NETWORK INTRUSION DETECTION SYSTEM

PROJECT REPORT

Submitted for

CSE3501-INFORMATION SECURITY ANALYSIS AND AUDIT

Submitted by

20BCB0124 – SHUBHAM JAISWAL

20BCB0129-K SAI SRI HARSHA

20BCB0132-LAKSHYA VASWANI

20BCB0134-RITIKA MARLAPALLI

20BCB0142-KAVYA SAGUTURU

Submitted to

FACULTY -SWETHA NG

Implementation link:

https://colab.research.google.com/drive/1NYmXg_94VWVOW4fx_OtliJultQlbIgFL?usp=sharing

Abstract:

Several studies have suggested that by selecting relevant features for a threat detection system, it is possible to considerably improve the detection accuracy and performance of the detection engine. Nowadays with the emergence of new technologies such as Cloud Computing or Big Data, large amounts of network traffic are generated and the threat detection system must dynamically collect and analyze the data produced by the incoming traffic. However in a large dataset not all features contribute to represent the traffic, therefore reducing and selecting a number of adequate features may improve the speed and accuracy of the threat detection system.

A feature selection mechanism has been proposed which aims to eliminate non-relevant features as well as identify the features which will contribute to improve the detection rate, based on the score each feature has established during the selection process. To achieve that objective, a recursive feature elimination process was employed and associated with a decision tree based classifier and later on, the suitable relevant features were identified. This approach was applied on the NSL-KDD dataset which is an

improved version of the previous KDD 1999 Dataset, scikit-learn that is a machine learning library written in python that is being planned to use.

Using this approach, relevant features can be identified inside the dataset and the accuracy rate can be improved. Understanding the factors that help identify relevant features will allow the design of a better threat detection system.

Introduction

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network

or a system for harmful activity or policy breaching.

Network intrusion detection systems (NIDS) are set up at a planned point within the network to examine traffic from all devices on the network. It performs an observation of passing traffic on the entire subnet and matches the traffic that is passed on the subnets to the collection of known attacks. Once an attack is identified or abnormal behavior is observed, the alert can be sent to the administrator.

In Data preprocessing all features are made numerical using one-Hot-encoding. The features are scaled to avoid features with large values that may weigh too much in the results. A one hot
encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers.
One-Hot-Encoding transforms all categorical features into binary features. Requirement for One-Hot-encoding: The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature. It is assumed that input features take on values in the range [0, n_values). Therefore the features first need to be transformed with Label Encoder, to transform every category to a number.

In Feature Selection Eliminate redundant and irrelevant data by selecting a subset of relevant features that fully represents the given problem Univariate feature selection with ANOVA F-test. This analyzes each feature individually to determine the strength of the relationship between the feature and labels. Using the Second Percentile method to select features based on percentile of the highest scores. When this subset is found: Recursive Feature Elimination (RFE) is applied.

Using the test data to make predictions of the model. Multiple scores are considered such as accuracy score, recall, f-measure, confusion matrix. perform a 10-fold cross-validation.

Problem Statement:

Using relevant features from the dataset apply appropriate feature selection mechanisms to implement a intuition detection system to implement a faster system with increased accuracy.

Implementation:

A security mechanism can be implemented using an Intrusion Detection System (IDS) which can be described as a collection of software or hardware devices able to collect, analyze and detect any unwanted, suspicious or malicious traffic either on a particular computer host or network. Therefore to achieve its task, an IDS should use some statistical or mathematical method to read and interpret the information it collects and subsequently reports any malicious activity to the network administrator.

Using Machine Learning Approach:

A data clean process can require a tremendous human effort, which is an extensive time consuming and expensive. A machine learning approach and data mining technique which is the application of machine learning methods to large databases are widely known and used to reduce or eliminate the need of a human interaction.

Machine learning helps to optimize performance criterion using example data or past experience using a computer program, models are defined with some parameters, and learning is the execution of the programming computer to optimize the parameters of the model using training data. The model can be predictive to make predictions in the future, or descriptive to gain knowledge from data. To perform a predictive or descriptive task, machine learning generally use two main techniques: Classification and Clustering.

All categories are described below:

1) Basic features: It contains all features which derived from TCP/IP connection such as Protocol_type, Service, duration etc.

2) Content features: Those features use domain knowledge to access the payload of the original TCP packets (e.g. host, num_root, is_guest_login and etc.)

3) Host-based traffic features: all attacks which span longer than 2 second intervals that have the same destination host as the current connection are accessed using these features (e.g. dst_host_count, dst_host_srv_count and etc.)

NSL KDD dataset new: -

1) Elimination of redundant records in the training set will help our classifier to be unbiased towards more frequent records.

2) No presence of duplicate records in the test set, therefore, the classifier performance will not be biased by the techniques which have better detection rates on the frequent records.

3) The training and test set contains both a reasonable number of instances which is affordable for experiments on the entire set without the need to randomly choose a small portion.

Proposed Method

Step 1: Data Cleaning and Preprocessing

Step 2: Features scaling

Step 3: Features Selection: Using ANOVA F TEST

Step 4: Model

Here, a decision tree model can be built to partition the data using information gain until instances in each leaf node have uniform class labels. This is a very simple but yet an effective hierarchical method for supervised learning (classification or regression) whereby the local space (region) is recognized in a sequence of repetitive splits in a reduced number of steps (small). At each test, a single feature is used to split the node according to the feature values.

The generation process of a decision tree done by recursively splitting on features is equivalent to dividing the original training set into smaller sets recursively until the entropy of every one of these subsets is zero (i.e everyone will have instances from a single class target)

A Decision Tree is made up of internal decision nodes and terminal leaves. A test function is implemented by each decision node with discrete results labelling the branches. Providing an input, at every node, a test is constructed and based on the outcome, one of the branches will be considered. Here the learning algorithm starts at the root and until a leaf node is reached.

Literature Review:

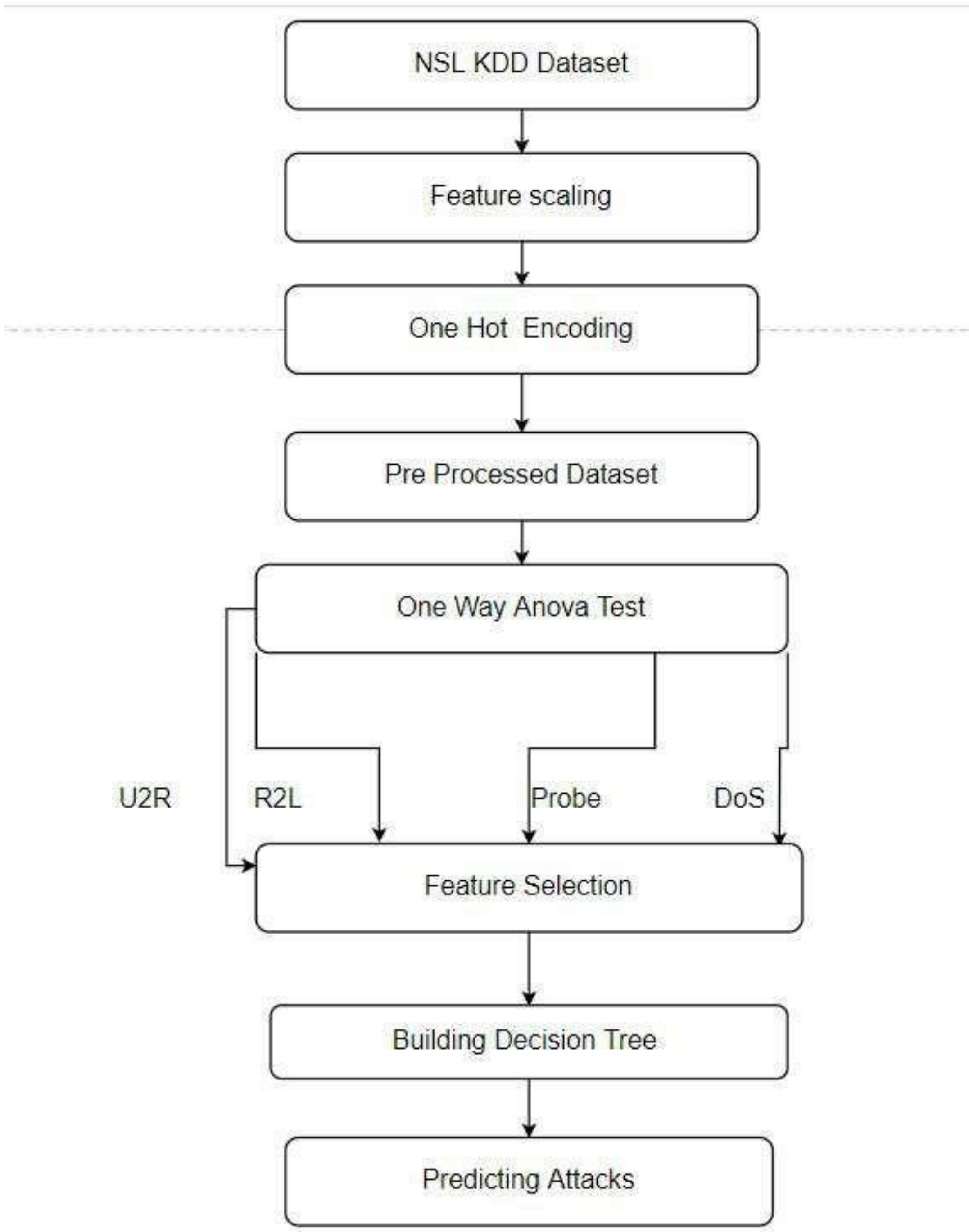
S.NO	TITLE/AUTHOR	ABSTRACT	METHODOLOGY	RESULT
1.	Big Data Analytics for Intrusion Detection 2020 Author: Luis, Filipe Dias, Miguel Correia	This chapter presents a survey on recent work on big data analytics for intrusion detection. This approach is promising for dealing with cyber security's big data problem. Rather than relying on human analysts to create signatures or classify huge volumes of data, Machine Learning can be used. Machine Learning allows the implementation of advanced algorithms to extract information from data using behavioral analysis or to find hidden correlations. However, the adversarial setting and the dynamism of the cyber threat landscape stand as difficult challenges when applying ML.	Machine Learning	Provide security monitoring with the means for automation, orchestration and real-time contextual threat awareness.

2.	Autonomic Intrusion Detection and Response Using Big Data 2019 Author: Kleber Vieira, Fernando Koch, João Bosco Mangueira Sobral, Jorge Lopes de Souza Leao, Carlos Becker Westphall	In this paper they present a method for autonomic intrusion detection and response to optimize processes of cyber security in large distributed systems. These environments are characterized by technology fragmentation and complex operations making them highly susceptible to attacks like hijacking, man-in-the-middle, denial-of-service, phishing, and others. The autonomic intrusion response system introduces models of operational analysis and reaction based on the combination of autonomic computing and big data.	fragmentation and complex operations	Improvement in effectiveness and scalability of the method in complex environments.
3.	Big Data Processing for Intrusion Detection System Context: A Review 2021 Author: Elayni Marwa, Farah Jemili, Ouajdi Korbaa, Basel Solaiman	The rapid growth of data, the increasing number of network based applications, and the advent of the omnipresence of internet and connected devices have affected the importance of information security. Hence, a security system such as an Intrusion Detection System (IDS) becomes a fundamental requirement. However, the complexity of the generated data and their huge size, plus, the variation of Cyber-attacks on: the network traffic, wireless network traffic, worldwide network traffic, connected devices and 5 G communication media, lead to hinder the IDS's efficiency.	Bid data technologies	Offers a review of IDS that deploy big data technologies and provides interesting recommendations
4.	A Big Data Analytical Approach to Cloud Intrusion Detection 2018 Author: Emrah Tuncel, Pelin Angin	Advances in cloud computing in the past decade have made it a feasible option for the high performance computing and mass storage needs of many enterprises due to the low startup and management costs. Due to this prevalent use, cloud systems have become hot targets for attackers aiming to disrupt reliable operation of large enterprise systems. The variety of attacks launched on cloud systems, including zero-day attacks that these systems are not prepared for, call for a unified approach for real-time detection and mitigation to provide increased reliability.	big data analytical, neural network	Detect deviations from the normal behavior of cloud systems in near real-time and introduce measures to ensure reliable operation of the system
5.	Addressing big data analytics for classification intrusion	Thus, enhancing the intrusion detection system is main object of numbers of research and developers for monitoring the	NSL-KDD standard, Support Vector Machine, Preprocessing,	Empirical results of hybrid outperformed the performance

	detection system 2020 Author: Keyan Abdul Aziz Mutlaq Alsibahi, Hadi Hussein Madhi, Hassanain Raheem	network security. Addressing challenges of big data in intrusion detection is one issue faced the researchers and developers due to dimensionality reduction in network data. In this paper, hybrid model is proposed to handle the dimensionality reduction in intrusion detection system.	genetic algorithm	
6.	Enhanced Big Data in Intrusion Detection System by Machine Learning 2020 Author: Saqr Mohammed Almansob, Shubhada Bhosale, Akram Alsubari, Santosh S Lomte	The Principal Component Analysis (PCA) is applied to reduce data dimension while preserving information of original data and building intrusion detection models by using K-nearest neighbors (K-NN) and Support Vector Machine (SVM) classifier to evaluate the accuracy of the system. The researcher has applied KDD99 datasets to train and test the model. Thus, the researcher has introduced a comparison between the K-NN classifier and SVM classifier. The results of the experiment showed that the SVM classifier high performance.	K-nearest neighbors (K-NN) and Support Vector Machine (SVM)	The experiment showed that the SVM classifier high performance, which reduces the false positive rate as well as giving high accuracy and is efficient for Big Data.
7.	Wireless Network Intrusion Detection System 2019 Author: Calvin Jia Liang	The Wireless Network Intrusion Detection System is a network-based intrusion detection system (IDS) that listens on a wireless network. The device has two network interfaces: the wireless interface is used to monitor network traffic, and the wired interface is used to configure the system and to send out detection alerts. The system requires minimal setup, configuration, and maintenance. It is a relatively inexpensive device that tries to improve user's situational-awareness of one's wireless network.	Wireless IDS	The IDS device is a self-contained single-board-computer capable of monitoring the user's wireless network, detecting suspicious network traffic, and reporting to the user via email.
8.	Network Intrusion Detection System (NIDS) using data mining techniques. 2019 Authors: Bane	This paper introduces the Network Intrusion Detection System (NIDS), which uses a suite of data mining techniques to automatically detect attacks against computer networks and systems. This paper focuses on two specific contributions: (i) an unsupervised anomaly detection technique that assigns a score to each network connection that	Data mining 1.Unsupervise anomaly detection. 2.Association pattern analysis.	Experimental results show that our anomaly detection techniques are successful in automatically detecting several intrusions that could not be identified using

	Raman Raghunath, Nitin Shivsharan	reflects how anomalous the connection is, and (ii) an association pattern analysis based module that summarizes those network connections that are ranked highly anomalous by the anomaly detection module.		popular signature-based tools .Furthermore, given the very high volume of connections observed per unit time, association pattern based summarization of novel attacks is quite useful in enabling a security analyst to understand and characterize emerging threats.
9.	High Performance Network Intrusion Detection System 2020 Authors: R. Madana Mohana, Anita Bai, Delshi Ramamoorthy	In this paper, we present intrusion detection system for finding the variant types of attacks in the network. It is the way to enhance the functionality in the network by reducing the chances of risks. ICMP protocol and AES encryption algorithm are used to report the error messages and manage the information being sent from source to destination. If there is any malicious activity occurred in the network, the user will be alerted of it by specifying them the type of malicious activity.	ICMP protocol and AES encryption algorithm.	As a result it reduces the chances of intrusions and contacting multiple resources for resolving single issue.
10.	Fuzzy Logic And Network Intrusion Detection System. 2020 Authors: Bhawna Sinha, Shiv Shanker Sahay, Braj Kishor Prasad.	Intrusion Detection System which are the key part of system defense are used to identify abnormal activities in a computer system. In general, the traditional intrusion detection relies on the extensive knowledge of security experts, in particular, on their familiarity with the computer system to be protected. Currently available intrusion detection systems focus mainly on determining uncharacteristic system events in distributed networks using signature based approach. Due to its limitation of finding novel attacks, we propose a hybrid model based on improved fuzzy and data mining techniques, which can detect both misuse and anomaly attacks.	Fuzzy logic	In the proposed system effectively identifying the intrusion activities within a network, detect an intrusion behavior of the networks.

Design Diagram:



Modules implemented and their snapshots:

Loading Dataset:

```
[4] # attach the column names to the dataset
    col_names = ["duration","protocol_type","service","flag","src_bytes",
        "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
        "logged_in","num_compromised","root_shell","su_attempted","num_root",
        "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
        "is_host_login","is_guest_login","count","srv_count","serror_rate",
        "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
        "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
        "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
        "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
        "dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]

    # KDDTrain+_2.csv & KDDTest+_2.csv are the datafiles without the last column about the difficulty score
    # these have already been removed.
    df = pd.read_csv("KDDTrain+_2.csv", header=None, names = col_names)
    df_test = pd.read_csv("KDDTest+_2.csv", header=None, names = col_names)

    # shape, this gives the dimensions of the dataset
    print('Dimensions of the Training set:',df.shape)
    print('Dimensions of the Test set:',df_test.shape)
```

Output:-

```
Dimensions of the Training set: (125973, 42)
Dimensions of the Test set: (22544, 42)
```

Elimination of redundant records in the training set will help our classifier to be unbiased towards more frequent records.

So here by eliminating reductant data from Training set we got Test set

Statistical Summary:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compro
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.0
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111	0.204409	0.001222	0.395736	0.2
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366	2.149968	0.045239	0.489010	23.9
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.0
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	77.000000	5.000000	1.000000	7479.0

8 rows × 38 columns

Sample view of our Dataset:

```python
# first five rows
df.head(5)
```

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

5 rows × 42 columns

Identify categorical features:

Input:

```python
# colums that are categorical and not binary yet: protocol_type (column 2), service (column 3), flag (column 4).
# explore categorical features
print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

```python
# Test set
print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

Output:

```
Training set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 70 categories
Feature 'flag' has 11 categories
Feature 'label' has 23 categories
```

```
Test set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 64 categories
Feature 'flag' has 11 categories
Feature 'label' has 38 categories
```

Label Encoder:

Input:

LabelEncoder

Insert categorical features into a 2D numpy array

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']
# insert code to get a list of categorical columns into a variable, categorical_columns
categorical_columns=['protocol_type', 'service', 'flag']
 # Get the categorical values into a 2D numpy array
df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]
df_categorical_values.head()
```

Output:

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF

Input:

Transform categorical features into numbers using LabelEncoder()

```
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
print(df_categorical_values_enc.head())
# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transform)
```

Output:

```
   protocol_type  service  flag
0              1       20     9
1              2       44     9
2              1       49     5
3              1       24     9
4              1       24     9
```

One-Hot-Encoding:

Input:

One-Hot-Encoding

```
enc = OneHotEncoder()
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)
# test set
testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)

df_cat_data.head()
```

Output:

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_aol	service_auth	service_bgp	service_
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 84 columns

Feature Scaling:

Input:

```
# Split dataframes into X & Y
# assign X as a dataframe of feautures and Y as a series of outcome variables
X_DoS = DoS_df.drop('label', axis=1)
Y_DoS = DoS_df.label
X_Probe = Probe_df.drop('label', axis=1)
Y_Probe = Probe_df.label
X_R2L = R2L_df.drop('label', axis=1)
Y_R2L = R2L_df.label
X_U2R = U2R_df.drop('label', axis=1)
Y_U2R = U2R_df.label
# test set
X_DoS_test = DoS_df_test.drop('label', axis=1)
Y_DoS_test = DoS_df_test.label
X_Probe_test = Probe_df_test.drop('label', axis=1)
Y_Probe_test = Probe_df_test.label
X_R2L_test = R2L_df_test.drop('label', axis=1)
Y_R2L_test = R2L_df_test.label
X_U2R_test = U2R_df_test.drop('label', axis=1)
Y_U2R_test = U2R_df_test.label
```

Output:

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1.]
```

Feature Selection:

1. Univariate Feature Selection using ANOVA F-test

```
[28] #univariate feature selection with ANOVA F-test. using secondPercentile method, then RFE
     #Scikit-learn exposes feature selection routines as objects that implement the transform method
     #SelectPercentile: removes all but a user-specified highest scoring percentage of features
     #f_classif: ANOVA F-value between label/feature for classification tasks.
     from sklearn.feature_selection import SelectPercentile, f_classif
     np.seterr(divide='ignore', invalid='ignore');
     selector=SelectPercentile(f_classif, percentile=10)
     X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
     X_newDoS.shape
```

Input:

Get the features that were selected: **DoS**

```
true=selector.get_support()
newcolindex_DoS=[i for i, x in enumerate(true) if x]
newcolname_DoS=list( colNames[i] for i in newcolindex_DoS )
newcolname_DoS
```

Output:

```
['logged_in',
 'count',
 'serror_rate',
 'srv_serror_rate',
 'same_srv_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_serror_rate',
 'dst_host_srv_serror_rate',
 'service_http',
 'flag_S0',
 'flag_SF']
```

Input:

Get the features that were selected: **Probe**

```
true=selector.get_support()
newcolindex_Probe=[i for i, x in enumerate(true) if x]
newcolname_Probe=list( colNames[i] for i in newcolindex_Probe )
newcolname_Probe
```

Output:

```
['logged_in',
 'rerror_rate',
 'srv_rerror_rate',
 'dst_host_srv_count',
 'dst_host_diff_srv_rate',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'dst_host_rerror_rate',
 'dst_host_srv_rerror_rate',
 'Protocol_type_icmp',
 'service_eco_i',
 'service_private',
 'flag_SF']
```

Input:

Get the features that were selected: **R2L**

```
true=selector.get_support()
newcolindex_R2L=[i for i, x in enumerate(true) if x]
newcolname_R2L=list( colNames[i] for i in newcolindex_R2L)
newcolname_R2L
```

Output:

```
['src_bytes',
 'dst_bytes',
 'hot',
 'num_failed_logins',
 'is_guest_login',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp',
 'service_ftp_data',
 'service_http',
 'service_imap4',
 'flag_RSTO']
```

Input:

Get the features that were selected: **U2R**

```
[35] true=selector.get_support()
     newcolindex_U2R=[i for i, x in enumerate(true) if x]
     newcolname_U2R=list( colNames[i] for i in newcolindex_U2R)
     newcolname_U2R
```

Output:

```
['urgent',
 'hot',
 'root_shell',
 'num_file_creations',
 'num_shells',
 'srv_diff_host_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp_data',
 'service_http',
 'service_telnet']
```

SUMMARY:

Summary of features selected by Univariate Feature Selection

```
[36] print('Features selected for DoS:',newcolname_DoS)
     print()
     print('Features selected for Probe:',newcolname_Probe)
     print()
     print('Features selected for R2L:',newcolname_R2L)
     print()
     print('Features selected for U2R:',newcolname_U2R)
```

```
Features selected for DoS: ['logged_in', 'count', 'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_

Features selected for Probe: ['logged_in', 'rerror_rate', 'srv_rerror_rate', 'dst_host_srv_count', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_ra

Features selected for R2L: ['src_bytes', 'dst_bytes', 'hot', 'num_failed_logins', 'is_guest_login', 'dst_host_srv_count', 'dst_host_same_src_port_rate'

Features selected for U2R: ['urgent', 'hot', 'root_shell', 'num_file_creations', 'num_shells', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_co
```

Building Decision tree:

Classifier is trained for all features and for reduced features, for later comparison. The classifier model itself is stored in the clf variable.

```
[47] # all features
clf_DoS=DecisionTreeClassifier(random_state=0)
clf_Probe=DecisionTreeClassifier(random_state=0)
clf_R2L=DecisionTreeClassifier(random_state=0)
clf_U2R=DecisionTreeClassifier(random_state=0)
clf_DoS.fit(X_DoS, Y_DoS)
clf_Probe.fit(X_Probe, Y_Probe)
clf_R2L.fit(X_R2L, Y_R2L)
clf_U2R.fit(X_U2R, Y_U2R)
```

```
DecisionTreeClassifier(random_state=0)
```

```
# selected features
clf_rfeDoS=DecisionTreeClassifier(random_state=0)
clf_rfeProbe=DecisionTreeClassifier(random_state=0)
clf_rfeR2L=DecisionTreeClassifier(random_state=0)
clf_rfeU2R=DecisionTreeClassifier(random_state=0)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS)
clf_rfeProbe.fit(X_rfeProbe, Y_Probe)
clf_rfeR2L.fit(X_rfeR2L, Y_R2L)
clf_rfeU2R.fit(X_rfeU2R, Y_U2R)
```

```
DecisionTreeClassifier(random_state=0)
```

Results and Analysis

The system was then deliberately attacked by the known attacks given in the test set and the ability of the IDS to predict the attack was noted to find it's accuracy

Prediction for Dos:-

Now we have input the code to find out about Dos attack

Input code:-

```
Y_DoS_pred=clf_DoS.predict(X_DoS_test)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Output:-

Predicted attacks	0	1
Actual attacks		
0	9499	212
1	2830	4630

Here the columns represent UDP (0) and TCP (1) attacks which were predicted. The rows represent whether the predicted attacks were actual attacks (1) or false flags (0).

Prediction for Probe:-

Now we have input the code to find out about Probe attack

Input code:-

Probe

```
Y_Probe_pred=clf_Probe.predict(X_Probe_test)
# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Output:

Predicted attacks	0	2
Actual attacks		
0	2337	7374
2	212	2209

Here the columns represent UDP (0) and TCP (2) attacks which were predicted. The rows represent whether the predicted attacks were actual attacks (2) or false flags (0).

Prediction for R2L:-

Now we have input the code to find out about R2L attack

Input code:-

R2L

```
[53] Y_R2L_pred=clf_R2L.predict(X_R2L_test)
     # Create confusion matrix
     pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Output:-

Predicted attacks	0	3
Actual attacks		
0	9707	4
3	2573	312

Here the columns represent UDP (0) and TCP (3) attacks which were predicted. The rows represent whether the predicted attacks were actual attacks (3) or false flags (0).

Prediction for U2R:-

Now we have input the code to find out about R2L attack

Input code:-

U2R

```
Y_U2R_pred=clf_U2R.predict(X_U2R_test)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Output:-

Predicted attacks	0	4
Actual attacks		
0	9703	8
4	60	7

Here the columns represent UDP (0) and TCP (4) attacks which were predicted. The rows represent whether the predicted attacks were actual attacks (4) or false flags (0).

Performing an accuracy for the system:

Cross Validation: Accuracy, Precision, Recall, F-measure

To make the algorithm fast the program, the system first flags the suspicious cases then checks these cases thoroughly instead of checking each case. Precision here denotes the ratio of the cases that were flagged to the cases that turned out to be true attacks.

Here recall means the speed per step.

f - measure just talks about the ratio between precision and recall.

Prediction for Dos: -

Input code:-

DoS

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Output:-

```
Accuracy: 0.99639 (+/- 0.00341)
Precision: 0.99505 (+/- 0.00477)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99585 (+/- 0.00392)
```

Here we can see the accuracy is almost 99.6 percent.

Prediction for Probe -

Input code:-

Probe

```
[56] accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
     print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
     print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Output:-

```
Accuracy: 0.99571 (+/- 0.00328)
Precision: 0.99392 (+/- 0.00684)
Recall: 0.99267 (+/- 0.00405)
F-measure: 0.99329 (+/- 0.00512)
```

Here we can see the accuracy is almost 99.6 percent.

Prediction for R2L:-

Input code:-

R2L

```
accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Output:-

```
Accuracy: 0.97920 (+/- 0.01053)
Precision: 0.97151 (+/- 0.01736)
Recall: 0.96958 (+/- 0.01379)
F-measure: 0.97051 (+/- 0.01478)
```

Here we can see the accuracy is almost 98 percent.

Prediction for U2R:-

Input code:-

U2R

```
accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Output:-

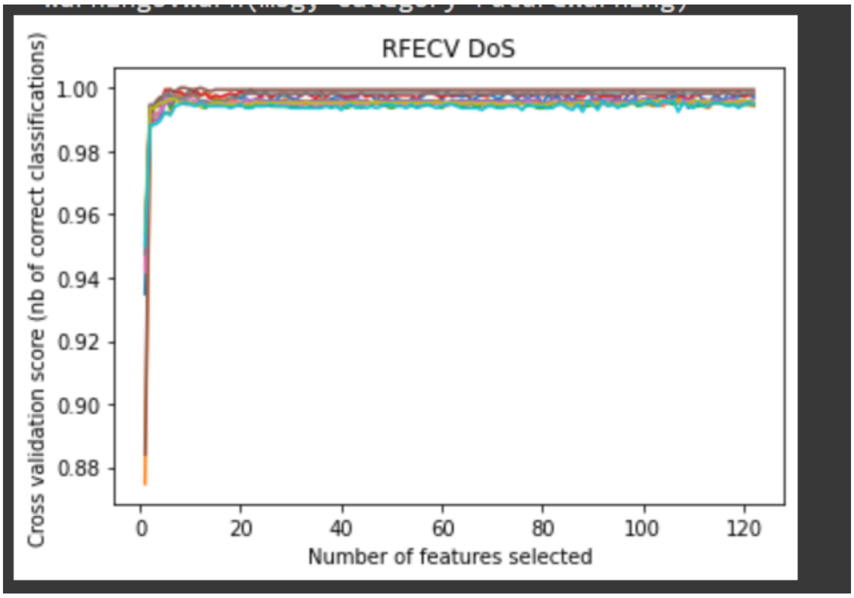
```
Accuracy: 0.99652 (+/- 0.00228)
Precision: 0.86295 (+/- 0.08961)
Recall: 0.90958 (+/- 0.09211)
F-measure: 0.88210 (+/- 0.06559)
```

Here we can see the accuracy is almost 99.7 percent.

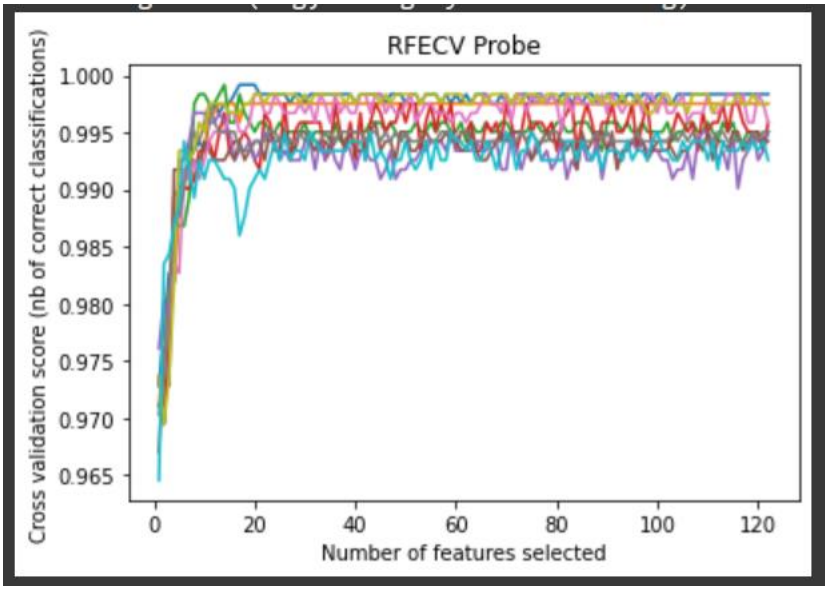
Graphs:

The graphs below section denote the validation score and accuracy of different attacks. This validation is done by our system when our system was attacked using the data from test set:-

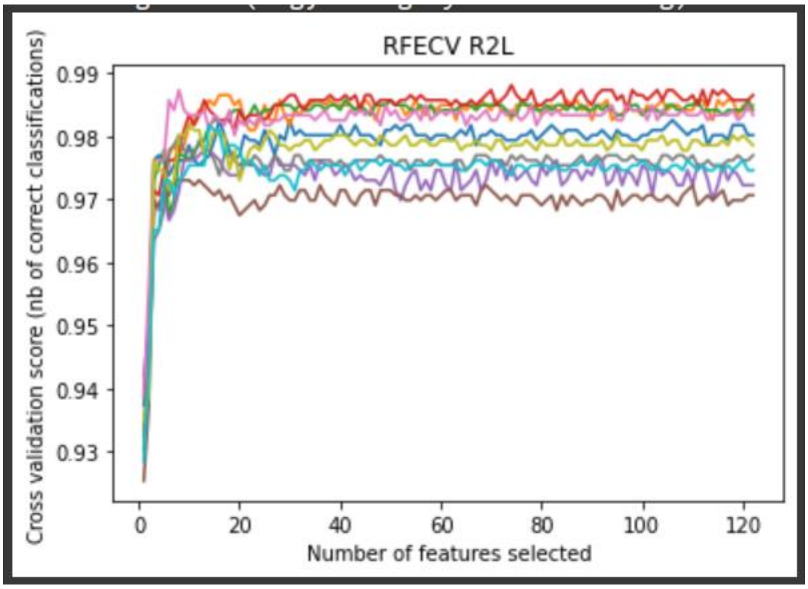
For DOS: -



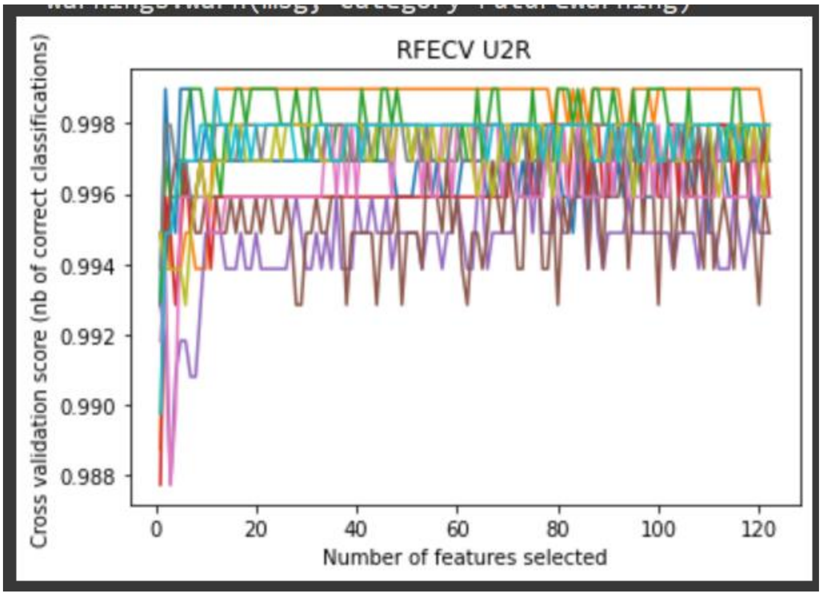
For Probe:-



For R2L:-



For U2R: -


RFECV U2R
Cross validation score (nb of correct classifications)
0.998
0.996
0.994
0.992
0.990
0.988
0
20
40
60
80
100
120
Number of features selected

Using same dataset with different algorithms:

Gausian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb_Dos = GaussianNB()
gnb_Probe = GaussianNB()
gnb_R2l = GaussianNB()
gnb_U2r = GaussianNB()
gnb_Dos.fit(X_DoS, Y_DoS)
gnb_Probe.fit(X_Probe, Y_Probe)
gnb_R2l.fit(X_R2L, Y_R2L)
gnb_U2r.fit(X_U2R, Y_U2R)
```

```
GaussianNB()
```

Dos

```
[ ] Y_DoS_pred=gnb_Dos.predict(X_DoS_test)
    # Create confusion matrix
    pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	9447	264
1	3701	3759

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn import metrics
    accuracy = cross_val_score(gnb_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(gnb_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(gnb_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(gnb_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.86733 (+/- 0.01474)
Precision: 0.98822 (+/- 0.01158)
Recall: 0.70308 (+/- 0.03682)
F-measure: 0.82145 (+/- 0.02371)
```

KNN

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    neigh_DoS = KNeighborsClassifier(n_neighbors=3)
    neigh_Probe = KNeighborsClassifier(n_neighbors=3)
    neigh_R2L = KNeighborsClassifier(n_neighbors=3)
    neigh_U2R = KNeighborsClassifier(n_neighbors=3)
    neigh_DoS.fit(X_DoS, Y_DoS)
    neigh_DoS.fit(X_DoS, Y_DoS)
    neigh_Probe.fit(X_Probe, Y_Probe)
    neigh_R2L.fit(X_R2L, Y_R2L)
    neigh_U2R.fit(X_U2R, Y_U2R)
```

```
KNeighborsClassifier(n_neighbors=3)
```

DOS

```
Y_DoS_pred=neigh_DoS.predict(X_DoS_test)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	9424	287
1	1718	5742

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn import metrics
    accuracy = cross_val_score(neigh_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(neigh_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(neigh_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(neigh_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99674 (+/- 0.00313)
Precision: 0.99585 (+/- 0.00455)
Recall: 0.99665 (+/- 0.00384)
F-measure: 0.99625 (+/- 0.00360)
```

Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression
    log_Dos = LogisticRegression(solver='lbfgs', max_iter=1000)
    log_Dos.fit(X_DoS, Y_DoS)
```

```
LogisticRegression(max_iter=1000)
```

```
[ ] Y_probe = log_Dos.predict(X_Probe_test)
    pd.crosstab(Y_Probe_test, Y_probe, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	8372	1339
2	637	1784

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn import metrics
    accuracy = cross_val_score(neigh_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99674 (+/- 0.00313)
Precision: 0.99505 (+/- 0.00477)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99585 (+/- 0.00392)
```

Random Forest Classifier

```
[100] from sklearn.ensemble import RandomForestClassifier
      Ran_Dos = RandomForestClassifier(max_depth=2, random_state=0)
      Ran_Dos.fit(X_DoS, Y_DoS)
```

```
RandomForestClassifier(max_depth=2, random_state=0)
```

```
[111] Y_probe = Ran_Dos.predict(X_Probe_test)
      pd.crosstab(Y_Probe_test, Y_probe, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	9559	152
2	918	1503

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(Ran_Dos, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.92056 (+/- 0.01326)
Precision: 0.99505 (+/- 0.00477)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99585 (+/- 0.00392)
```

+ Code + Text

ACCURACY COMPARISION AMONG DIFFERENT ALGORITMS:

ACCURACY	Decision Tree	Naïve Bayes	KNN
DOS	99.7%	86.7%	99.6%
PROBE	99%	97.8%	99%
R2L	97.4%	93.5%	96.7%
U2R	99.6%	97.2%	99.7%

CONCLUSION

The aim of this project was to demonstrate the benefits of employing machine learning algorithms for the development of an Intrusion Detection System (IDS). The limitations, as found in the previous research conducted for the same, were eliminated to obtain better results with an efficient system. The result was a system with an accuracy of above 95 in predicting Dos, Probe, U2R and R2L attacks.

The dataset used was NSL KDD dataset which is a new and improved dataset which included a substantial number of U2R and R2L attacks as compared to the old KDD dataset used in much research. The machine learning algorithm employed is also lightweight compared to many heavy computational cost algorithms. The data taken from NSL KDD dataset was preprocessed to remove redundancy and unnecessary features to provide a high accuracy model with low computational cost. Pre-processing of data also provided better results in identifying U2R and R2L attacks which were lacking in the previous research.

References

1. Vinayakumar, R., K. P. Soman, and Prabaharan Poornachandran. "Applying convolutional neural network for network intrusion detection." *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017.
2. Vinayakumar, R., K. P. Soman, and Prabaharan Poornachandran. "A Comparative Analysis of Deep Learning Approaches for Network Intrusion Detection Systems (N-IDSs): Deep Learning for N-IDSs." *International Journal of Digital Crime and Forensics (IJDCF)* 11.3 (2019): 65-89.

3. Vinayakumar, R., et al. "Deep learning approach for intelligent intrusion detection system." *IEEE Access* 7 (2019): 41525-41550.

4. Vinayakumar, R., K. P. Soman, and Prabaharan Poornachandran. "Evaluating effectiveness of shallow and deep networks to intrusion detection system." *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017.
5. Nkiama, Herve, Syed Zainudeen Mohd Said, and Muhammad Saidu. "A subset feature elimination mechanism for intrusion detection system." *International Journal of Advanced Computer Science and Applications* 7.4 (2016): 148-157.
6. Al-Jarrah, O. Y., et al. "Machine-learning-based feature selection techniques for large-scale network intrusion detection." *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2014.
7. Kaur, Rajveer, Gulshan Kumar, and Krishan Kumar. "A comparative study of feature selection techniques for intrusion detection." *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2015.
8. Alazab, A. et al.," Using feature selection for intrusion detection system," 2012 International Symposium on Communications and Information Technologies (ISCIT), pp.296–301. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=638091 0.