**NETWORK INTRUSION DETECTION SYSTEM USING SINGLE LEVEL MULTI-MODEL DECISION TREES**

**A Report submitted in partial fulfillment of the requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

**MALINA MEHER SATYA SWAROOP, 1210316426**

**V.R. KAUSHIK, 1210316458**

**J. RAHUL VARMA, 1210316419**

**B. ROHIT, 1210316405**

Under the esteemed guidance of

**P.V. NAGESWARA RAO, Professor, GIT**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GITAM**

**(Deemed to be University)**

**VISAKHAPATNAM**

**MARCH, 2020**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM INSTITUTE OF TECHNOLOGY**

**GITAM**

**(Deemed to be University)**



We, hereby declare that the project entitled “NETWORK INTRUSION DETECTION SYSTEM” is an original work done by M.SWAROOP(1210316426),V.R.KAUSHIK(1210316458),B.ROHIT(1210316405), J.RAHUL VARMA(1210316419) in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering.

The work has not been submitted to any other college or university for the award of any degree or diploma.

**Project Guide: Prof. P.V. Nageswara Rao, GIT**

**Signature:**

**ACKNOWLEDGEMENT**

We would like to extend our gratitude to our guide and mentor, Prof. **P.V. NAGESWARA RAO** for his continued support and advice for the betterment of the results. We would also like to thank our AMC, **Dr. G. Apparao** for his words of help and guidance. Our sincere thanks to the Head of the Department, **Prof. K. Thammi Reddy** for his counsel to students, constantly encouraging us to pursue the best results.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| ABSTRACT | 5 |
| PROBLEM STATEMENT | 6 |
| INTRODUCTION | 7 |
| DATASET | 9 |
| METHODOLOGY  DECISION TREE CLASSIFIER | 12  15 |
| IMPLEMENTATION | 18 |
| CODE | 18 |
| EVALUATION AND RESULTS | 29 |
| FLOWCHART | 36 |
| TOOLS AND TECHNOLOGIES | 37 |
| CONCLUSION | 41 |
| REFERENCES | 42 |
|  |  |

1. **ABSTRACT**

Intrusion detection is one of the key interests in network administration and security. There is a need to protect the networks from known vulnerabilities and at the same time take steps to identify new and unknown but potential device abusses by creating more robust and effective systems for intrusion detection.

An intrusion detection system (IDS) audits the traffic flowing in the network for suspicious activity and signals when any malicious activity is discovered. For most Intrusion Detection Systems discovering anomalies and generating reports for the intrusions discovered are the elementary responsibilities. In some cases, if a malicious activity or intermittent traffic is detected in the network, intrusion detection systems are programmed to take actions like detering traffic sent from incredulous IP addresses. An IDS is a software that is developed to detect intrusions within the network. It is employed to hunt and pinpoint the intruders within the network.

NETWORK-BASED IDS: The data fed into this type of IDS is obtained from the flow of packets in the network having data.

MISUSE DETECTION: This type of IDS has the functionality to trace previously recognized patterns of pernicious activity going on in the network and spot intrusions.

The IDS we will be developing is a network-based IDS that is programmed to detect any misuse of the network resources (misuse detection) i.e. it detects malicious packets flowing in a network.

We will be building a classifier using Decision Trees. They help in increasing the accuracy at which intrusions are detected. Before building the classifier we will be required to select the most optimal features using Feature selection. We will be employing the concept of Recursive Feature Elimination(RFE). Because of RFE the attacks are detected more efficiently even in highly congested networks. It also leads to a lesser number of false positives and therefore a lower rate of false alarms. Also, The time taken to detect attacks can be cut down significantly by using databases for storage and the concept of Dynamic multi-boosting. The proposed network intrusion detection system can classify packets in real-time based on the packets collected from the network flow.

1. **PROBLEM STATEMENT**

Lately, intrusion detection has become a major concern in the field of network security and administration. Considering intrusion as a security threat, a network needs a system which protects it from known vulnerabilities and at the same time take measures to detect unknown ones for efficient functioning of the network. The detection system must be accurate upto some extent in detecting attacks with a possible minimum number of false positives.

1. **INTRODUCTION**

Intrusion is considered to compromise the integrity, confidentiality, or availability of valuable assets on the computer systems. An intrusion detection system (IDS) audits the traffic flowing in the network for suspicious activity. It then alerts the system administrator when any malicious activity is discovered in the network. The primary functions of intrusion detection systems are discovering anomalies and producing detailed reports for the intrusions discovered . In some cases, if malicious activity or intermittent traffic is detected in the network, intrusion detection systems are programmed to take actions like detering traffic sent from incredulous IP addresses. An IDS is a programmable software that is developed to detect intrusions within the network. It is employed to hunt and pinpoint the intruders causing chaos within the network. The main principles of IDS are integrity, availability, confidentiality and accountability. An IDS is built using both software and hardware. It can detect highly dangerous intrusions within the network. The main purpose of IDS is to detect unauthorized packets and malicious communications that happen in computer systems and networks. The most vital ingredient for the success of intrusion detection systems is feature selection. We can classify the Intrusion Prevention Systems into 4 kinds:

a**. Network Based Intrusion Prevention System** – It detects and identifies any suspicious or unwanted traffic flowing within the network by evaluating the activity of various protocols for the network.

b. **Wireless Intrusion Prevention System** – It detects and identifies any suspicious or unwanted traffic flowing within the network by evaluating the activity of the wireless protocols within the network.

c**. Network Behaviour Analysis** – It evaluates the entire network to spot for vulnerable threats. For example, Remote Code Executions, Denial of Service.

d**. Host Based Intrusion Prevention System** – In this type of system a software is installed on the host to monitor for any pernicious activity. This is done by evaluating the host system.

Intrusions are considered as a sequence of steps taken to compromise the integrity, confidentiality, or availability of valuable assets on the computer systems. Intruders gain unauthorized access to the resources available on the system. They use all kinds of techniques to gain access to confidential information and manipulate the data available on the system. This can sometimes damage the system and render it worthless. An IDS can be considered to be a blend of software and hardware units that can be used to identify and pinpoint unauthorized experiments to gain access to the network. All the network related activities can be audited by an IDS which in turn can be used to suspect the traces of invasions within the network. The end goal of an IDS is to trigger alerts when a suspicious activity has occurred by notifying the System Administrator. Intrusion detection techniques can be classified into 2 types:

A) **Anomaly Detection**: In this kind of detection the system alerts malicious tasks by identifying deviations i.e how differently are the network activities occurring as compared to regular patterns.

B) **Misuse Detection**: In this kind of system intrusions are detected on the basis of already known patterns i.e. previously occurred malicious activity. This method can be used to identify and pinpoint known attack patterns more accurately. Misuse Detection has a disadvantage over Anomaly Detection i.e. only those kinds of intrusions can be identified which can correlate to already known patterns of attack. An ideal IDS will monitor all the happenings within the network and then decide whether those happenings are malicious or normal. The decision is based on system availability, confidentiality and integrity of the information resources. The various aspects to be considered while building and IDS are the following: data pre-processing, collection of data, detecting intrusions and curating reports with appropriate responses.

An Intrusion Detection System works in the following manner: Collecting Data, Selecting Features, Analysing the Data, and the Actions to be Performed.

a. **Collecting Data**: We need to gather reports on the traffic flowing in the network like hosts alive, protocols used and the various forms of traffic flowing.

b. **Selecting Features**: After collecting a huge amount of data, the next step is to pick all those required features which we want to work upon.

c. **Analysing the Data**: In this step the data about the features which are selected data is evaluated to help us determine if the data is unnatural or not.

d. **Actions to be Performed**: When a malicious attack has taken place the system administrator is alarmed or notified by the IDS. The details about the type of attack are also provided by the IDS. The IDS closes the unnecessary network ports and processes to further mitigate the attacks from happening.

1. **DATASET**

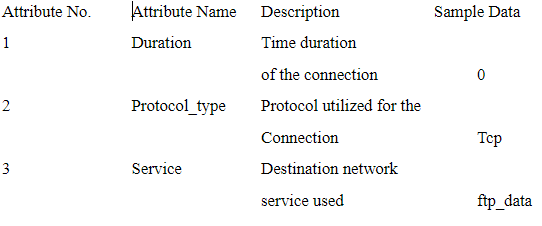
An accurate intrusion detection system requires a good set of labeled examples to sharply detect intrusions in the real world. In order to achieve higher accuracies while modeling an intrusion detection system, a dataset that represents the real world more accurately can only be used. The NSL-KDD data set is a polished version of its forerunner KDD 99 data set.

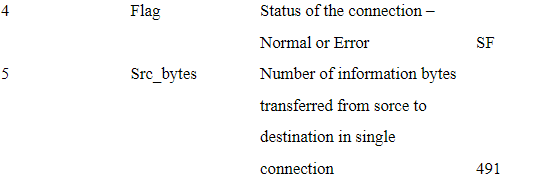
NSL-KDD data set is employed to check the validity and performance of the different classification algorithms in classifying the attacks in the traffic patterns of the network.

Description:

It has all the basic records of the complete KDD data set required for various models. There are a bunch of files to download on the internet for ML practitioners and researchers.

Each record has a total of 41 attributes for identifying the various features of the network packets and a label is named to every packet. This label is used to classify that record either as an attack category or as none/normal. The fundamental points of the features are their name, their description, and sample values. The following table contains information about all the 41 attributes present within the NSL-KDD data set. The 42nd attribute unfolds data about labels of each record which represents the packets in the network and that they’re categorized as 1 ordinary class and 4 attack-type classes. The 4 attack classes used to classify the records in the dataset are DoS, Probe, R2L, and U2R. The mentioned table is as follows :





**The advantages in adopting the NSL-KDD dataset are**

1. There won't be any similar records that have a stronger rate of decline in the test sample.
2. As compared to the KDD-99, the NSL-KDD shows a lesser number of data points. Therefore using them for training machine learning models is less costly algorithmically.
3. The number of records chosen for each tough group of rates is inversely proportional to the percentage of records in the original KDD dataset.

**NSL-KDD train and test data distribution**

The 4 attack categories available within the NSL-KDD data set are:

1. **DOS**: This kind of attack leads to draining of the victims’s resources and making it incapable in responding to legitimate requests. This is one of the 4 attack categories.

Example: syn flooding. The suitable features from the dataset for this attack class are: “serror\_rate” and “flag\_SF”.

2. **U2R**(unauthorized access to local root privileges): In this kind of attack, an attacker tries to obtain root/administrator privileges by taking advantage of some vulnerability within the victim’s system. The attacker usually uses a traditional account to login into a victim’s system. The suitable attributes from the dataset for this attack class are: “root\_shell”, “service\_http”, and “dst\_host\_same\_src\_port\_rate”.

3. **Probing**:This kind of attack involves obtaining sensitive information present in the victim’s computer/device. The suitable attributes for this attack class are: “Protocol\_type\_icmp” and “dst\_host\_same\_src\_port\_rate”.

4. **R2L**: This kind of attack involves unapproved access of the victim's device by gaining root access where he/she can view the data within that device with root privileges and all this is done from a far off(remote) machine by the attacker.

E.g. password bruteforce attack. The suitable features from the dataset for this attack class are: “dst\_bytes”, “dst\_host\_srv\_diff\_host\_rate”, and “dst\_host\_same\_src\_port\_rate”.

1. **METHODOLOGY**

**PROPOSED IDS METHODOLOGY FOR THE ML MODEL:**

The primary goal is to design a plan for detecting intrusions within the system with the least possible number of features within the dataset. Based on the data from previous papers published, we can tell that only a subdivision of features in the dataset are derivative to the Intrusion Detection System. We have to cut back the dimensionality of the dataset to build an improved classifier in a justifiable amount of time. The approach we are going to use has a total of 4 stages: In the first stage, we pick out the significant features for every class using feature selection.In the next we combine the various features, so that the final cluster of features are optimal and relevant for each attack class. The third stage is for building a classifier. Here, the optimal features found in the previous stage are sent as input into the classifier. In the last stage, we test the model by employing a test dataset.

The proposed approach consists of 4 fundamental stages:

1. Feature selection
2. Combining the optimal features
3. Building a classifier
4. Evaluation

**FEATURE SELECTION**:

We already know that network intrusion systems require bulk amounts of raw data to be dealt with. The most crucial building block of developing such a sophisticated system is Feature Selection. One thing which has a major effect on the dimensionality of the dataset is the complexity of the model. It results in obtaining high computational time and costs and also low classification accuracies. A significant number of techniques and approaches are used to get rid of repetitive and insignificant features.

These methods are chosen in such a way that we select only those features which are optimal and can be used to enhance the performance of the model. The 2 basic methods used for feature selection are: wrapper methods and filter methods.

**Wrapper algorithms** - In these types of algorithms a learning algorithm is used to make a decision for the features.

**Filter algorithms -** In these types of algorithms a measure which is not dependent on the others is employed. Some of the measures are consistency, information and distance. A relationship between the set of features can be estimated using these measures.

We will be using Information Gain (IG) to select the subset of relevant features in this project. Information Gain often costs less and is faster as compared to the wrapper methods. We calculate Information gain for all the attributes present in the training dataset. It is calculated for each class separately. In the next step the values of the information gain are ranked i.e. the feature with the highest information gain being at rank 1. It means that this particular feature can distinctively classify for the particular class. If the value of the Information gain is less than the fixed threshold value for a particular feature, that feature can be eliminated from the feature space. A better and optimal threshold value is obtained by examining the distribution of Information Gain for each attribute tested with different values of fixed threshold on the training dataset.

In the next stage the feature selection for each category can be broken down into 4 stages:

**Stage 1**: We divide the training dataset into 4 datasets. The training dataset is divided into 4 datasets in such a way that each dataset consists of records belonging to the same attack class along with some of the records of the original dataset. This stage is performed so that the feature selection method is unbiased while selecting features for frequently occurring attacks in the dataset. This step helps us in preventing unwanted bias and to make sure the results retrieved are accurate.

**Stage 2**: In this stage the datasets for each attack class are sent separately as input into the method used to calculate the information gain. The output of this method gives us the most significant features for each attack type.

**Stage 3**: In the third stage we generate a list of ranked features for each attack class. Now we eliminate all the irrelevant features from the list in accordance with the fixed threshold values.

**COMBINING OPTIMAL FEATURES:**

In the last stage of feature selection, we combine the list of features generated for each attack into a single list . For some of the attack classes the highest ranks i.e., the top 4 features chosen for classification. But for some types of attack classes, we can only take 1 feature since that particular feature is at the top of the rank table and the remaining features are at the very bottom of the table. So, the final set of combined optimal features can be used to entirely distinguish the attack types.

**BUILDING A CLASSIFIER:**

The obtained subset of relevant features of each attack type in the above step is now used to build a classifier in this phase. As the model that is going to be used is a decision tree classifier, the dataset containing only the obtained subset of features is used to build 4 classifiers i.e., 1 for each attack type. This creates trees with only the relevant features of each attack type in each decision tree.

**MODEL USED:**

**DECISION TREE CLASSIFIER**:

A Decision tree is an algorithm which takes decisions at each node of the tree and is widely used for regression and classification. It is a supervised learning algorithm in Machine learning where which attribute should be at which node is learnt by using a set of labeled examples. The main advantage in using decision trees is that they can be trained very easily and they can even classify non linear data. It is more productive than most of the classification algorithms in ML like K-Nearest Neighbours in most of the cases. The common measures used to select attributes at each node in Decision trees are Info gain and Gain ratio.

1. **Entropy**: A measure of impurity which is defined as the degree or the amount of uncertainty in the randomness of elements

E(S) = Σ−𝑃𝐼𝑐𝑖=1𝑙𝑜𝑔2𝑃𝑖

Homogeneity of a sample is calculated by the Entropy.

* Entropy = 0, if the sample is absolutely homogeneous
* Entropy = 1, if the sample is uniformly divided.

2. **Information Gain**: The relative variation in entropy can be measured using Information Gain which corresponds to an independent attribute.While constructing a Decision tree, an attribute at a node is selected by calculating the information gain of all attributes and determining the attribute which has the maximum information gain. Info gain calculates the information encompassed by each feature.

Gain (D, Attr.) = Entropy(D) – Entropy (D, Attr.)

Where Gain (D, Attr.) is the information gain of the attribute Attr. Entropy(D) is the entropy of the complete set, and after implementing the attributes entropy is calculated using Entropy(D, Attr). Overfitting is one of the main problems of the decision tree as it constructs its nodes by thoroughly going through the entire training set which may result in a low accuracy while using unknown data like the test set.

**Decision Tree is implemented as follows**:

Decision Tree works in the following way:

a.The best attribute is identified for separating the records using the Attribute Selection Steps (ASM).

b. Now the dataset is split into smaller subsets by using the previously selected attribute as the decision node.

c. A tree is built by the algorithm by iteratively executing this cycle before any one condition fits in :

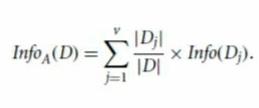
1. There are no attributes left over anymore.

2. No more instances do exist

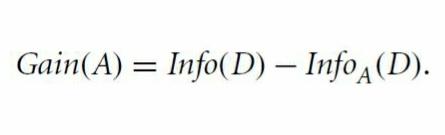
3. The Tuples generated have the same value of the attribute .

.

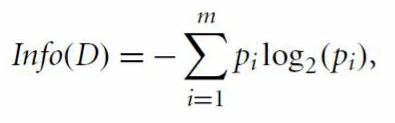
**Entropy of attribute**:



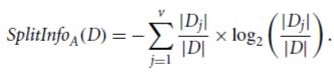
**Information gain**:



**Entropy**:



**Split info:**

****

**6. IMPLEMENTATION**

**CODE:**

**import pandas as pd**

**import numpy as np**

**import sys**

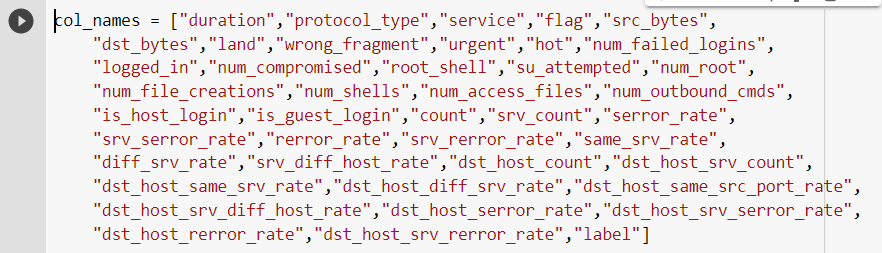
**import sklearn**

**from IPython.display import Image**

**from sklearn import tree**

**import pydotplus**

**DATA PREPROCESSING:**



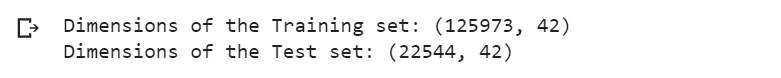
**df = pd.read\_csv("NIDS/KDDTrain+\_2.csv", header=None, names = col\_names)**

**df\_test = pd.read\_csv("NIDS/Network-Intrusion-Detection/KDDTest+\_2.csv", header=None, names = col\_names)**

**#this gives the dimensions of the dataset using .shape**

**print('Dimensions of the Training set are:',df.shape)**

**print('Dimensions of the Test set are:',df\_test.shape)**

****

**Identify categorical features:**

**print('Training set:')**

**for cname in df.columns:**

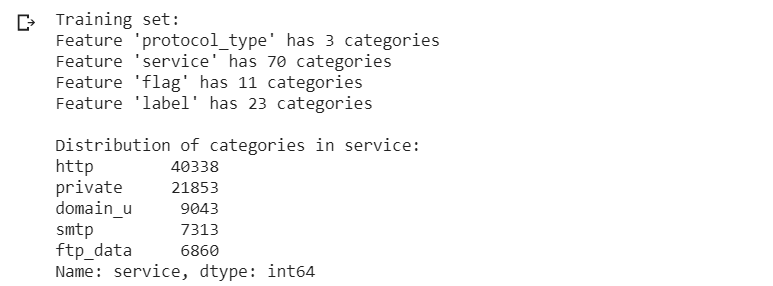
**if (df[cname].dtypes == 'object') :**

**u\_categ = len(df[cname].unique())**

**print("Attribute: '{cname}' has {u\_cat} categories".format(cname=cname, u\_cat=u\_categ))**

**print('Distribution of categories in service column:')**

**print(df['service'].value\_counts().sort\_values(ascending=False).head())**

****

**Make column names for dummies:**

**from sklearn.preprocessing import LabelEncoder,OneHotEncoder**

**# code to get a list of categorical columns into a variable, categ\_cols**

**categ\_cols=['protocol\_type', 'service', 'flag']**

**df\_categ\_vals = df[categ\_cols]**

**testdf\_categ\_vals = df\_test[categ\_cols]**

**# protocol type col**

**u\_protocol = sorted(df.protocol\_type.unique())**

**s\_1 = 'Protocol\_type\_'**

**u\_protocol\_2 = [s\_1 + str for str in u\_protocol]**

**# service col**

**u\_service = sorted(df.service.unique())**

**s\_2 = 'service\_'**

**u\_service\_2 = [s\_2 + str for str in u\_service]**

**# flag col**

**u\_flag = sorted(df.flag.unique())**

**s\_3 = 'flag\_'**

**u\_flag\_2=[s\_3 + atr for str in u\_flag]**

**dum\_cols=u\_protocol\_2 + u\_service\_2 + u\_flag\_2**

**print(dum\_cols)**

**# for test set**

**u\_service\_test = sorted(df\_test.service.unique())**

**u\_service\_2\_test=[s\_2 + str for str in u\_service\_test]**

**testdum\_cols=u\_protocol\_2 + u\_service\_2\_test + u\_flag\_2**

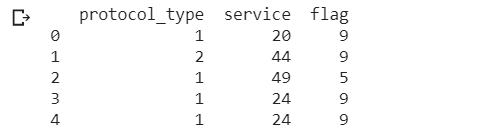
**Transform categorical features into numbers using LabelEncoder():**

**df\_categ\_vals\_enc=df\_categ\_vals.apply(LabelEncoder().fit\_transform)**

**print(df\_categ\_vals\_enc.head())**

**# test set**

**testdf\_categ\_vals\_enc=testdf\_categ\_vals.apply(LabelEncoder().fit\_transform)**

****

# One-Hot-Encoding:

**enc = OneHotEncoder()**

**df\_categ\_vals\_encenc = enc.fit\_transform(df\_categ\_vals\_enc)**

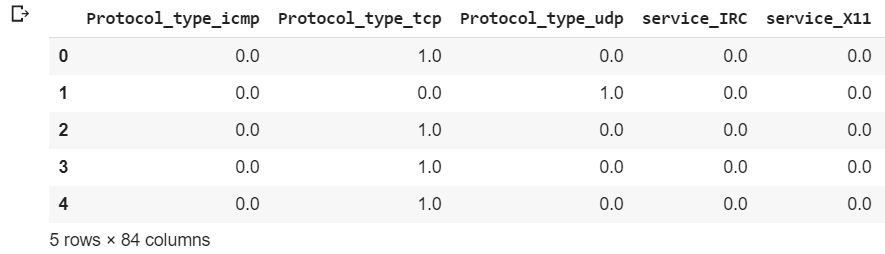
**df\_categ = pd.DataFrame(df\_categ\_vals\_encenc.toarray(),columns=dum\_cols)**

**# for test set**

**testdf\_categ\_vals\_encenc = enc.fit\_transform(testdf\_categ\_vals\_enc)**

**testdf\_categ = pd.DataFrame(testdf\_categ\_vals\_encenc.toarray(),columns=testdum\_cols)**

**df\_categ.head()**

****

**Join encoded categorical dataframe with the non-categorical dataframe:**

**new\_dataset=df.join(df\_categ)**

**new\_dataset.drop('flag', axis=1, inplace=True)**

**new\_dataset.drop('protocol\_type', axis=1, inplace=True)**

**new\_dataset.drop('service', axis=1, inplace=True)**

**# test data**

**new\_dataset\_test=df\_test.join(testdf\_categ)**

**new\_dataset\_test.drop('flag', axis=1, inplace=True)**

**new\_dataset\_test.drop('protocol\_type', axis=1, inplace=True)**

**new\_dataset\_test.drop('service', axis=1, inplace=True)**

**print(new\_dataset.shape)**

**print(new\_dataset\_test.shape)**

****

**Split Dataset into 4 datasets for every attack category:**

****

****

**# put the new label column back**

**new\_dataset['label'] = new\_label\_2df**

**new\_dataset\_test['label'] = new\_label\_2df\_test**

**print(new\_dataset['label'].head())**

**elm\_DoS = [2,3,4]**

**elm\_Probe = [1,3,4]**

**elm\_R2L = [1,2,4]**

**elm\_U2R = [1,2,3]**

**DoS\_dataset=new\_dataset[~new\_dataset['label'].isin(elm\_DoS)];**

**Probe\_dataset=new\_dataset[~new\_dataset['label'].isin(elm\_Probe)];**

**R2L\_dataset=new\_dataset[~new\_dataset['label'].isin(elm\_R2L)];**

**U2R\_dataset=new\_dataset[~new\_dataset['label'].isin(elm\_U2R)];**

**#for test set**

**DoS\_dataset\_test = new\_dataset\_test[~new\_dataset\_test['label'].isin(elm\_DoS)];**

**Probe\_dataset\_test = new\_dataset\_test[~new\_dataset\_test['label'].isin(elm\_Probe)];**

**R2L\_dataset\_test = new\_dataset\_test[~new\_dataset\_test['label'].isin(elm\_R2L)];**

**U2R\_dataset\_test = new\_dataset\_test[~new\_dataset\_test['label'].isin(elm\_U2R)];**

**print('Training set:')**

**print('Dimensions of DoS:' ,DoS\_dataset.shape)**

**print('Dimensions of Probe:' ,Probe\_dataset.shape)**

**print('Dimensions of R2L:' ,R2L\_dataset.shape)**

**print('Dimensions of U2R:' ,U2R\_dataset.shape)**

**print('Test set:')**

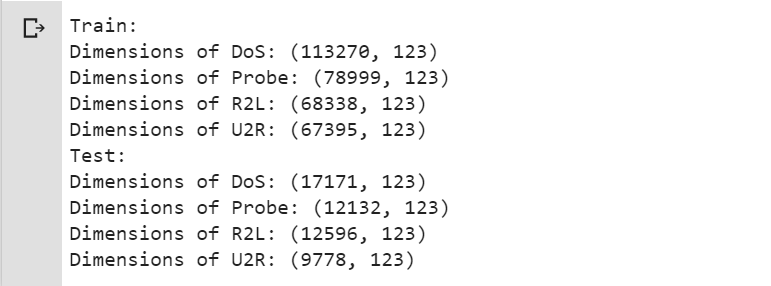
**print('Dimensions of DoS:' ,DoS\_dataset\_test.shape)**

**print('Dimensions of Probe:' ,Probe\_dataset\_test.shape)**

**print('Dimensions of R2L:' ,R2L\_dataset\_test.shape)**

**print('Dimensions of U2R:' ,U2R\_dataset\_test.shape)**

**Output:**

****

**Feature scaling:**

**# divide the dataset into features & class**

**X\_DoS = DoS\_dataset.drop('label',1)**

**Y\_DoS = DoS\_dataset.label**

**X\_Probe = Probe\_dataset.drop('label',1)**

**Y\_Probe = Probe\_dataset.label**

**X\_R2L = R2L\_dataset.drop('label',1)**

**Y\_R2L = R2L\_dataset.label**

**X\_U2R = U2R\_dataset.drop('label',1)**

**Y\_U2R = U2R\_dataset.label**

**# test set**

**X\_DoS\_test = DoS\_dataset\_test.drop('label',1)**

**Y\_DoS\_test = DoS\_dataset\_test.label**

**X\_Probe\_test = Probe\_dataset\_test.drop('label',1)**

**Y\_Probe\_test = Probe\_dataset\_test.label**

**X\_R2L\_test = R2L\_dataset\_test.drop('label',1)**

**Y\_R2L\_test = R2L\_dataset\_test.label**

**X\_U2R\_test = U2R\_dataset\_test.drop('label',1)**

**Y\_U2R\_test = U2R\_dataset\_test.label**

**col\_Names=list(X\_DoS)**

**col\_Names\_test=list(X\_DoS\_test)**

**from sklearn import preprocessing**

**sclr\_1 = preprocessing.StandardScaler().fit(X\_DoS)**

**X\_DoS=sclr\_1.transform(X\_DoS)**

**sclr\_2 = preprocessing.StandardScaler().fit(X\_Probe)**

**X\_Probe=sclr\_2.transform(X\_Probe)**

**sclr\_3 = preprocessing.StandardScaler().fit(X\_R2L)**

**X\_R2L=sclr\_3.transform(X\_R2L)**

**sclr\_4 = preprocessing.StandardScaler().fit(X\_U2R)**

**X\_U2R=sclr\_4.transform(X\_U2R)**

**# test data**

**sclr\_5 = preprocessing.StandardScaler().fit(X\_DoS\_test)**

**X\_DoS\_test = sclr\_5.transform(X\_DoS\_test)**

**sclr\_6 = preprocessing.StandardScaler().fit(X\_Probe\_test)**

**X\_Probe\_test = sclr\_6.transform(X\_Probe\_test)**

**sclr\_7 = preprocessing.StandardScaler().fit(X\_R2L\_test)**

**X\_R2L\_test = sclr\_7.transform(X\_R2L\_test)**

**sclr\_8 = preprocessing.StandardScaler().fit(X\_U2R\_test)**

**X\_U2R\_test = sclr\_8.transform(X\_U2R\_test)**

# 

# Feature Selection:

**from sklearn.feature\_selection import RFE**

**from sklearn.tree import DecisionTreeClassifier**

**classif = DecisionTreeClassifier(random\_state=0)**

**feature\_sel = RFE(estimator=classif, n\_features\_to\_select=13, step=1)**

**Y\_DoS = Y\_DoS.astype('int')**

**feature\_sel.fit(X\_DoS, Y\_DoS)**

**X\_DoS\_rfe = feature\_sel.transform(X\_DoS)**

**true = feature\_sel.support\_**

**sel\_colindexDoS=[i for i, j in enumerate(true) if j]**

**Y\_Probe = Y\_Probe.astype('int')**

**feature\_sel.fit(X\_Probe, Y\_Probe)**

**X\_Probe\_rfe = feature\_sel.transform(X\_Probe)**

**true=feature\_sel.support\_**

**sel\_colindexProbe=[i for i, j in enumerate(true) if j]**

**Y\_R2L = Y\_R2L.astype('int')**

**feature\_sel.fit(X\_R2L, Y\_R2L)**

**X\_R2L\_rfe = feature\_sel.transform(X\_R2L)**

**true = feature\_sel.support\_**

**sel\_colindexR2L=[i for i, j in enumerate(true) if j]**

**Y\_U2R = Y\_U2R.astype('int')**

**feature\_sel.fit(X\_U2R, Y\_U2R)**

**X\_U2R\_rfe = feature\_sel.transform(X\_U2R)**

**true = feature\_sel.support\_**

**sel\_colindexU2R=[i for i, j in enumerate(true) if j]**

**Build the model:**

**classif\_DoS\_rfe = DecisionTreeClassifier(random\_state=0)**

**classif\_Probe\_rfe = DecisionTreeClassifier(random\_state=0)**

**classif\_R2L\_rfe = DecisionTreeClassifier(random\_state=0)**

**classif\_U2R\_rfe = DecisionTreeClassifier(random\_state=0)**

**classif\_DoS\_rfe.fit(X\_DoS\_rfe, Y\_DoS)**

**classif\_Probe\_rfe.fit(X\_Probe\_rfe, Y\_Probe)**

**classif\_R2L\_rfe.fit(X\_R2L\_rfe, Y\_R2L)**

**classif\_U2R\_rfe.fit(X\_U2R\_rfe, Y\_U2R)**

**X\_DoS\_rfe\_test = X\_DoS\_test[:,sel\_colindexDoS]**

**X\_Probe\_rfe\_test = X\_Probe\_test[:,sel\_colindexProbe]**

**X\_R2L\_rfe\_test = X\_R2L\_test[:,sel\_colindexR2L]**

**X\_U2R\_rfe\_test = X\_U2R\_test[:,sel\_colindexU2R]**

**EVALUATION AND RESULTS**

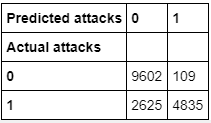
**CONFUSION MATRICES:**

**DoS:**

**DoS\_Y\_predicted = classif\_DoS\_rfe.predict(X\_DoS\_rfe\_test)**

**pd.crosstab(Y\_DoS\_test, DoS\_Y\_predicted, rownames=['Actual attacks'],**

**colnames=['Predicted attacks'])**

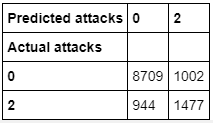
****

**PROBE:**

**Probe\_Y\_predicted=classif\_Probe\_rfe.predict(X\_Probe\_rfe\_test)**

**pd.crosstab(Y\_Probe\_test, Probe\_Y\_predicted, rownames=['Actual attacks'],**

**colnames=['Predicted attacks'])**

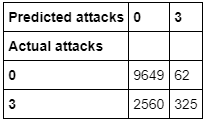
****

**R2L:**

**R2L\_Y\_predicted=classif\_R2L\_rfe.predict(X\_R2L\_rfe\_test)**

**pd.crosstab(Y\_R2L\_test, R2L\_Y\_predicted, rownames=['Actual attacks'],**

**colnames=['Predicted attacks'])**

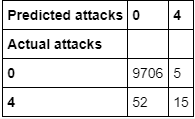
****

**U2R:**

**U2R\_Y\_predicted=classif\_U2R\_rfe.predict(X\_U2R\_rfe\_test)**

**pd.crosstab(Y\_U2R\_test, U2R\_Y\_predicted, rownames=['Actual attacks'],**

**colnames=['Predicted attacks'])**

****

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

**Accuracy**:

The accuracy (AC) is defined as the distribution of the total number of correct predictions. The equation is estimated as:

Accuracy= TP+TN/TP+TN+FP+FN

**Precision**:

Precision is defined as the ratio of the total no. of true positives and the sum of the number of true positives and the number of false positives. It is calculated by the equation:

Precision 1= TP/TP+FP

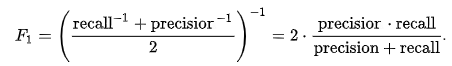
**Recall**:

Recall can be defined as the proportion of the total number of positive examples rightly listed, divided by the total number of positive ones. High Recall suggests correct identification of class. It is also defined in scientific terms as Detection Frequency, True Positive Rate, or Sensitivity. The equation computes as:

Recall = TP/TP+FN

**F-score**:

The F score is also known as F1 score or F measure. It is a measure of the accuracy of a test. The F score is interpreted as the weighted harmonic mean of the test’s precision and recall. This score is calculated using the following equation:



**from sklearn.model\_selection import cross\_val\_score**

**DoS:**

**acc = cross\_val\_score(classif\_DoS\_rfe, X\_DoS\_rfe\_test, Y\_DoS\_test, cv=10, scoring='accuracy')**

**print("Accuracy: %0.5f (+/- %0.5f)" % (acc.mean(), acc.std() \* 2))**

**prec = cross\_val\_score(classif\_DoS\_rfe, X\_DoS\_rfe\_test, Y\_DoS\_test, cv=10, scoring='precision')**

**print("Precision: %0.5f (+/- %0.5f)" % (prec.mean(), prec.std() \* 2))**

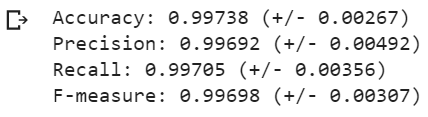
**recall = cross\_val\_score(classif\_DoS\_rfe, X\_DoS\_rfe\_test, Y\_DoS\_test, cv=10, scoring='recall')**

**print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() \* 2))**

**f1 = cross\_val\_score(classif\_DoS\_rfe, X\_DoS\_rfe\_test, Y\_DoS\_test, cv=10, scoring='f1')**

**print("F-measure: %0.5f (+/- %0.5f)" % (f1.mean(), f1.std() \* 2))**

**RESULT:**

****

**Probe:**

**acc = cross\_val\_score(classif\_Probe\_rfe, X\_Probe\_rfe\_test, Y\_Probe\_test, cv=10, scoring='accuracy')**

**print("Accuracy: %0.5f (+/- %0.5f)" % (acc.mean(), acc.std() \* 2))**

**prec = cross\_val\_score(classif\_Probe\_rfe, X\_Probe\_rfe\_test, Y\_Probe\_test, cv=10, scoring='precision\_macro')**

**print("Precision: %0.5f (+/- %0.5f)" % (prec.mean(), prec.std() \* 2))**

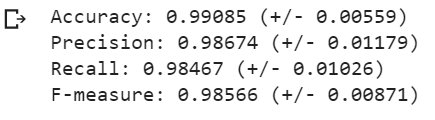
**recall = cross\_val\_score(classif\_Probe\_rfe, X\_Probe\_rfe\_test, Y\_Probe\_test, cv=10, scoring='recall\_macro')**

**print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() \* 2))**

**f1 = cross\_val\_score(classif\_Probe\_rfe, X\_Probe\_rfe\_test, Y\_Probe\_test, cv=10, scoring='f1\_macro')**

**print("F-measure: %0.5f (+/- %0.5f)" % (f1.mean(), f1.std() \* 2))**

**RESULT:**

****

**R2L:**

**acc = cross\_val\_score(classif\_R2L\_rfe, X\_R2L\_rfe\_test, Y\_R2L\_test, cv=10,**

**scoring='accuracy')**

**print("Accuracy: %0.5f (+/- %0.5f)" % (acc.mean(), acc.std() \* 2))**

**prec = cross\_val\_score(classif\_R2L\_rfe, X\_R2L\_rfe\_test, Y\_R2L\_test, cv=10,**

**scoring='precision\_macro')**

**print("Precision: %0.5f (+/- %0.5f)" % (prec.mean(), prec.std() \* 2))**

**recall = cross\_val\_score(classif\_R2L\_rfe, X\_R2L\_rfe\_test, Y\_R2L\_test, cv=10,**

**scoring='recall\_macro')**

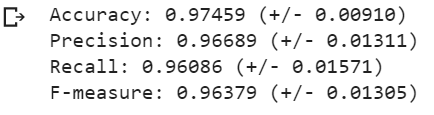
**print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() \* 2))**

**f1 = cross\_val\_score(classif\_R2L\_rfe, X\_R2L\_rfe\_test, Y\_R2L\_test, cv=10,**

**scoring='f1\_macro')**

**print("F-measure: %0.5f (+/- %0.5f)" % (f1.mean(), f1.std() \* 2))**

**RESULT:**

****

**U2R:**

**acc = cross\_val\_score(classif\_U2R\_rfe, X\_U2R\_rfe\_test, Y\_U2R\_test, cv=10,**

**scoring='accuracy')**

**print("Accuracy: %0.5f (+/- %0.5f)" % (acc.mean(), acc.std() \* 2))**

**prec = cross\_val\_score(classif\_U2R\_rfe, X\_U2R\_rfe\_test, Y\_U2R\_test, cv=10,**

**scoring='precision\_macro')**

**print("Precision: %0.5f (+/- %0.5f)" % (prec.mean(), prec.std() \* 2))**

**recall = cross\_val\_score(classif\_U2R\_rfe, X\_U2R\_rfe\_test, Y\_U2R\_test, cv=10, scoring='recall\_macro')**

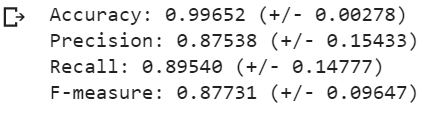
**print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() \* 2))**

**f1 = cross\_val\_score(classif\_U2R\_rfe, X\_U2R\_rfe\_test, Y\_U2R\_test, cv=10,**

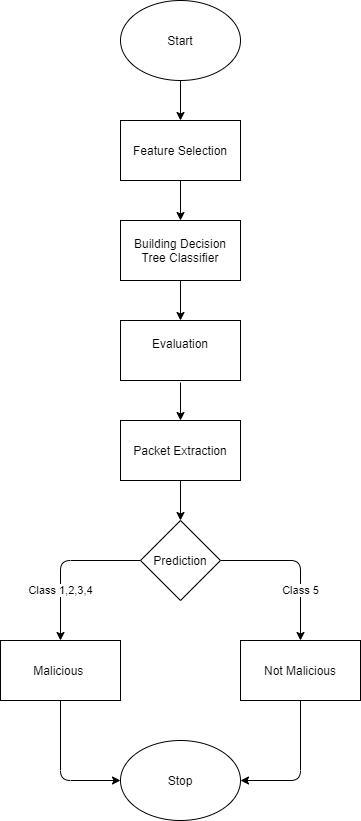
**scoring='f1\_macro')**

**print("F-measure: %0.5f (+/- %0.5f)" % (f1.mean(), f1.std() \* 2))**

**RESULT:**

****

**7. FLOWCHART**



**8. TOOLS**

**scikit-learn**

scikit-learn is a Machine learning library developed with the help of SciPy. It is widely used by Machine learning practitioners.

David Cournapeau started the project in 2007 for GSoC competition and since then many developers have been working on the code which is open sourced.

**Important features of scikit-learn:**

1. It is a plain and productive tool used for data processing software and data mining. Some of the algorithms it includes are: regression, classification, clustering algorithms like k-means, random forests, gradient boosting, support vector machines, etc.
2. Open to all and reusable in different situations.
3. It was built with the foundation of matplotlib, NumPy and SciPy.
4. Commercially available open source-BSD license.

**Installation**

**Dependencies**

scikit-learn requires:

1. Python (>= v3.5)
2. NumPy (>= v1.11.0)
3. SciPy (>= v0.17.0)
4. joblib (>= v0.11)

The last version to support Python 2.7 and Python 3.4 was Scikit-learn 0.20. The later versions of Scikit-learn i.e. Scikit-learn 0.21 and later require Python 3.5 or newer.

Scikit-learn also provides plotting facilities (i.e., functions start with "[plot\_](https://github.com/scikit-learn/scikit-learn#id2)" and classes end with "Display") which need Matplotlib (>= v1.5.1) for running. Some examples may also need scikit-image >= v0.12.3, and some may need pandas >= v0.18.0.

**User installation**

If your system already has the required versions numpy and scipy which is completely working, the simplest method to install scikit-learn will be by using



**PyCharm**

PyCharm is an integrated development environment (IDE) employed in programming and developing daily use softwares, developed for the Python language explicitly. PyCharm is developed by the very popular Czech company- JetBrains. It provides an interface for code analysis, a graphical debugger for debugging the code, integration with version control systems (VCSes), and supports web development using Django. And it also supports Data Science using Anaconda.

PyCharm is a cross-platform software, which makes it compatible with versions of Windows, Linux and MacOS. The software's Community Version comes out under the Apache License. It also provides extra features to Professional Edition-published under a proprietary license that can be used for technical purposes.

**Installing Python**

1. Python can be downloaded from its official website http://www.python.org/downloads/, where you choose a version as per your needs. We choose Python v3.6.3 for our convenience.
2. Python is installed by running the exe file after the completion of the download.
3. Now, you can see a Window where Python is being installed.
4. It will display a message saying : The Setup was Successful. Now press the close button.

**Installing Pycharm**

1. PyCharm can be downloaded from its official website https://www.jetbrains.com/pycharm/download/.Visit the website and click DOWNLOAD"under the Community Section.
2. PyCharm can be installed by running the exe file, once the download is complete.You can click the "Next" button on the Setup Wizard.
3. The Path of installation can be changed on the following screen. Click the "Next" button.
4. You can build a desktop shortcut on the next screen if you wish, and press "Next".
5. Choose the folder in the Start tab. Make sure JetBrains is selected and press "Install".
6. Wait until the setup is complete.
7. When the installation is complete, you will be able to see a screen that shows the message that says PyCharm is installed. Click on "Run PyCharm Community Edition" before clicking on the "Finish" button.

**Numpy**:

NumPy is Python's most commonly used scientific computing program. Among other items, it includes the following:

1. a powerful N-dimensional array object
2. sophisticated (broadcasting) functions
3. tools for integrating C/C++ and Fortran code
4. Fourier transform calculator, functions for linear algebra, and random number facilities

We may use NumPy as a powerful multi-dimensional container of general purpose data in addition to its obvious scientific uses. We can also describe arbitrary data types. This allows NumPy to integrate with a wide variety of databases seamlessly and effortlessly.

NumPy is licensed under the BSD license.This enables reuse of the package with few restrictions.

**Pandas**:

**pandas** is a Python programming language software library written for use in. The package's principal aim is to control and analyze data. It provides only data structures and other operations to manipulate numerical tables and statistics. This program is fully free and was published under the BSD license with three clauses. The library name derives from the term "panel data," a term for data sets which uses observations for identical individuals from multiple time periods.

**Features**:

1. DataFrame object with Integrated Indexing for manipulating data.
2. It provides resources between in-memory data structures and various file formats for both reading and writing data.
3. Data synchronization, and streamlined data management.
4. Pivoting and reshaping of huge data sets.
5. Label slicing, clever indexing and subsetting of large data sets.
6. Insertion and deletion of column Data Structure.
7. This also comes with a community by engine that enables operations on data sets with split-apply-combine.
8. Data set merging and joining.

**9. CONCLUSION**

Nowadays, almost every system is prone to attacks. There is a need to increase the security in our daily use systems. This can be achieved by using Intrusion Detection Systems. It helps us in detecting malicious activities efficiently. Another type of Intrusion detection system is used to detect anomalies. But, the current software which detects anomalies detects high number false positives which leads to an increase in the rate of false alarms. Also the results obtained were highly inaccurate and in many cases some types of attacks were not able to be detected. Therefore we conducted an experiment to assess the accuracies and detection rates of various algorithms using the NSL-KDD dataset. According to our results we were able to conclude that our approach i.e. Single Level Multimodel using Decision Trees, has performed exceptionally well and has shown very low rates of false alarms. We have taken into consideration the factors like Accuracy, Precision, Recall and F-Measure to select our approach.

**10. REFERENCES:**

Precision and recall**:** [Precision and Recall Definition](https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall)

Numpy: <https://numpy.org/>

Code in colab: <https://colab.research.google.com/drive/1y4SmxQa-i16VW62zePzlFuBL2poWaV9t>

Decision tree classifier: [sklearn.tree.DecisionTreeClassifier — scikit-learn 0.22.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

NSL-KDD Dataset: [NSL-KDD](https://www.kaggle.com/hassan06/nslkdd)