MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MINOR PROJECT

ON

NETWORK INTRUSION DETECTION SYSTEM

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF BACHELOR OF TECHNOLOGY

SUBMITTED BY:

RAHUL KUMAR JAIN	**141112224**
RAJASHEKAR REDDY	**141112259**
KORVA RAMESH	**141112295**
SAURAV BILUNG	**141112280**

UNDER THE GUIDANCE OF

DR. NAMITA TIWARI

SESSION 2016-17

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that **Rahul Kumar Jain, Rajashekar Reddy, Korva Ramesh** and **Saurav Bilung** students of B.Tech 3rd Year (Computer Science & Engineering),have successfully completed their project
“**Network Intrusion Detection System**” in partial fulfillment of their minor project in Computer Science & Engineering.

Dr. Namita Tiwari	Dr. Rajesh Wadhvani
(Project Guide)	(Project Coordinator)

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, hereby, declare that the following report which is being presented in the Minor Project Documentation entitled "**Network Intrusion Detection System**" is the partial fulfillment of the requirements of the third year (sixth semester) Minor Project in the field of Computer Science and Engineering. It is an authentic documentation of our own original work carried out under the able guidance of Dr. Namita Tiwari. The work has been carried out entirely at Maulana Azad National Institute of Technology, Bhopal. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization.

We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the ones to take responsibility.

RAHUL KUMAR JAIN	**141112224**
RAJASHEKAR REDDY	**141112259**
KORVA RAMESH	**141112295**
SAURAV BILUNG	**141112280**

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide Dr. Namita Tiwari & Project Coordinator Dr. Rajesh Wadhvani, for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully. His rigorous evaluation and constructive criticism was of great assistance.

We are also grateful to our respected director Dr. Narendra S Chaudhary for permitting us to utilize all the necessary facilities of the college.

Needless to mention is the additional help and support extended by our respected HOD, Dr. R. K. Pateriya, in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help.
Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much-needed support and encouragement.

Contents

ABSTRACT

An Intrusion Detection System (IDS) is a software application that monitors networks or system activities for all the malicious activities and unauthorized access to devices.

Intrusion Detection System (IDS) is software or hardware system that automates the process of monitoring and analyzing the events that occur in a computer network, to detect malicious activity. Since the severity of attacks occurring in the network has increased drastically, Intrusion detection system have become a necessary addition to security infrastructure of most organizations.

There are networks based intrusion detection systems (NIDS) and host based intrusion detection system (HIDS). HIDS examines the activity on individual computer or host on which the IDS is installed. The activities include login attempts, process schedules, system files integrity checking system call tracing etc. NIDS monitors and analyzes the individual packets passing around a network for detecting attacks or malicious activities happening in a network that are designed to be overlooked by a firewall's simplistic filtering rules.

Two types of intrusion detection methods are used in NIDS. Signature based detection method and anomaly based detection method, are used in network intrusion detection system. In signature based detection technique, IDS uses database containing attack signatures, to detect intrusion in the data and for this reason it has good detection rate .But the problem with that detection scheme is that it cannot detect novel attacks because no signatures yet exist for that novel attack. Contrarily anomaly based detection technique looks for the unusual behavior and that is why it can detect novel attacks. Anomaly based detection method makes a threshold for normal behavior of system or network. It is done by assimilate the normal behavior of the system or network in a learning phase. Any process that violates a certain threshold is considered as a possible intrusion. Network based detection technique, however, faces a lot of false alarms since to define the normal behavior of the system is very difficult.

In this project we have applied supervised algorithms for anomaly detection model. These algorithms are support vector machine (SVM), Naive Bayes classifier and Decision tree. These algorithms are selected for their very efficient performance. The algorithms are trained and tested on KDD99 dataset. This dataset is being used widely as a benchmark dataset to see the detection model capability.

1. INTRODUCTION

Internet is forcing organizations into an era of open and trusted communications. This openness at the same time brings its share of vulnerabilities and problems such as financial losses, damage to reputation, maintaining availability of services, protecting the personal and customer data and many more, pushing both enterprises and service providers to take steps to guard their valuable data from intruders, hackers and insiders. Intrusion Detection System has become the fundamental need for the successful content networking.

IDS provide two primary benefits: Visibility and Control. It is the combination of these two benefits that makes it possible to create and enforce an enterprise security policy to make the private computer network secure.
There are two types of Intrusion Detection System.

Host based IDS (HIDS):

Examines the activity on individual computer or host on which the IDS is installed. The activities include login attempts, process schedules, system files integrity checking system call tracing etc.

Network based IDS (NIDS):

Monitors and analyzes the individual packets passing around a network for detecting attacks or malicious activities happening in a network that are designed to be overlooked by a firewall's simplistic filtering rules.
Sometimes two kinds of IDS are combined together to form a Hybrid IDS.
Two types of intrusion detection methods are used in NIDS.

1. Signature based detection method
2. anomaly based detection method

1.1 Signature based detection method

Signature based detection works in a similar fashion to a virus scanner. This style of detection relies on rules and tries to associate possible patterns to intrusion attempts. Viruses are known to often attempt a series of steps to penetrate a system. This series of steps would be compiled into such a rule. Whenever the IDS software (an agent) collects the data it then compares what it has observed against the rules that have been defined and then has to decide whether it is a positive or a negative attempt.

Advantages of Signature Based Detection :

- Signature based detection often considered to be much more accurate at identifying an intrusion attempt.
- In Signature based detection time is saved since administrators spend less time dealing with false positives.
- Signature based detection systems are fast since they are only doing a comparison between what they are seeing and a predetermined rule.

Disadvantages of Signature Based Detection:

- Signature based systems can only detect an intrusion attempt if it matches a pattern that is in the database, therefore causing databases to constantly be updated.
- Whenever a new virus or attack is identified it can take vendors anywhere from a few hours to a few days to update their signature databases.
- Signature based systems can suffer a substantial performance slow down if not properly equipped with the necessary hardware to keep up with the demands.

1.2 Anomaly Based Intrusion detection method

An anomaly is defined as something that is not nominal or normal. Anomaly detection is split into two separate categories: static and dynamic.

Static:

1. It assumes that one or more sections on the host should remain constant.
2. It focus only on the software side and ignore any unusual changes in hardware.
3. It is used to monitor data integrity.

Dynamic:

1. It depends on a baseline or profile.
2. In this baseline established by IDS or network administrator.
3. In this baseline tells the system what kind of traffic looks normal.
4. It may include information about bandwidth, ports, and time frames.

Advantages of Anomaly Based Detection:

- New threats can be detected without having to worry about databased being up to date.
- Very little maintenance once system is installed it continues to learn about network activity and continues to build its profiles.
- The longer the system is in use the more accurate it can become at identifying threats.

Disadvantages of Anomaly Based Detection:

- The network can be in an unprotected state as the system builds its profile.
- If malicious activity looks like normal traffic to the system it will never send an alarm.
- False positives can become cumbersome with an anomaly based setup. Normal usage such as checking e-mail aftera meeting has the potential to signal an alarm.

The problem with Signature based Intrusion detection scheme is that it cannot detect novel attacks because no signatures yet exist for that novel attack. But anomaly based detection technique looks for the unusual behavior and that is why it can detect novel attacks. So we implemented anomaly based Intrusion Detection System.

2. LITERATURE REVIEW

2.1 KDD DATASET INFORMATION

The KDD data set is a well-known benchmark in the research of Intrusion Detection techniques. The KDD Cup 99 dataset, which derived from the DARPA IDS evaluation dataset, was used for the KDD Cup 99 Competition (KDD Cup 99 Dataset, 2009). The complete dataset has almost 5 million input patterns and each record represents a TCP/IP connection that is composed of 41 features that are both qualitative and quantitative in nature The dataset used in our study is a smaller subset (10% of the original training set), that contains 494,021 instances and it was already employed as the training set in the competition. For the test set, we used the original KDD Cup 99 dataset containing 331,029 patterns.

The dataset contains normal packets along with attack packets. The attacks that are included in the dataset can be broadly classified into 4 categories. The types of attacks present in the dataset are included in each category:

Types of attacks:

Denial of Service (DoS): attacks, where an attacker makes some computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate users access to a machine.

Probe attacks: where an attacker scans a network to gather information or find known vulnerabilities.

Remote-to-Local (R2L) attacks: where an attacker sends packets to a machine over a network, then exploits machines vulnerability to illegally gain local access as a user.

User-to-Root (U2R) attacks: where an attacker starts out with access to a normal user account on the system and is able to exploit vulnerability to gain root access to the system.

Probe	Ipsweep , Nmap, Portsweep, Satan
DOS	Back, Land, Neptune, Pod, Smurf, Teardrop
U2R	Buffer_overflow, Loadmodule, Perl, Rootkit
R2L	Ftp_write, Guess_passwd, Imap, Multihop, Phf, Spy, Warezclient, Warezmaster

The dataset contains 42 features out of which 34 are continuous, 7 are nominal, and the last feature field is the class label indicating the nature of the packets - either ‘normal’ or the type of attack, as specified in Table 1.

1.TABLE FEATURES OF KDD CUP’99 DATASET:

Sr.No	Feature Name
1	Duration
2	Protocol_type
3	Service
4	Flag
5	Src_bytes
6	Dst_bytes
7	Land
8	Wrong_fragment
9	Urgent
10	Hot
11	Num_failed_logins
12	Logged_in
13	Num_compromised
14	Root_shell
15	Su_attempted
16	Num_root
17	Num_file_creations
18	Num_shells
19	Num_acess_files
20	Num_outboun_cms
21	Is_host_login
22	Is_guest_login
23	Count
24	Srv_count
25	Serror_rate
26	Srv_serror_rate
27	Rerror_rate
28	Srv_rerror_rate
29	Same_srv_rate
30	Diff_srv_rate
31	Srv_iff_host_rate
32	Dst_host_count
33	Dst_host_srv_count
34	Dst_host_same_srv_rate
35	Dst_host_diff_srv_rate
36	Dst_host_same_src_port_rate
37	Dst_host_srv_diff_host_rate
38	Dst_host_serror_rate
39	Dst_host_srv_serror_rate
40	Dst_host_rerror_rate
41	Dst_host_srv_rerror_rate
42	Normal or Attack

2.2 Data set transformation

For two class classification(Normal or Attack):

The training dataset of KDD which contains approximately 4,900,000 single connection instances. Each of the connection instancescontains 42 features including attacks or normal. From these labeled connection instances, we need to convert the nominal features to numeric values so that it suitable input for classification using machine learning techniques.

0 tcp,ftp_data SF 491 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 1 0 0 150 25 0.17 0.03 0.17 0 0 0 0.05 0 normal
0 udp other SF 146 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 1 0 0 0 0 0.08 0.15 0 255 1 0 0.6 0.88 0 0 0 0 0 normal

For this transformation, we will be using table 2. Also, we have to assign a numeric value to the last feature within the connection instance which is a target class. For doing this, we have assigned a target class zero for normal connection and one for any deviation from that (i.e. if that is an attack) as per transformation table 2.In this step, some useless data is filtered and then modified. For example, some text items need to be converted into numeric values. Each and every instance in the dataset has 42 features or attributes including target class shown in Table II.

II.TABLE
TRANSFORMATION TABLE

Type	Feature Name	Numeric Value
Attack or Normal	Normal	0
Attack or Normal	Attack	1
Protocol_type	TCP	2
	UDP	3
	ICMP	4
Flag	OTH	5
	REJ	6
	RSTO	7
	RSTOS0	8
	RSTR	9
	S0	10
	S1	11
	S2	12
	S3	13
	SF	14
	SH	15
Service	All Services	16 to 81

0,2,32,14,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,0,0,0,1,0,0,150,25,0.17,0.03,0.17,0,0,0,0.05,0,0
0,3,16,14,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,1,0,0,0,0,0.08,0.15,0,255,1,0,0.6,0.88,0,0,0,0,0,0

For five class classification(Normal or type of attacks):

Rest of the attributes other than class attributes will have same values as stated above. The values of class attributes are as follows:

Normal	0
Dos	1
R2L	2
Probe	3
U2R	4

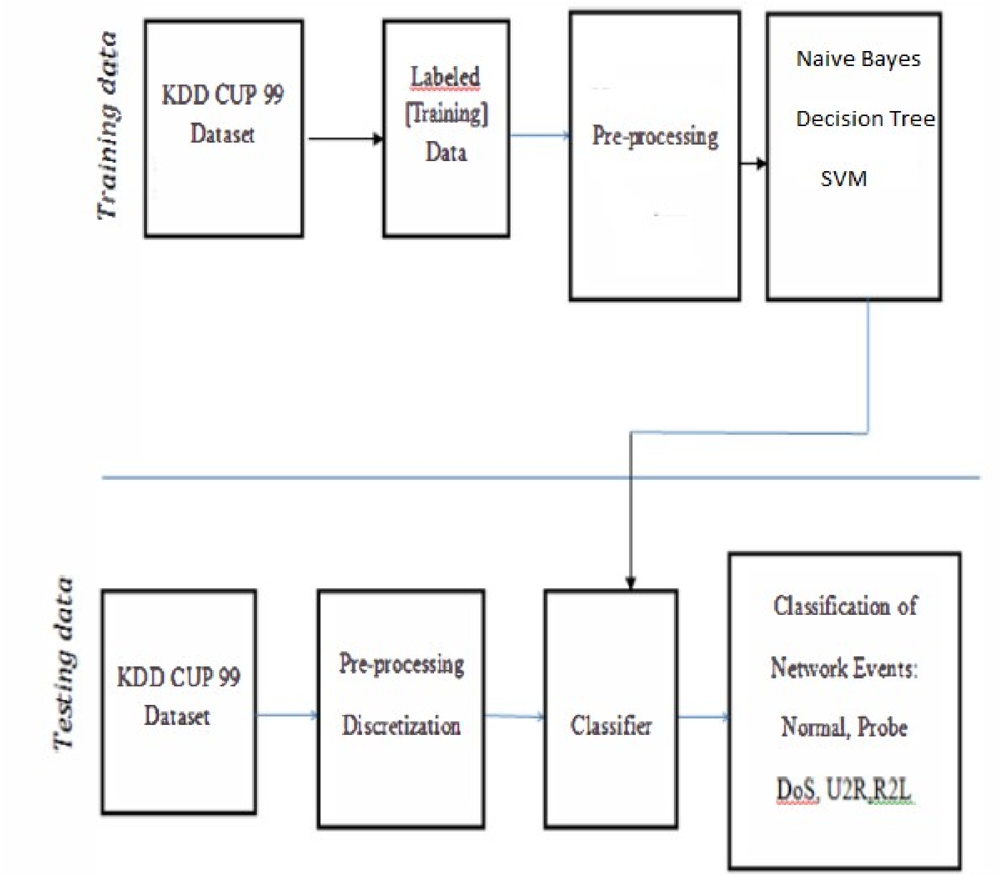
2.3 Splitting of datasets into training and testing

The training data consist of a set of training examples/instances [80% sampled over complete dataset] where each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal or labeled class). The remaining 20% sample of dataset will later be used towards testing part.

We have provided two files for training and testing. Both are text files. Training file is used for training purpose and the testing file is used for testing the datasets.

2.4 Classifiers

We have used three classifiers namely Naive Bayes, Decision Tree and Support Vector Machines (SVM). These classifiers are explained in detail in methodologies section.



3. PROPOSED WORK

We have worked on three popular approaches for doing intrusion detection system. We started with studying them, then we implemented the three techniques in python languages, and then we evaluated them by running them on KDD dataset and also compare the three techniques among themselves

The Three techniques implemented are:

1. **Naïve Bayes:**
   We have taken 10% of KDD dataset and preprocessed it and we have applied Naïve Bayes classifier on that preprocessed data followed by training and testing. After that we have calculated accuracy , recall ,F1 measure, confusion matrix
   and precision.

2. **SVM:**
   We have taken 10% of KDD dataset and preprocessed it and we have applied
   Support Vector Machine (SVM) classifier on that preprocessed data followed by training
   and testing. After that we have calculated accuracy, recall, F1 measure, confusion
   Matrixand precision.

3. **Decision Tree**:
   We have taken 10% of KDD dataset and preprocessed it and we have applied Decision Tree classifier on that preprocessed data followed by training
   and testing. After that we have calculated accuracy, recall, F1 measure, confusion
   Matrixand precision.

4.SOFTWARE AND HARDWARE REQUIREMENTS

4.1 Software Requirements:

The following softwares were used for this project:

- Operating system : Microsoft Windows 10
- Python Software : Pycharm
- Scikit programming

4.2 Hardware Requirements:

The following hardware configuration is required to run the various softwares for this project.

- Processor : Intel Core i3 CPU
- Memory : 2GB RAM
- Storage Required : 50 MB
- Graphics Card : Not Needed

5.METHODOLOGY

5.1 Naive Bayes

Naive Bayes classifier infers that for a given class,features are independent. Using the most frequent values of the features naive Bayes classifier dispense the
class label to the instances. It calculates the prior probability of each class in the training phase using the occurrences of the each feature for each class. Naive
Bayes finds the posterior probability of the class based on the class prior probability. It deduce that the result of the predictor for a given class is independent of the values of other predictor. Using the aforementioned probabilities
it assigns the class label to the new data.

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

Gaussian Naïve Bayses

implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma^2_y}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma^2_y}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

```
def naive_bayses_training():
    print("naive bayses classification algorithm")
    clf = GaussianNB()
    clf.fit(X_train, Y_train)
    GaussianNB(priors=None)

    filename = 'naive.sav'
    pickle.dump(clf, open(filename, 'wb'))

    predictions = clf.predict(X_validation)
    print(accuracy_score(Y_validation, predictions))
    print(confusion_matrix(Y_validation, predictions))
    print(classification_report(Y_validation, predictions))
```

5.2 Support Vector Machines(SVM)

Support vector machine belongs to supervised classification algorithm that linearly separates the data. It separates the classes using hyper plan which is a maximum separable line used to separate the classes data, Fig. 2. SVM uses class labelled data in training phase just like other supervised classification algorithms. Testing is done by consignment of class label to the instances in the testing phase. SVM maps the data into feature space and separates the data into its classes by the hyper plan that has maximum margin between the instances of the classes. Though SVM is a binary classifier, it can do multiclass classification as well. In multiclass classification approach numerous binary SVM is used incascade manner. Two different methods, one-vs-all and one-vs-one, are used for multiclass classification. Classifier model is created for each class in one-vs-all method whereas in one-vs-one binary classifier is built for among different classes. N classifier models are created in one-vs-all strategy for N number of classes while in one-vs-on N(N-1)/2 classifier models are build. Since SVM plot hyperplane for linearly separable data but not all the data are linearly separable, as shown in Fig. 3. To deal with that kind of data SVM maps the input features into high dimension space use kernel trick , making it linearly separable data. There are basically four kernel functions, Linear, Polynomial, Radial basis function (RBF), and sigmoid.

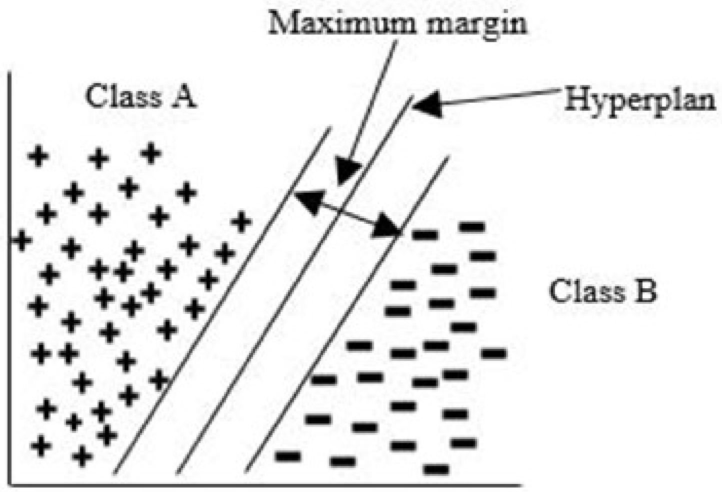


Fig. Linearly Separable Data

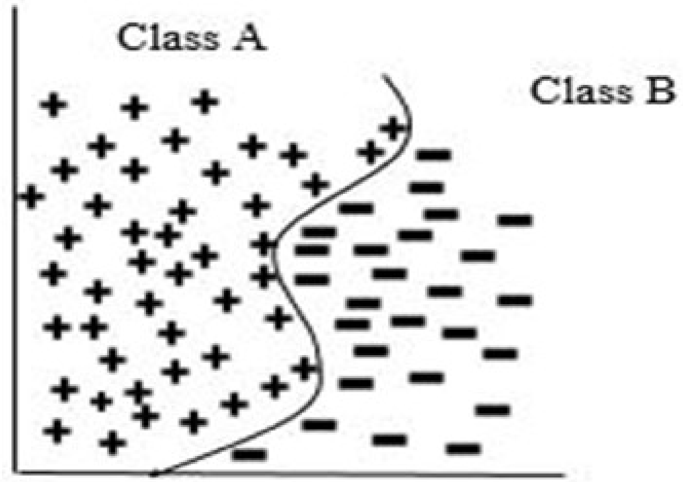


Fig. Linearly Non-Separable Data

```
def svm_training():
    print("SVM classification algorithm")
    sv = svm.SVC()
    sv.fit(X_train, Y_train)

    filename = 'svm.sav'
    pickle.dump(sv, open(filename, 'wb'))

    predictions = sv.predict(X_validation)
    print(accuracy_score(Y_validation, predictions))
    print(confusion_matrix(Y_validation, predictions))
    print(classification_report(Y_validation, predictions))
```

5.3 Decision Tree

A decision tree is a classifier expressed as a recursive partition of the in-stance space. The decision tree consists of nodes that form a *rooted tree*, meaning it is a *directed tree* with a node called “root” that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an *internal* or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attribute’s value. In the case of numeric attributes, the condition refers to a range.

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Figure 9.1 describes a decision tree that reasons whether or not a potential customer will respond to a direct mailing. Internal nodes are represented as circles, whereas leaves are denoted as tri-angles. Note that this decision tree incorporates both nominal and numeric at-tributes. Given this classifier, the analyst can predict the response of a potential customer (by sorting it down the tree), and understand the behavioral charac-teristics of the entire potential customers population regarding direct mailing. Each node is labeled with the attribute it tests, and its branches are labeled with its corresponding values.

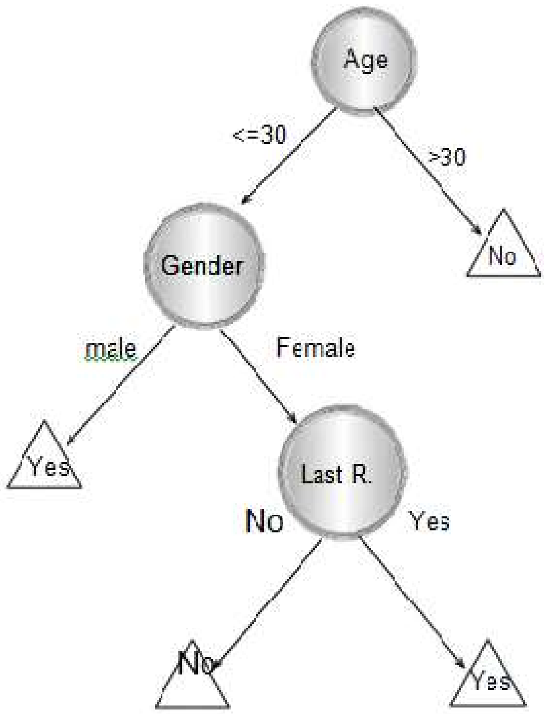


Fig. Decision Tree Presenting Response to Direct Mailing.

```
def decision_tree_training():
    print("decision tree  classification algorithm")
    clf = tree.DecisionTreeClassifier()
    clf.fit(X_train, Y_train)

    filename = 'decision_tree.sav'
    pickle.dump(clf, open(filename, 'wb'))

    predictions = clf.predict(X_validation)
    print(accuracy_score(Y_validation, predictions))
    print(confusion_matrix(Y_validation, predictions))
    print(classification_report(Y_validation, predictions))
```

6.IMPLEMENTATION AND CODING

6.1 Two class and five class classification with KDD 10% dataset:

```
import pandas
import numpy as np
import pickle
from pandas.tools.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.svm import SVC
from sklearn import svm

#loading of the data
url="kddcup-2 classification.data_10_percent_corrected"
names = ['a', 'b', 'c', 'd',
'e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','aa','ab','ac','ad','ae','
af','ag','ah','ai','aj','ak','al','am','an','ao','class']
dataset = pandas.read_csv(url, names=names)
print("data loaded successfully")
print(dataset.shape)

#printing dataset description and classes of datasets
def details_of_dataset():
print(dataset.describe())
print(dataset.groupby('class').size())


#showing dataset through histogram
def show_dataset_histogram():
  dataset.hist()
  plt.show()
```

```
#showing dataset through scatter plotting
def show_dataset_scatter():
  scatter_matrix(dataset)
  plt.show()

#spliting the loaded dataset into two,
#80% of which we will use to train our models and 20% that we will hold back as a validation dataset.
array = dataset.values
X = array[:,0:40]
Y = array[:,41]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y,
test_size=validation_size, random_state=seed)
length=len(X_validation)

#acuuracy=ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100

#buidling models
def model_accuracy():
  seed = 7
scoring = 'accuracy'

models = []
  models.append(('CART', DecisionTreeClassifier()))
  models.append(('NB', GaussianNB()))
# models.append(('SVM', SVC()))

  # evaluate each model in turn
results = []
  names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)


def naive_bayses_training():
print("naive bayses classification algorithm")
  clf = GaussianNB()
  clf.fit(X_train, Y_train)
  GaussianNB(priors=None)
```

```
    filename = 'naive.sav'
  pickle.dump(clf, open(filename, 'wb'))

    predictions = clf.predict(X_validation)
  print(accuracy_score(Y_validation, predictions))
  print(confusion_matrix(Y_validation, predictions))
  print(classification_report(Y_validation, predictions))


  def decision_tree_training():
  print("decision tree  classification algorithm")
    clf = tree.DecisionTreeClassifier()
    clf.fit(X_train, Y_train)

    filename = 'decision_tree.sav'
  pickle.dump(clf, open(filename, 'wb'))

    predictions = clf.predict(X_validation)
  print(accuracy_score(Y_validation, predictions))
  print(confusion_matrix(Y_validation, predictions))
  print(classification_report(Y_validation, predictions))


def svm_training():
print("SVM classification algorithm")
  sv = svm.SVC()
  sv.fit(X_train, Y_train)

  filename = 'svm.sav'
pickle.dump(sv, open(filename, 'wb'))

  predictions = sv.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))


#functions calls
#details_of_dataset()
#show_dataset_histogram()
#show_dataset_scatter()
#model_accuracy()
naive_bayses_training()
decision_tree_training()
#knn_training()
svm_training()
```

Naïve Bayes two class classifier:

```
Run  main
data loaded successfully
naive bayses classification algorithm
0.980385608016
[[19237   339]
 [ 1599 77630]]
             precision    recall  f1-score   support

        0.0       0.92      0.98      0.95     19576
        1.0       1.00      0.98      0.99     79229

avg / total       0.98      0.98      0.98     98805
```

Naïve Bayes 5 class classifier:

```
Run  main
C:\Users\jainr\Anaconda3\python.exe C:/Python27/minor/data/iris/main.py
data loaded successfully
naive bayses classification algorithm
0.721319771267
[[11173  5991   230  1515   667]
 [  423 59209     0 18507    28]
 [   11     9    75     1   127]
 [    1    19     1   806     3]
 [    2     0     0     0     7]]
             precision    recall  f1-score   support

        0.0       0.96      0.57      0.72     19576
        1.0       0.91      0.76      0.83     78167
        2.0       0.25      0.34      0.28       223
        3.0       0.04      0.97      0.07       830
        4.0       0.01      0.78      0.02         9

avg / total       0.91      0.72      0.80     98805


Process finished with exit code 0
```

Decision Tree two class classifier:

```
Run  main
data loaded successfully
decision tree  classification algorithm
0.999757097313
[[19565    11]
 [   13 79216]]
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00     19576
        1.0       1.00      1.00      1.00     79229

avg / total       1.00      1.00      1.00     98805
```

Decision Tree 5 class classifier:

```
Run  main
C:\Users\jainr\Anaconda3\python.exe C:/Python27/minor/data/iris/main.py
data loaded successfully
decision tree  classification algorithm
0.999554678407
[[19556     2     4     7     7]
 [    4 78162     0     1     0]
 [    5     0   216     1     1]
 [    6     4     0   820     0]
 [    2     0     0     0     7]]
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00     19576
        1.0       1.00      1.00      1.00     78167
        2.0       0.98      0.97      0.98       223
        3.0       0.99      0.99      0.99       830
        4.0       0.47      0.78      0.58         9

avg / total       1.00      1.00      1.00     98805


Process finished with exit code 0
```

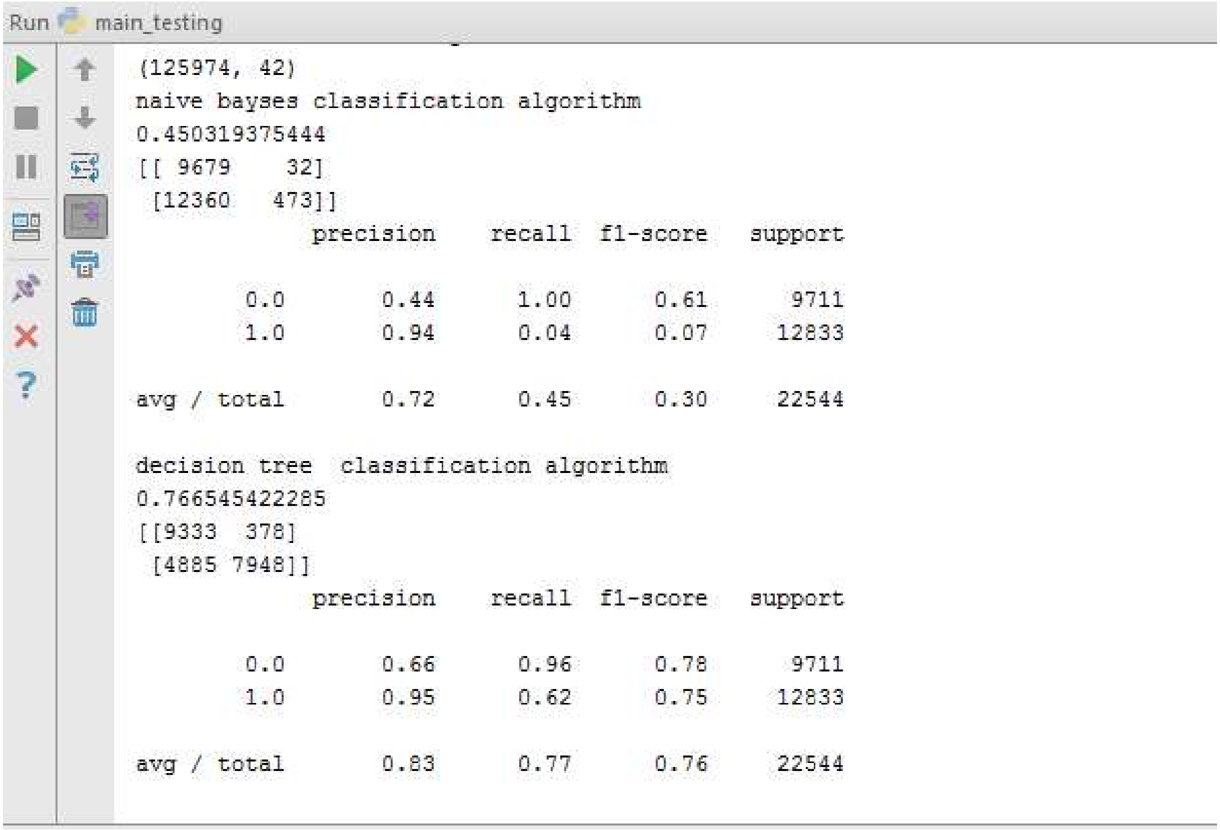
6.2 Two class and five class classification with two files (1 training and 1 testing file)

```
import pandas
import numpy as np
import pickle
from pandas.tools.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.svm import SVC
from sklearn import svm

#loading of the data
url="KDDTrain.txt"
names = ['a', 'b', 'c', 'd',
'e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','aa','ab','ac','ad','ae','af','ag','ah
','ai','aj','ak','al','am','an','ao','class']
dataset = pandas.read_csv(url, names=names)
print("data loaded successfully")
print(dataset.shape)

#printing dataset description and classes of datasets
def details_of_dataset():
print(dataset.describe())
print(dataset.groupby('class').size())


#showing dataset through histogram
def show_dataset_histogram():
  dataset.hist()
  plt.show()

#showing dataset through scatter plotting
def show_dataset_scatter():
  scatter_matrix(dataset)
  plt.show()
```

```
#spliting the loaded dataset into two,
#80% of which we will use to train our models and 20% that we will hold back as a validation
dataset.
array = dataset.values
X_train = array[:,0:40]
Y_train = array[:,41]
X_validation=np.loadtxt("KDDTest.txt",delimiter=",",usecols=range(0,40))
y_test,Y_validation=np.loadtxt("KDDTest.txt",delimiter=",",usecols=(0,41),unpack=True)
length=len(X_validation)

#acuuracy=ratio of the number of correctly predicted instances in divided by the total number of
instances in the dataset multiplied by 100

def naive_bayses_training():
print("naive bayses classification algorithm")
  clf = GaussianNB()
  clf.fit(X_train, Y_train)
  GaussianNB(priors=None)

  filename = 'naive_testing.sav'
pickle.dump(clf, open(filename, 'wb'))

  predictions = clf.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

  f = open("result_naive.txt", "w")
  f.write("Prediction \t Actual \n")
for i in range(0, length-1):
    f.write('%d' % predictions[i])
    f.write("\t")
    f.write('%d' % Y_validation[i])
    f.write("\n")
  f.close()


def decision_tree_training():
print("decision tree  classification algorithm")
  clf = tree.DecisionTreeClassifier()
  clf.fit(X_train, Y_train)

  filename = 'decision_tree_testing.sav'
pickle.dump(clf, open(filename, 'wb'))

  predictions = clf.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
```

```
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

  f = open("result_tree.txt", "w")
  f.write("Prediction \t Actual \n")
for i in range(0, length - 1):
    f.write('%d' % predictions[i])
    f.write("\t")
    f.write('%d' % Y_validation[i])
    f.write("\n")
  f.close()

def svm_training():
print("SVM classification algorithm")
  sv = svm.SVC()
  sv.fit(X_train, Y_train)

  filename = 'svm_testing.sav'
pickle.dump(sv, open(filename, 'wb'))

  predictions = sv.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

  f = open("result_tree.txt", "w")
  f.write("Prediction \t Actual \n")
for i in range(0, length - 1):
    f.write('%d' % predictions[i])
    f.write("\t")
    f.write('%d' % Y_validation[i])
    f.write("\n")
  f.close()

#functions calls
#details_of_dataset()
#show_dataset_histogram()
#show_dataset_scatter()

naive_bayses_training()
decision_tree_training()
#knn_training()
svm_training()
```

Naïve Bayes and Decision Tree Results:



```
Run  main_testing
(125974, 42)
naive bayses classification algorithm
0.450319375444
[[ 9679    32]
 [12360   473]]
             precision    recall  f1-score   support

        0.0       0.44      1.00      0.61      9711
        1.0       0.94      0.04      0.07     12833

avg / total       0.72      0.45      0.30     22544

decision tree  classification algorithm
0.766545422285
[[9333  378]
 [4885 7948]]
             precision    recall  f1-score   support

        0.0       0.66      0.96      0.78      9711
        1.0       0.95      0.62      0.75     12833

avg / total       0.83      0.77      0.76     22544
```

7.RESULT ANALYSIS

Validation of results:

Validation of results in nothing but analyzing the performance of trained_classifier_model.
In this project I used following factors for performance analysis: -

Accuracy:
Accuracy= (no of correctly detected intrusion)/(total no of records)
Accuracy(in %): [(TP + TN) / (TP + TN + FP + FN)] * 100

Precision:
Precision=(TP) / (TP + FP)

Recall:
Recall= (TP) / (TP + FN)

F-measure:
F-measure= (2TP) / (2TP + FP + FN)

Where:
TP = True prediction of Positive Label (Actual 1, Predicted 1)
TN = True prediction of Negative Label (Actual 0, Predicted 0)
FP = False prediction of Positive Label (Actual 0, Predicted 1)
FN = False prediction of Negative Label (Actual 1, Predicted 0)

8.APPLICATIONS

Possible uses of Network Intrusion Detection System:

1. **Real Time detection and quick response:**
   Network based IDS monitors traffic on a real time. So, network based IDS can detect malicious activity as they occur. Based on how the sensor is configured, such attack can be stopped even before they can get to a host and compromise the system.

2. Supports customization of detection capabilities to stop activity that is only of concern to a single organization

3. Reduces the amount of network traffic reaching other security controls, which both lowers the workload for those controls and protects those controls from direct attacks

CONCLUSION

As stated earlier, the signature based techniques are good but has the obvious short comings like failure to detect novel attacks, increasing signature database etc. So the viable alternative would be to analyse the behavior of the network as a whole and trying to build the model based on the observations. So Anomaly based detection has been a wide area of interest for researchers since it provides the base line for developing promising techniques.

The main challenge of anomaly intrusion detection is to minimize false positives.

Presently, the work caters only to identify and classify the events into normal and the attacks into multiple attack classes.

This project gives the comparative study of machine learning algorithms SVM, Naive Bayes, and decision tree for anomaly detection. The performance of the algorithms is tested on KDD99 10% dataset. The overall accuracy of decision tree is high among all other algorithms and low misclassification rate.

1. Algorithms based on Machine Learning were implemented successfully showing different accuracies. KDD 10% dataset was preprocessed . Decision tree classification , surprisingly, proved to be very effective in detecting network attacks with a 78% accuracy. SVMbeing a supervised approach showed decent results with a 76.5 % accuracy. Naïve Bayes shows result with a 50% accuracy.
2. KDD99 has huge records that's why its short version is widely used and is called kddcup.data_10_percent (KDD99_10%). 22 attacks are in training set, 14 additional attacks are in testing set. Training set contains 492021 instances while 311029 instances are for testing dataset .

   KDD99 contains 4 attack classes, Root-to-Local (R2L), Denial of Service (DoS), Probe, User-to-Root (U2R), and one legitimate data class called, Normal. The instances of each class are described by 41 features.

Accuracy of Decision Tree is very much high among all other algorithms. Misclassification rate is also low for Decision Tree. Naive Bayes has high misclassification rate as well as low accuracy rate.

REFERENCES

1. "KDD Cup 1999 Data"
   http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, accessed: 2015-11-28.

2. KDD Cup 1999 Feature Set,"
   http://kdd.ics.uci.edu/databases/kddcup99/task.html, accessed: 2015-11-28.

3. International Conference on Research Advances in Integrated Navigation Systems (RAINS - 2016), April 06-07, 2016, R. L. Jalappa Institute of Technology, Doddaballapur, Bangalore, India
4. Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian, 13-16 August 2006
   -DETECTING WEB-BASED ATTACKS BY MACHINE LEARNING

5. 2016 3rd International Conference On Computer And Information Sciences (ICCOINS)
   -Machine Learning Algorithms In Context Of Intrusion Detection

6. 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)
   -Machine Learning Techniques for Intrusion Detection on KDD Dataset

7. SCIKIT-LEARN documentation
   http://scikit-learn.org

8. Coursera online course of "An Introduction to Interactive Programming in Python"
   https://www.coursera.org

9. An Introduction to Machine Learning
   https://in.udacity.com