MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INTRUSION DETECTION USING MACHINE LEARNING MODELS

Minor Project Report

Semester VI

Arpit Chachane	171112019
Ananya Shrivastava	171112032
Anuj Shrivastav	171112033
Bhavesh Lohar	171112044

Submitted by:
Under the Guidance of
Dr. VASUDEV DEHALWAR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Session: 2019-20

MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project report carried out on "Intrusion Detection Using Machine Learning Models" by the 3rd year students:

Arpit Chachane	171112019
Ananya Shrivastava	171112032
Anuj Shrivastav	171112033
Bhavesh Lohar	171112044

Have successfully completed their project in partial fulfilment of their Degree in Bachelor of Technology in Computer Science and Engineering.

Dr. Vasudev Dehalwar
(Project Guide)

DECLARATION

We, hereby declare that the following report which is being presented in the Minor Project Documentation Entitled as “Intrusion Detection Using Machine Learning Models” is an authentic documentation of our own original work and to best of our knowledge. 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. Any contribution made to the research by others, with whom we have worked at Maulana Azad National Institute of Technology, Bhopal or elsewhere, is explicitly acknowledged in the report. We our solely responsible for all the result and research mentioned in the project.

Arpit Chachane	171112019
Ananya Shrivastava	171112032
Anuj Shrivastav	171112033
Bhavesh Lohar	171112044

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide and coordinator Dr. Vasudev Dehalwar, for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully.

It is imperative for us to mention the fact that the report of minor project could not have been accomplished without the periodic suggestions and advice of our project guide Dr. Vasudev Dehalwar and project coordinators Dr. Dhirendra Pratap Singh and Dr. Jaytrilok Choudhary.

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

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation 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.

ABSTRACT

In this project, we present a comparative evaluation of machine learning approaches to network intrusion detection. A Network Intrusion Detection System (NIDS) is a critical component of every Internet connected system for defense against likely attacks from both external and internal sources. A NIDS is used to detect network born attacks such as Denial of Service (DoS) attacks, malware replication, and intruders that are operating within the system.

Multiple machine learning approaches have been proposed for intrusion detection systems. We evaluate four models, Naive Bayes classifier, Logistic regression, Decision tree, K-nearest neighbours on their accuracy, precision, recall and f1 score. Their performance is evaluated using the network intrusion dataset provided by Knowledge Discovery in Databases (KDD). This dataset was used for the third international Knowledge Discovery and Data Mining Tools competition held in conjunction with KDD Cup 1999. The results were then compared to a baseline shallow algorithm that uses multinomial logistic regression to evaluate if deep learning models perform better on this dataset.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Over the past few decades, the Internet has penetrated all aspects of our lives. Experts predict that by 2020 there would be 50 billion connected devices. As technology becomes more and more integrated, the challenge to keep the systems safe and away from vulnerability attacks increases. Over the years we have seen an increase in hacks in banking systems, healthcare systems and many Internet of Things (IoT) devices. These attacks cause billions of dollars in losses every year and loss of systems at crucial times. This has led to higher importance in cyber security specifically in the intrusion detection systems. A related challenge with most modern-day infrastructure is that data requirements pertaining to security are often difficult to get. A network intrusion detection system (NIDS) is a software application that monitors the network traffic for malicious activity. One popular strategy is to monitor a network's activity for anomalies, or anything that deviates from normal network Anomaly detection creates models of normal behavior for networks and other devices and then looks for deviations from those patterns of behavior at a much faster pace. Machine learning is used to build anomaly detection models and there are two approaches shallow learning and deep learning. Shallow learners mostly depend on the features used for creating the prediction model.[1]

On the other hand, deep learners have the potential to extract better representations from the raw data to create much better models. Machine learners can learn better because they are composed of the multiple hidden layers. At each layer the model can extract a better representation from the feature set when compared to shallow learners who don’t have hidden layers. In this project, we evaluate machine learning models like Naïve Bayes classifier, decision tree, K-nearest neighbors and logistic regression. For this project, we have used the KDD Cup 1999 Dataset and compare the models on Accuracy, Precision, Recall and F1 Score.

Literature Review and Survey

The purpose of this project is to compare different ML models for network intrusion detection. Thus, great emphasis is placed on a thorough description of the ML methods, and references to related works for each ML method are provided. This project does not describe all of the different techniques of network anomaly detection; instead, it concentrates only on few ML techniques.

Yang Xin and Lingshuang Kong [1] compared 39 ML models on both KDD as well as NSL-KDD dataset. However, in this project, we are focusing only on few ML models and compare them.

Brian Lee and Sandhya Amaresh [2] also compared models on the NSL-KDD dataset but more focus has been on the implementation side i.e. “how” part. In our project, we intend to focus on “what” part as well.

Also while researching, we have found some of the methods described worked either for wired networks or for wireless networks. The ML methods covered in this project are applicable to intrusion detection in wired as well as in wireless networks.

GAPS IDENTIFIED

While studying about this topic, we have encountered some problems that are present there in the current research area. The problems were-

(i) The benchmark datasets are few, although the same dataset is used, and the methods of sample extraction used by each institute vary. We can see in research papers [1] and [2] , and nearly all other studies present on the internet, the same dataset has been used again and again.

(ii) The evaluation metrics are not uniform, many studies only assess the accuracy of the test, and the result is one-sided. However, studies using multi-criteria evaluation often adopt different metric combinations such that the research results cannot be compared with one another.

(iii) Less consideration is given to deployment efficiency, and most of the research stays in the lab irrespective of the time complexity of the algorithm and the efficiency of detection in the actual network

PROPOSED WORK

Study about Machine Learning Models, reasons behind choosing a model, how to implement a given model and compare using following performance metrics

Accuracy	Defined as the percentage of correctly classified records over the total number of records
Precision (P)	Defined as the % ratio of the number of true positives (TP) records divided by the number of true positives (TP) and false positives (FP) classified records. P =TP/ (TP + FP) ×100%
Recall (R)	Defined as the % ratio of number of true positives records divided by the number of true positives and false negatives (FN) classified records. R = TP/ (TP + FN) ×100%
F-Measure (F)	Defined as the harmonic mean of precision and recall and represents a balance between them. F = 2.P.R/(P+R)

Table 1: Performance metrics

METHODOLOGY

Since we have intended to work both on “what” and “how” part, we’ll –

- Describe Decision tree, Logistic regression, Naïve Bayes classifier, K nearest neighbor why they are chosen for the dataset, implement the models and finally write the results obtained on each of the models.
- Compare all the results on the basis of performance metrics.

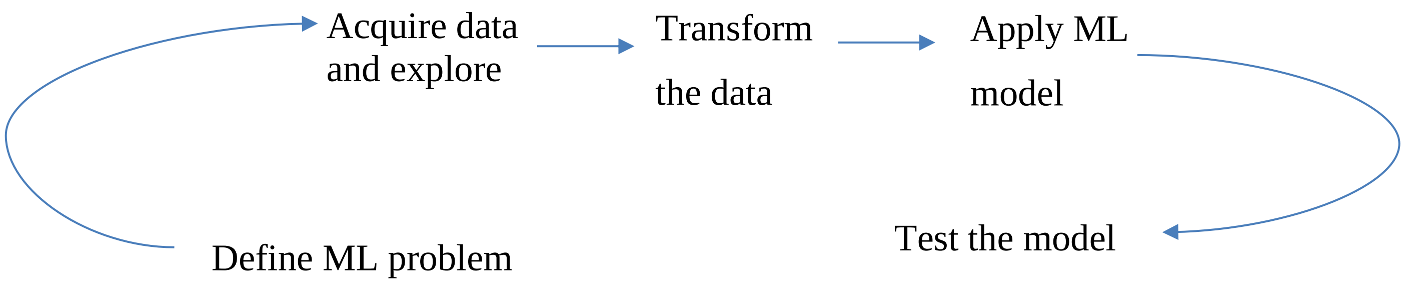


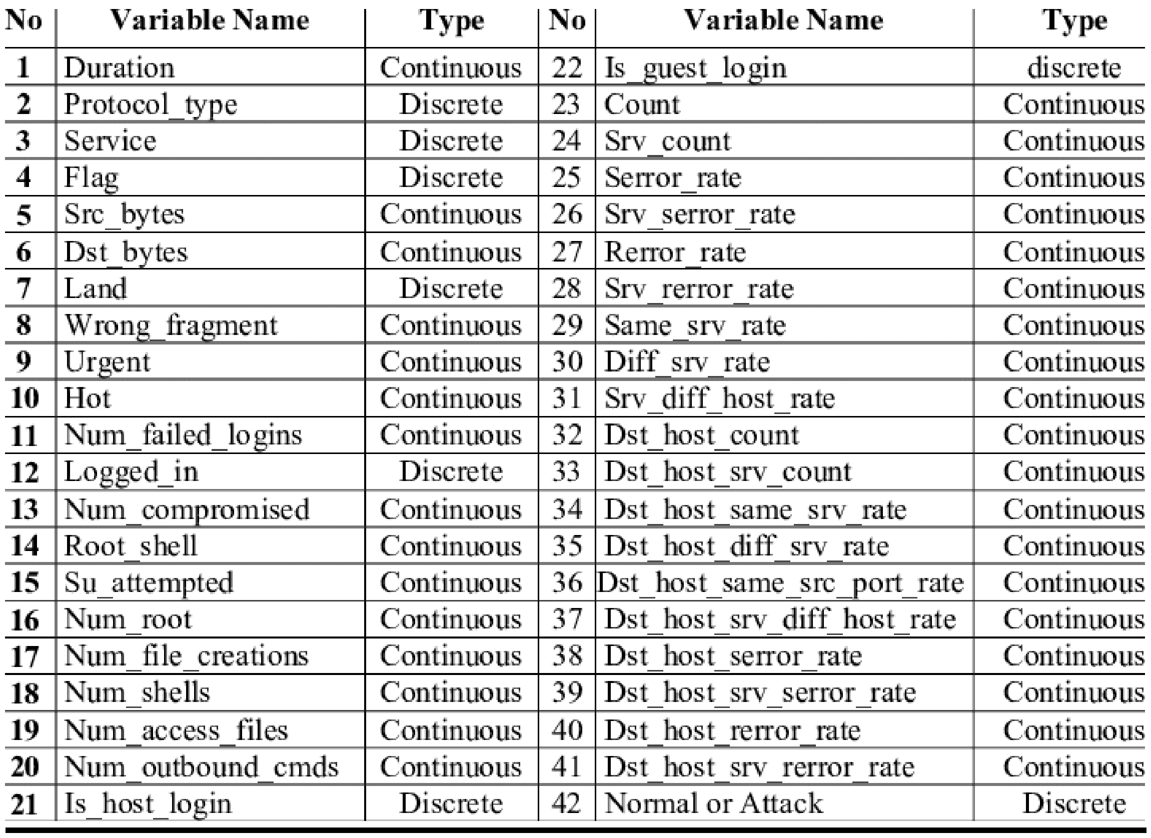
Figure 1: Workflow of project

About the Dataset:

For our work, we’ll use NSL-KDD dataset to see the difference in performances for different models. The KDD Cup dataset was prepared using the network traffic captured by 1998 DARPA IDS evaluation program. The network traffic includes normal and different kinds of attack traffic, such as DoS, Probing, user-to-root (U2R), and root-to-local (R2L).

The KDD Cup dataset has been widely used as a benchmark dataset for many years in the evaluation of NIDS. One of the major drawbacks with the dataset is that it contains an enormous amount of redundant records both in the training and test data. This redundancy makes the learning algorithms biased towards the frequent attack records and leads to poor classification results for the infrequent, but harmful records.

NSL-KDD was proposed to overcome the limitation of KDD Cup dataset. The dataset is derived from the KDD Cup dataset. It improved the previous dataset in two ways. First, it eliminated all the redundant records from the training and test data. Second, it partitioned all the records in the KDD Cup dataset into various difficulty levels based on the number of learning algorithms that can correctly classify the records Each record in the NSL-KDD dataset consists of 41 features and is labeled with either normal or a kind of attack. These features include basic features derived directly from a TCP/IP connection, traffic features accumulated in a window interval, either time, e.g. two seconds, or many connections, and content features extracted from the application layer data of connections.



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

Table 2: KDD 99 Dataset

MODELS USED

Naive Bayes Classifier:

A Naive Bayes Classifier is a supervised machine-learning algorithm that uses the Bayes' Theorem, which assumes that features are statistically independent. The theorem relies on the *naive* assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results.

Naive Bayes Classifiers rely on the Bayes' Theorem, which is based on conditional probability or in simple terms, the likelihood that an event (A) will happen *given that* another event (B) has already happened. Essentially, the theorem allows a hypothesis to be updated each time new evidence is introduced. The equation below expresses Bayes' Theorem in the language of probability:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Let's explain what each of these terms means.

- "P" is the symbol to denote probability.
- P(A | B) = The probability of event A (hypothesis) occurring given that B (evidence) has occurred.
- P(B | A) = The probability of the event B (evidence) occurring given that A (hypothesis) has occurred.
- P(A) = The probability of event B (hypothesis) occurring.
- P(B) = The probability of event A (evidence) occurring.

Logistic Regression:

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values).
Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.
1 / (1 + e^-value)

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modelled is a binary value (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in input data has an associated b coefficient (a constant real value) that must be learned from your training data.

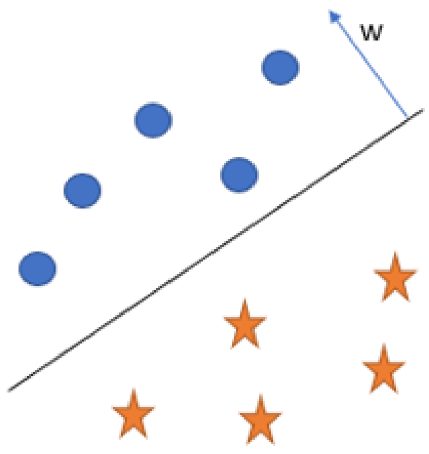


Figure 2: Logistic regression plane

$$\pi : w^T x + b = 0$$

K Nearest Neighbour:

K-nearest neighbour (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

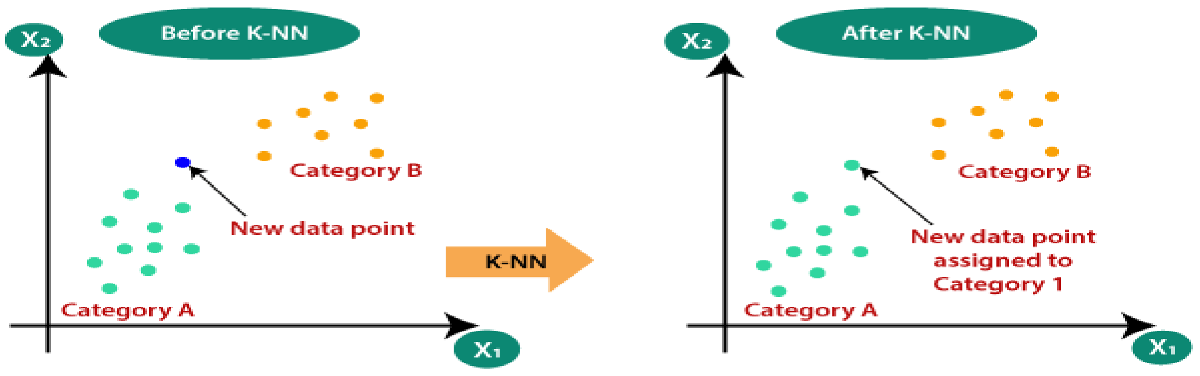


Figure 3: KNN

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbours
- **Step-2:** Calculate the Euclidean distance of **K number of neighbours**
- **Step-3:** Take the K nearest neighbours as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbours, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbour is maximum.
- **Step-6:** Our model is ready.

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Decision Tree:

Decision tree is one of the predictive modelling approaches used in data science and machine learning.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric **supervised learning** method used for both **classification** and **regression** tasks.

Tree models where the target variable can take a discrete set of values are called **classification trees**. Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**. Classification And Regression Tree (CART) is general term for this.

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

Information Gain: It is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class.

According to the value of information gain, we split the node and build the decision tree. A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

- S= Total number of samples
- P(yes)= probability of yes
- P(no)= probability of no

Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:
- Gini Index= 1- $\sum_j P_j^2$

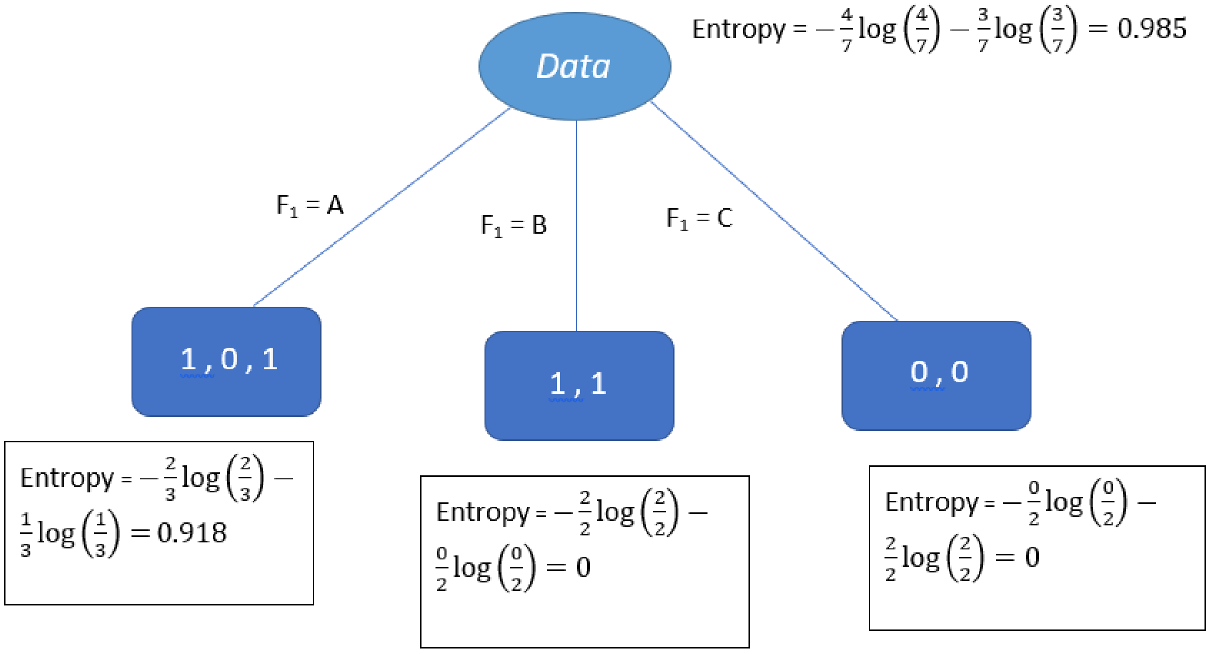

Data
Entropy = $-\frac{4}{7}\log\left(\frac{4}{7}\right)-\frac{3}{7}\log\left(\frac{3}{7}\right)=0.985$
F_1 = A
F_1 = B
F_1 = C
1 , 0 , 1
1 , 1
0 , 0
Entropy = $-\frac{2}{3}\log\left(\frac{2}{3}\right)-\frac{1}{3}\log\left(\frac{1}{3}\right)=0.918$
Entropy = $-\frac{2}{2}\log\left(\frac{2}{2}\right)-\frac{0}{2}\log\left(\frac{0}{2}\right)=0$
Entropy = $-\frac{0}{2}\log\left(\frac{0}{2}\right)-\frac{2}{2}\log\left(\frac{2}{2}\right)=0$

Figure 4: An example of a decision tree

CODE

```
import numpy as np
import pandas as pd
pd.options.display.max_columns=50
np.random.seed(50)

datacols = ["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","attack"]

import random

data=pd.read_csv('./kddcup.data.corrected',names=datacols)

mapping = {'ipsweep.': 'Probe','satan.': 'Probe','nmap.': 'Probe','portsweep.
': 'Probe','saint.': 'Probe','mscan.': 'Probe',
        'teardrop.': 'DoS','pod.': 'DoS','land.': 'DoS','back.': 'DoS','neptu
ne.': 'DoS','smurf.': 'DoS','mailbomb.': 'DoS',
        'udpstorm.': 'DoS','apache2.': 'DoS','processtable.': 'DoS',
        'perl.': 'U2R','loadmodule.': 'U2R','rootkit.': 'U2R','buffer_overflo
w.': 'U2R','xterm.': 'U2R','ps.': 'U2R',
        'sqlattack.': 'U2R','httptunnel.': 'U2R',
        'ftp_write.': 'R2L','phf.': 'R2L','guess_passwd.': 'R2L','warezmaster
.': 'R2L','warezclient.': 'R2L','imap.': 'R2L',
        'spy.': 'R2L','multihop.': 'R2L','named.': 'R2L','snmpguess.': 'R2L',
'worm.': 'R2L','snmpgetattack.': 'R2L',
        'xsnoop.': 'R2L','xlock.': 'R2L','sendmail.': 'R2L',
        'normal.': 'Normal'
        }

data['attack_class']=data['attack'].apply(lambda x: mapping[x])

must_rows = set(np.where(data['attack_class']=='U2R')[0])

rows_to_keep=np.random.choice(list(range(1,4800000)),100000)

rows_to_keep = set(rows_to_keep)
rows_to_keep=list(rows_to_keep.union(must_rows))
```

NOTE: Actual data file contains 48 lacs data points , but using this much data would make computation and processing very slow . Hence , we are taking only around 1 lac points

```
data=data.loc[rows_to_keep]
```

```
data.head()
```

Splitting the data into Train and Test

```
from sklearn.model_selection import train_test_split
```

```
Y=data['attack_class']
data.drop('attack_class',axis=1,inplace=True)
```

```
X_train,X_test,y_train,y_test= train_test_split(data,Y,train_size=0.7,random_
state=42)
```

We can see that for num_outbound_cmds, is_host_login --> all values are zero , hence they are redundant columns and therefore removing them both from train as well as test data

```
X_train.drop(columns=['num_outbound_cmds','is_host_login'],axis=1,inplace=Tru
e)
X_test.drop(columns=['num_outbound_cmds','is_host_login'],axis=1,inplace=True
)


# Attack Class Distribution
attack_class_freq_train = pd.DataFrame(y_train.value_counts())
attack_class_freq_test = pd.DataFrame(y_test.value_counts())
attack_class_freq_train['frequency_percent_train'] = round((100 * attack_clas
s_freq_train / attack_class_freq_train.sum()),2)
attack_class_freq_test['frequency_percent_test'] = round((100 * attack_class_
freq_test / attack_class_freq_test.sum()),2)

attack_class_dist = pd.concat([attack_class_freq_train,attack_class_freq_test
], axis=1,sort=False)


# Attack class bar plot
%matplotlib inline
plot = attack_class_dist[['frequency_percent_train', 'frequency_percent_test'
]].plot(kind="bar");
plot.set_title("Attack Class Distribution", fontsize=20);
plot.grid(color='lightgray', alpha=0.5);
```

Scaling Numerical Attributes

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
# extract numerical attributes and scale it to have zero mean and unit varian
ce
cols = X_train.select_dtypes(include=['float64','int64']).columns
scaler.fit(X_train.select_dtypes(include=['float64','int64']))
sc_train = scaler.transform(X_train.select_dtypes(include=['float64','int64']
))
sc_test = scaler.transform(X_test.select_dtypes(include=['float64','int64']))

# turn the result back to a dataframe
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)
```

Encoding of Categorical Attributes

```
from sklearn.preprocessing import LabelEncoder
encoder= LabelEncoder()

# extract categorical attributes from both training and test sets
cattrain = X_train.select_dtypes(include=['object']).copy()


cattest = X_test.select_dtypes(include=['object']).copy()

# encode the categorical attributes
traincat = cattrain.apply(encoder.fit_transform)
testcat = cattest.apply(encoder.fit_transform)


y_train=pd.DataFrame(y_train,columns=['attack_class'])
y_test=pd.DataFrame(y_test,columns=['attack_class'])

y_train=y_train.apply(encoder.fit_transform)
y_test=y_test.apply(encoder.fit_transform)


sc_traindf.reset_index(drop=True,inplace=True)
traincat.reset_index(drop=True,inplace=True)
sc_testdf.reset_index(drop=True,inplace=True)
testcat.reset_index(drop=True,inplace=True)
X_train=pd.concat([sc_traindf,traincat],axis=1)
X_test=pd.concat([sc_testdf,testcat],axis=1)
```

Feature Selection

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(X_train, y_train.values.ravel());
# extract important features
score = np.round(rfc.feature_importances_,3)

import matplotlib.pyplot as plt
```

```
importances = pd.DataFrame({'feature':X_train.columns,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index
('feature')
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();

final_features=list(importances[:20].index)



X_train2=X_train.loc[:,final_features]
X_test2=X_test.loc[:,final_features]
```

FITTING THE MODELS

```
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier

# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train2, y_train.values.ravel());

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(multi_class='auto',n_jobs=-1, random_stat
e=0,solver='lbfgs')
LGR_Classifier.fit(X_train2, y_train.values.ravel());

# Train Gaussian Naive Baye Model
GNB_Classifier = GaussianNB()
GNB_Classifier.fit(X_train2, y_train.values.ravel())

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_stat
e=0)
DTC_Classifier.fit(X_train2, y_train.values.ravel());



combined_model = [('Naive Baye Classifier', GNB_Classifier),
                 ('Decision Tree Classifier', DTC_Classifier),
                 ('KNeighborsClassifier', KNN_Classifier),
                 ('LogisticRegression', LGR_Classifier)
                ]
VotingClassifier =  VotingClassifier(estimators = combined_model,voting = 'so
ft', n_jobs=-1)
VotingClassifier.fit(X_train2, y_train.values.ravel());

from sklearn import metrics
```

```
models = []
#models.append(('SVM Classifier', SVC_Classifier))
models.append(('Naive Baye Classifier', GNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
#models.append(('RandomForest Classifier', RF_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))
models.append(('VotingClassifier', VotingClassifier))

for i, v in models:
    scores = cross_val_score(v, X_train2, y_train.values.ravel(), cv=10)
    accuracy = metrics.accuracy_score(y_train.values.ravel(), v.predict(X_tra
in2))
    confusion_matrix = metrics.confusion_matrix(y_train.values.ravel(), v.pre
dict(X_train2))
    classification = metrics.classification_report(y_train.values.ravel(), v.
predict(X_train2))
    print()
    print('============================== {} Model Evaluation ===============
==============='.format(i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

EVALUATING ON TEST DATA

```
for i, v in models:
    accuracy = metrics.accuracy_score(y_test.values.ravel(), v.predict(X_test
2))
    confusion_matrix = metrics.confusion_matrix(y_test.values.ravel(), v.pred
ict(X_test2))
    classification = metrics.classification_report(y_test.values.ravel(), v.p
redict(X_test2))
    print()
    print('============================== {} Model Test Results =============
================='.format(i))
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

RESULT

Model	Accuracy	Macro-precision	Macro-recall	Macro-F1-Score
K-nearest neighbour	0.989	0.80	0.76	0.74
Naïve Bayes	0.9103	0.43	0.72	0.42
Logistic regression	0.996	0.73	0.69	0.71
Decision Tree	0.9994	0.97	0.87	0.91

Table 3: Results obtained on dataset

TOOLS & TECHNOLOGY

Software Requirements :

- Anaconda (https://www.anaconda.com/distribution/) distribution of python
- Jupyter Notebook - An IDE for Python programming

Programming Language and Modules:

- Python (https://www.python.org) programming language widely used in Data Science
- Numpy (https://www.numpy.org) library
- Pandas (https://www.pandas.pydata.org) library
- Scikit-learn (https://scikit-learn.org) library

Hardware Requirements:

- PC or Laptop (preferably with 16 GB RAM)
- GPU (preferred)
- Internet Connection

CONCLUSION

Although all the classifiers results in 90+ accuracy, but accuracy may not be the best metric to evaluate them as accuracy loses its meaning when we deal with imbalanced dataset and hence a single metric macro f1 score is used here to compare models.

Among the four classifiers, decision tree classifier performed the best

REFERENCES

[1] Machine Learning and Deep Learning Methods for Cybersecurity, Yang Xin ,Lingshuang Kong , Zhi Liu ,Yuling Chen , Yanmiao Li , Hongliang Zhu ; Mingcheng Gao , Haixia Hou, And Chunhua Wang.

[2] Comparative Study of Deep Learning Models for Network Intrusion Detection Brian Lee,Sandhya Amaresh,Clifford Green,Daniel Engels.