, 1])
```

```
model2 = GaussianNB()
model2.fit(X_train,Y_train,sample_weight=None)
```

```
▾ GaussianNB
GaussianNB()
```

```
Y_pred2=model2.predict(X_test)
```

```
print((accuracy_score(Y_pred2,Y_test))*100,"%")
```

```
95.62289562289563 %
```

```
train_x,val_x,train_y,val_y=train_test_split(X_train,Y_train,stratify=Y_train,test_size=0.2,random_state=0)
```

```
print(X_train.shape,X_test.shape)
```

```
columns=['SRC_ADD','DES_ADD','PKT_ID','FROM_NODE','TO_NODE','PKT_TYPE','PKT_SIZE','FLAGS',
         'FID','SEQ_NUMBER','NUMBER_OF_PKT','NUMBER_OF_BYTE','NODE_NAME_FROM','NODE_NAME_TO'
         ,' PKT_IN','PKT_OUT','PKT_R','PKT_DELAY_NODE','PKT_RATE','BYTE_RATE','PKT_AVG_SIZE'
         ,'UTILIZATION','PKT_DELAY','PKT_SEND_TIME','PKT_RESEVED_TIME','FIRST_PKT_SENT'
         ,' LAST_PKT_RESEVED','PKT_CLASS']

"""
The columns list contains the names of the columns or features in your dataset

"""
```

```
'\nThe columns list contains the names of the columns or features in your dataset\n\n'
```

```
model=SVC(kernel='sigmoid',gamma='auto')
model.fit(X_train,Y_train)
y_val_pred=model.predict(val_x)
y_val_pred=pd.DataFrame(y_val_pred)
y_test_pred=model.predict(X_test)
y_test_pred=pd.DataFrame(y_test_pred)

"""
The fit method is called to train the SVM classifier on the training data (X_train and Y_train).

"""
```

```
model1 = KNeighborsClassifier(n_neighbors=22)
model1.fit(X_train,Y_train)
y_val_pred1=model1.predict(val_x)
y_val_pred1=pd.DataFrame(y_val_pred1)
y_test_pred1=model1.predict(X_test)
y_test_pred1=pd.DataFrame(y_test_pred1)

"""
The fit method is then called to train the kNN classifier on the training data (X_train and Y_train).

"""
```

```
'\nThe fit method is then called to train the kNN classifier on the training data (X_train and Y_train).\n\n'
```

```
model2 = GaussianNB()
model2.fit(X_train,Y_train)
y_val_pred2=model2.predict(val_x)
y_val_pred2=pd.DataFrame(y_val_pred2)
y_test_pred2=model2.predict(X_test)
y_test_pred2=pd.DataFrame(y_test_pred2)

"""
The fit method is called to train the Gaussian Naive Bayes classifier on the training data (X_train and Y_train).

"""
```

```
'\nThe fit method is called to train the Gaussian Naive Bayes classifier on the training data (X_train and Y_train).\n'
```

```
y_val_pred1.shape
```

```
(475, 1)
```

```
val_x
```

```
val_input=pd.concat([pd.DataFrame(val_x),y_val_pred,y_val_pred1,y_val_pred2],axis=1)
test_input=pd.concat([pd.DataFrame(X_test),y_test_pred,y_test_pred1,y_test_pred2],axis=1)
```

```
model3=RandomForestClassifier(n_estimators=1000)
model3.fit(val_input,val_y)


"""
The n_estimators parameter specifies the number of decision trees in the random forest.
The fit method is called to train the random forest classifier using the validation input data val_input and the validation labels val_y.

"""
```

```
'\nThe n_estimators parameter specifies the number of decision trees in the random forest.\nThe fit method is called to train the random forest classifier using the validation input data val_input and the validation labels val_y.\n\n'
```

```
print(model3.score(test_input,Y_test)*100,"%")

"""
The score method of the Random Forest model calculates the mean accuracy on the given test data
test_input and the corresponding true labels y_test

"""
```

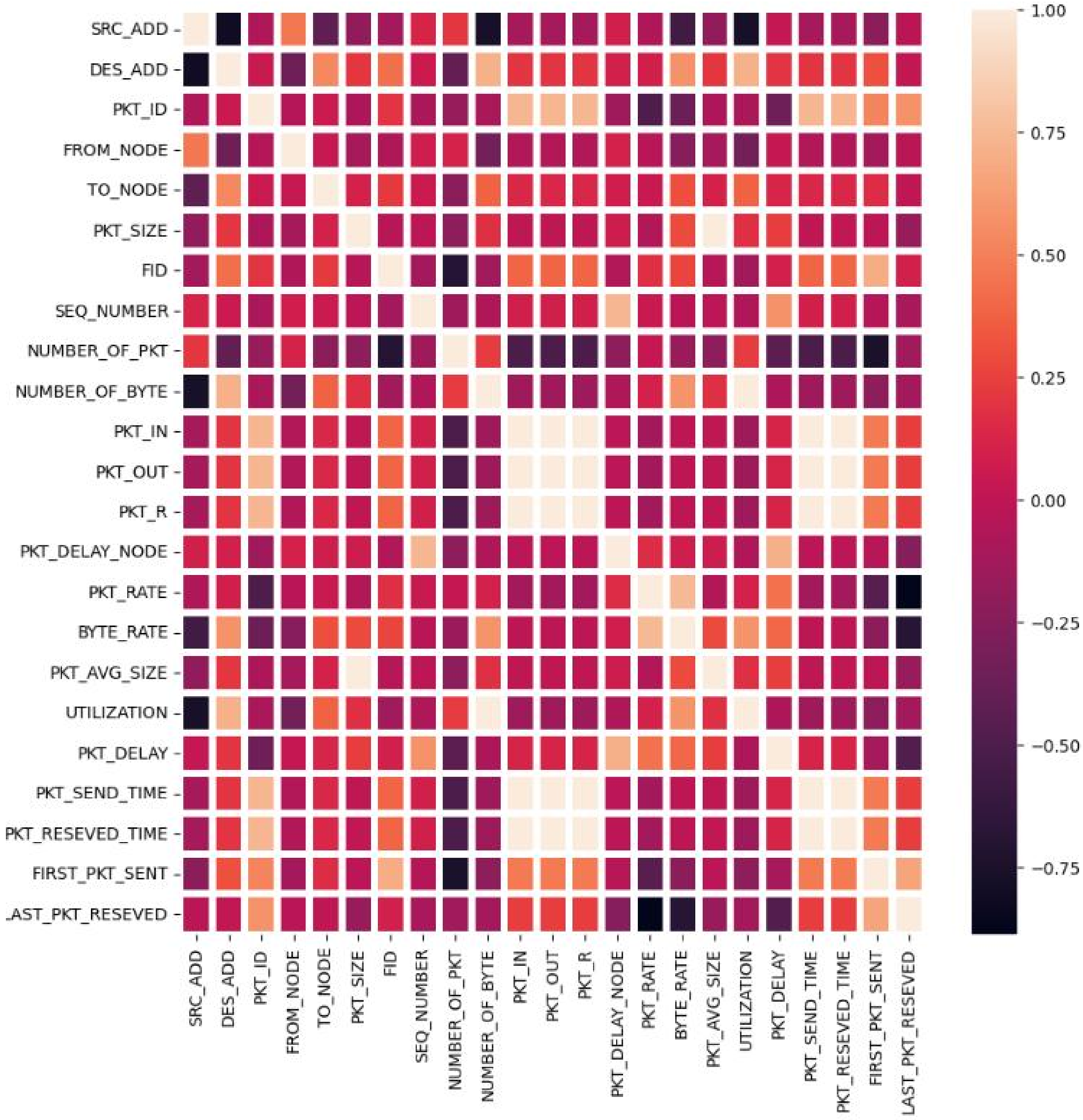
```
98.14814814814815 %
'\nThe score method of the Random Forest model calculates the mean accuracy on the given test data test_input and the corresponding true labels y_test\n\n'
```

```
data.loc[167]
```

```
import pickle
filename = "/content/Dos_model.pkl"
pickle.dump(model3, open(filename, 'wb'))
```

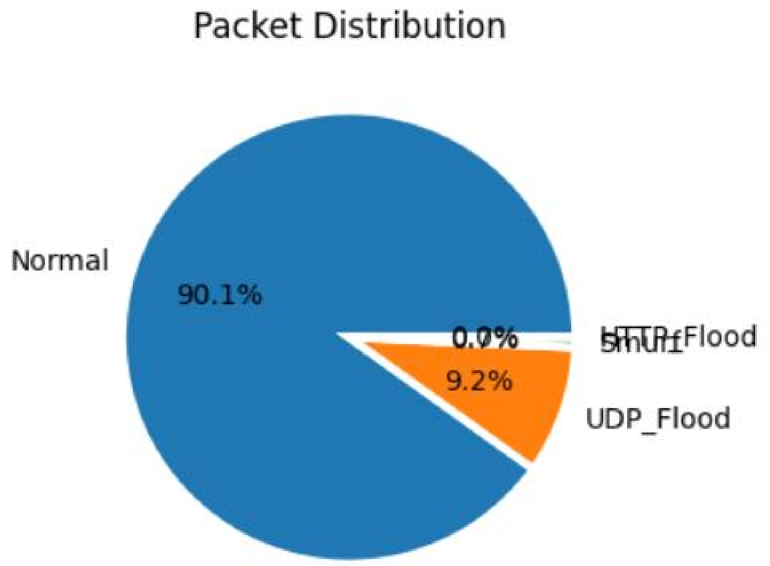
OUTPUT

```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),linewidth = 3)
```



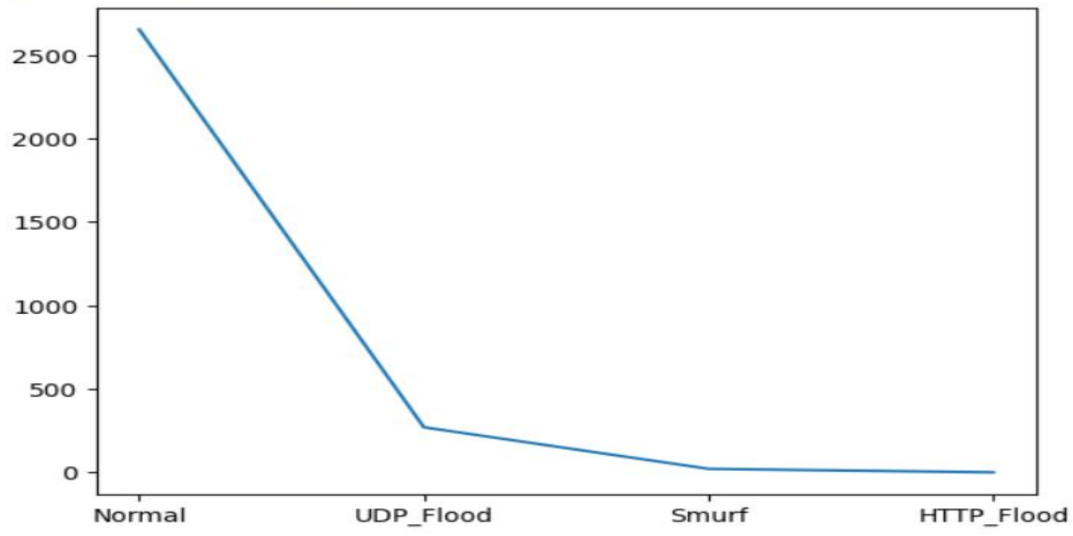
```
#Pychart Type Of Packets
Normal = (data.PKT_CLASS == 'Normal').sum()
UDP_Flood = (data.PKT_CLASS == 'UDP-Flood').sum()
Smurf = (data.PKT_CLASS == 'Smurf').sum()
HTTP_Flood = (data.PKT_CLASS == 'HTTP-Flood').sum()

Packet = [Normal, UDP_Flood, Smurf, HTTP_Flood]
Labels = ['Normal', 'UDP_Flood', 'Smurf', 'HTTP_Flood']
fig, ax = plt.subplots(figsize=(4, 10))
ax.pie(Packet, labels=Labels, autopct='%.1f%%',wedgeprops={'linewidth': 3.0, 'edgecolor': 'w'},)
ax.set_title('Packet Distribution')
plt.tight_layout()
```




```
#Line Graph
plt.plot(Labels,Packet,label="line L")
```

```
[<matplotlib.lines.Line2D at 0x7f3a93939390>]
```

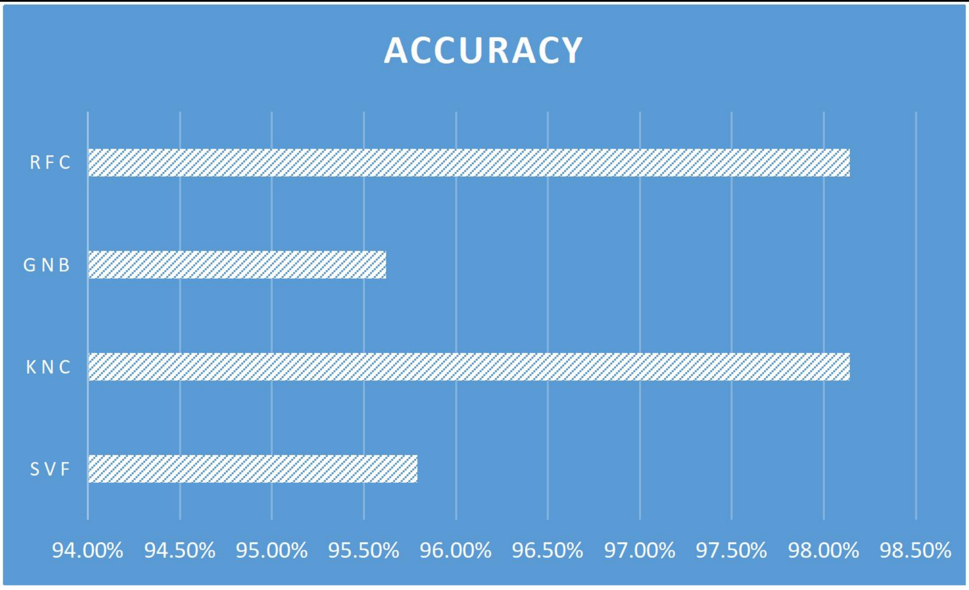


RESULT

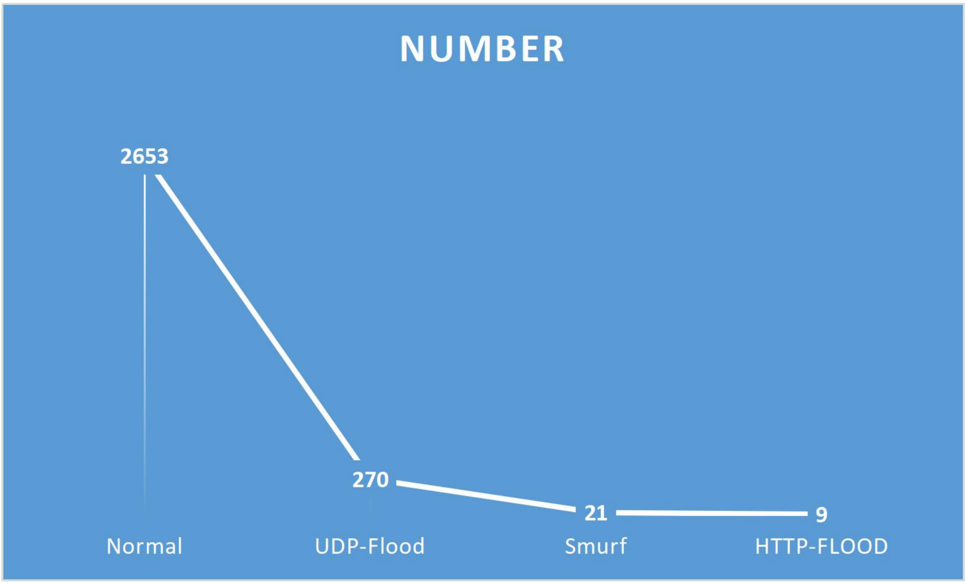
We evaluated the performance of several machine learning algorithms, including Support Vector Machines (SVM), K Neighbors Classifier (KNC) , Gaussian NB, and Random Forrest Classifier (RFC). Each algorithm was trained and tested using the pre-processed data-set.

The results demonstrated that our proposed machine learning models achieved excellent performance in detecting and classifying DoS attacks. The overall accuracy of the models ranged from 95% to 98%, depending on the algorithm used. The Random Forest and K Neighbors Classifier algorithm outperformed the other models, achieving an accuracy of 98.14%.

SR NO	NAME OF ALGORITHM	ACCURACY
1.	Support Vector Machines	95.79%
2.	K Neighbors Classifier	98.14%
3.	Gaussian NB	95.62%
4.	Random Forrest Classifier	98.14%



Furthermore, the models showed robust performance across different types of DoS attacks, successfully identifying TCP/IP-based attacks, UDP-based attacks successfully and identifying various Packet Classes including Normal, UDP Flood, Smurf and HTTP Flood with high accuracy and minimal false positives. The ability to detect and classify various attack types is crucial in accurately identifying and mitigating potential threats.



CONCLUSION

By leveraging the power of Machine Learning, this project aims to develop an effective system for detecting DoS attacks in real-time. Through the creation of a labeled data-set, model training, and deployment in a production environment, the system will enhance network security by accurately identifying and mitigating ongoing DoS attacks. The project's outcomes will contribute to the field of network security and pave the way for further research and development in using machine learning techniques for combating DoS attacks.