PROJECT REPORT ON

"INTRUSION DETECTION OF A NETWORK BASED ON MACHINE LEARNING"

Submitted

In the partial fulfilment of the

requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

BY

Chava Lakshitha	171FA04384
Chunduri Sravya	171FA04385

Under the guidance of

Mr.Anjaneyulu Nelluru

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH

Deemed to be UNIVERSITY

Vadlamudi, Guntur.

VIGNAN'S FOUNDATION FOR SCIENCE,

TECHNOLOGY AND RESEARCH

Deemed to be UNIVERSITY

VADLAMUDI-522 213, GUNTUR DIST, ANDHRA PRADESH, INDIA.

CERTIFICATE

This is to certify that the Project Report entitled that is being submitted by **Chava Lakshitha(171FA04384) and Chunduri Sravya(171FA04385)** in partial fulfilment for the award of B. Tech degree in Information Technology at Vignan's Foundation for Science, Technology and Research, Deemed to be University, is a record of bonafide work carried out by them at **"VIGNAN'S FOUNDATION FOR SCIENCE ,TECHNOLOGY AND RESEARCH"** under the co-guidance of the following faculty member of Department of CSE.

Mr.Anjaneyulu Nelluru **External Examiner** **Dr.Dondeti Venkatesulu** **Assistant Professor** **Professor, HOD**

ACKNOWLEDGMENT

We are very grateful to our beloved Chairman **Dr. Lavu. Rathaiah**, and Vice Chairman **Mr. Lavu. Krishna Devarayalu,** for their love and care.

It is our pleasure to extend our sincere thanks to Vice-Chancellor **Dr. M.Y.S. Prasad** and Dean Engineering & Management, **Dr. M. Santhi Sree Rukmini,** for providing an opportunity to do my academics in VFSTR.

It is a great pleasure for me to express my sincere thanks to **Dr. Dondeti Venkatesulu HOD, CSE** of VFSTR**,** for providing required infrastructure to do my Project.

We extend our whole hearted gratitude to Asst **Prof**. **Dr.Deepak Raj.D.M**, under whose valuable guidance that the project came out successfully after each stage.

We feel it our responsibility to thank **Mrs. D. Radha Rani** and **Mr. Vijay Babu,** for helping in all required aspects during the course.

We thank all our **Faculty** members and **Programmers** of Department of Computer Science and Engineering who provided technical support to us in our academics throughout course.

Finally, we wish to express thanks to our family members for the love and affection overseas and forbearance and cheerful depositions, which are vital for sustaining effort, required for completing this work.

With Sincere regards,

Chava Lakshitha(171FA04384)

Chunduri Sravya(171fa04385)

Date:21 June, 2021

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

1. INTRODUCTION

We are living in the era of networks and internet. Internet has become an important part of our life. Almost everything, we perform through internet. Social networking, Business, Entertainment, Education, etc. Which leaves us more vulnerable towards attacks. Any actions that compromise the availability, confidentiality and integrity of a system. So, To stop potential threats an internet, intrusion detection system are used. The last decade has seen rapid advancements in machine learning techniques enabling automation and predictions in scales never imagined. The most common risk to a network's security is an intrusion such as brute force, denial of service or even an infiltration from within a network. With the changing patterns in network behaviour, it is necessary to switch to a dynamic approach to detect and prevent such intrusions. It can detect new types of intrusions but is very prone to false positive alarms. Hence, only one unsupervised algorithm K-means clustering is discussed ahead. To reduce the false positives, we can introduce a labelled dataset and build a supervised machine learning model by teaching it the difference between a normal and an attack packet in the network.

Four major classes of attacks are:

- DOS (Denial of Service)
- Probe.
- R2L (Remote to Local)
- U2R (User to Root)

1.1 ABOUT THE PROJECT

The rapid growth in the use of computer networks results in the issues of maintaining the network security. This requires the network administrators to adopt various types of Intrusion Detection Systems (IDS) that help in monitoring the network traffics for unauthorized and malicious intent. The main function of Intrusion Detection System is to protect the resources from threats. It analyzes and predicts the behaviors of users, and these behaviors will be considered as an attack or a normal behavior. In this project, we use the different machine learning approaches such as Support Vector Machine (SVM) and Random Forest with the NSL-KDD dataset to detect the intrusion effectively. We compared the test accuracies of these multiple classification models to identify the best model for performing network intrusion detection.

Keywords : Random Forest, SVM, Machine Learning, NSL-KDD dataset.

Intrusion Detection Systems

- A set of techniques used for detection of abnormal behavior on network.
- Based on the assumption that the behavior of intruder is different from that of a normal user.
- IDS is capable of detect malicious activities that can not be detected by conventional firewalls.
- It is passive alert system, i.e., it only detects attacks but not prevent it.
- Intrusion detection categories are:
  - Misuse/signature based Intrusion Detection System.
  - Anomaly based Intrusion Detection System.
- Dataset used in research work is NSL-KDD dataset.

Misuse Intrusion Detection System

- Presents the attack in the form of signatures and patterns.
- Patterns are then maintained as a database of attacks.
- These signatures/patterns are compared with the data received on the network.

Advantages:

- It produces very low false positives (which is not actually an intruder activity).
- Easy to develop.
- Requires less computational resources.

Disadvantages:

- Detect only those threats that are in its database.
- Can detect previously known attacks only.
- Updating is too much time consuming.

A simple example is antivirus systems.

Anomaly Intrusion Detection System

- Based on the statistical analysis.
- Detects attacks based on irregularities in the pattern w.r.t. the normal pattern.
- Creates a model of the normal behavior of the system.
- Then look for activities that differ from the create model.

Advantages:

- Anomaly IDS is capable of detecting new unknown threats.

Disadvantages:

- Time requires in training may be long.
- Increased problem of false positive alerts.

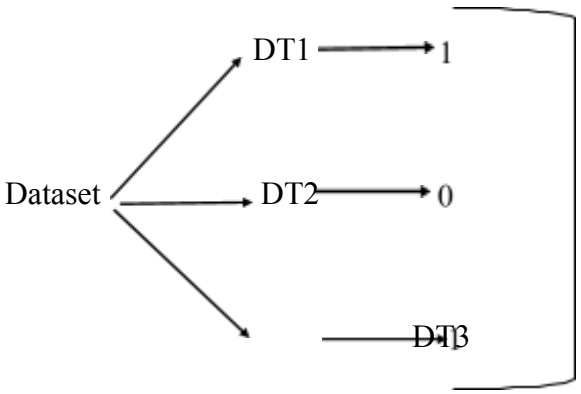
1.3 METHODOLOGY

Random Forest

Random forest algorithm is being used for the stock market prediction. Since it has been termed as one of the easiest to use and flexible machine learning algorithm, it gives good accuracy in the prediction. This is usually used in the classification tasks. Because of the high volatility in the stock market, the task of predicting is quite challenging.

The random forest algorithm represents an algorithm where it randomly selects different observations and features to build several decision tree and then takes the aggregate of the several decision trees outcomes. The data is split into partitions based on the questions on a label or an attribute.

The basic approach of the supervised learning model is to learn the patterns and relationships in the data from the training set and then reproduce them for the test data.



Majority shows 1, so

predicted value is 1.

Decision Tree

Classification is an instance of supervised learning where a set is analysed and categorized based on a common attribute. From the values or the data are given, classification draws some conclusion from the observed value. If more than one input is given then classification will try to predict one or more outcomes for the same.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

Decision Tree is one of the most commonly used, practical approaches for supervised learning. It can be used to solve both Regression and Classification tasks with the latter being put more into practical application.

Support vector machine

SVM is a supervised ML algorithm based on the idea of max-margin separation hyper-plane in *n*-dimensional feature space. It is used for the solution of both linear and nonlinear problems. For nonlinear problems, kernel functions are used. The idea is to first map a low dimensional input vector into a high dimensional feature space using the kernel function. Next, an optimal maximum marginal hyper-plane is obtained, which works as a decision boundary using the support vectors.[65, 66] For NIDS, the SVM algorithm can be used to enhance its efficiency and accuracy by correctly predicting the normal and malicious classes.

AutoEncoder

AutoEncoder (AE) is a popular DL technique that belongs to the family of unsupervised neural networks. It works on the idea of matching the output as close to input as possible by learning the best features. It contains input and output layers of the same dimension, while the dimensions of the hidden layers are normally smaller than the input layer. AE is symmetric and works in Encoder-Decoder fashion. Different variants of AE are Stacked AE, Sparse AE, and Variational AE.

Shone et al proposed an IDS based on deep AE and ML technique RF. To make the model efficient in terms of computational and time, only the encoder part of AE is utilized to make it work in a non-symmetric fashion. Two non-symmetric deep AEs, with three hidden layers each, are arranged in a stacked manner. RF was used for classification. Experiments were performed for multiclass classification scenarios using KDD Cup '99 and NSL-KDD datasets. The proposed method showed their efficiency compare to Deep Belief Network (DBN) used in Reference in terms of detection accuracy and reduced training time. But the model showed inefficiency for detecting R2L and U2R attacks due to lack of data for training the model.

Yan et al proposed an IDS using stacked sparse autoencoder (SSAE) and SVM. The SSAE was used as the feature extraction method and SVM as a classifier. Binary-class and multi-class classification problem is considered for conducting experiments. The results showed the proposed model superiority in performance comparing different feature selection, ML, and DL methods using the NSL-KDD dataset. Although, the model achieves reasonable detection rates for U2R and R2L attacks but it is still less comparing the other classes of the dataset.

A-Qatf et al also proposed a similar idea of self-taught learning based on sparse AE and SVM. To validate their performance, they performed experiments on the proposed model considering the NSL-KDD dataset. The results showed improved overall performance comparing other DL and ML models. But the proposed methodology performance in R2L and U2R class is not discussed.

Papmartizivanous et al[100] proposed an autonomous misuse detection system by combining the advantages of self-taught learning[101] and MAPE-K frameworks. They used sparse AE for the unsupervised learning algorithm to learn useful features while performing the Plan activity within the MAPE-K Framework. Experiments performed using the KDD Cup'99 and NSL-KDD datasets. The main drawback is the lack of detection accuracy for U2R and R2L attack classes.

Khan et al proposed an efficient two-stage model based on deep stacked AE. The initial stage classified the dataset into the attack and normal classes with probability values. These probability scores are then used as an additional feature and are input to the final decision stage for normal and multiclass attack classification. The performance of the proposed model was tested using KDD Cup'99 and UNSWNB15 datasets. To reduce the problems due to class imbalance of the datasets, a different methodology was adopted for both datasets. For KDD Cup'99, the down sampling was performed to remove repeated records. While, to balance the distribution of records in UNSWNB15, up sampling of the dataset was performed using SMOTE. This pre-processing of the dataset dramatically improves the DR efficiency of attack class with lower training instances.

Malaiya et al proposed different IDS models based on fully connected networks, Variational AE, and Sequence-to-Sequence (Seq2Seq) structures, respectively. These models were examined for different datasets NSL-KDD, KyotoHoneypot, UNSW-NB15, IDS2017, and MAWILab traces. Results showed that the Seq2Seq model constructed using two RNNs performed the best comparing other models in terms of detection accuracy across all the datasets.

Yang et al proposed a model for ID based on the supervised adversarial variational AE with regularization and DNN (SAVAER-DNN). The performance of the model was tested using benchmark data NSL-KDD and UNSW-NB15. Experimental results confirm the model's effectiveness in detecting low frequency and new attacks.

Andresini et al incorporated the idea of AE to proposed a multistage model involving the ID convolution layer and two stacked fully connected layers. In the initial unsupervised stage, two AEs were trained separately using Normal and Attack flows to reconstruct the samples again. In the supervised stage, these new reconstructed samples are used to build a new augmented dataset that is used as input to a 1D-CNN. Then the output of this convolution layer is flattened and fed to fully connected layers, and lastly, a soft max layer classifies the dataset. Experiments were performed on the KDD Cup'99, UNSWNB15, and CICIDS2017 datasets and the proposed methodology achieves superior performance comparing different DL models. They have not shown how the minority classes perform using this methodology. The second drawback is that it does not provide any information on the characteristics of the attack.

CHAPTER-2

MACHINE LEARNING APPROACH

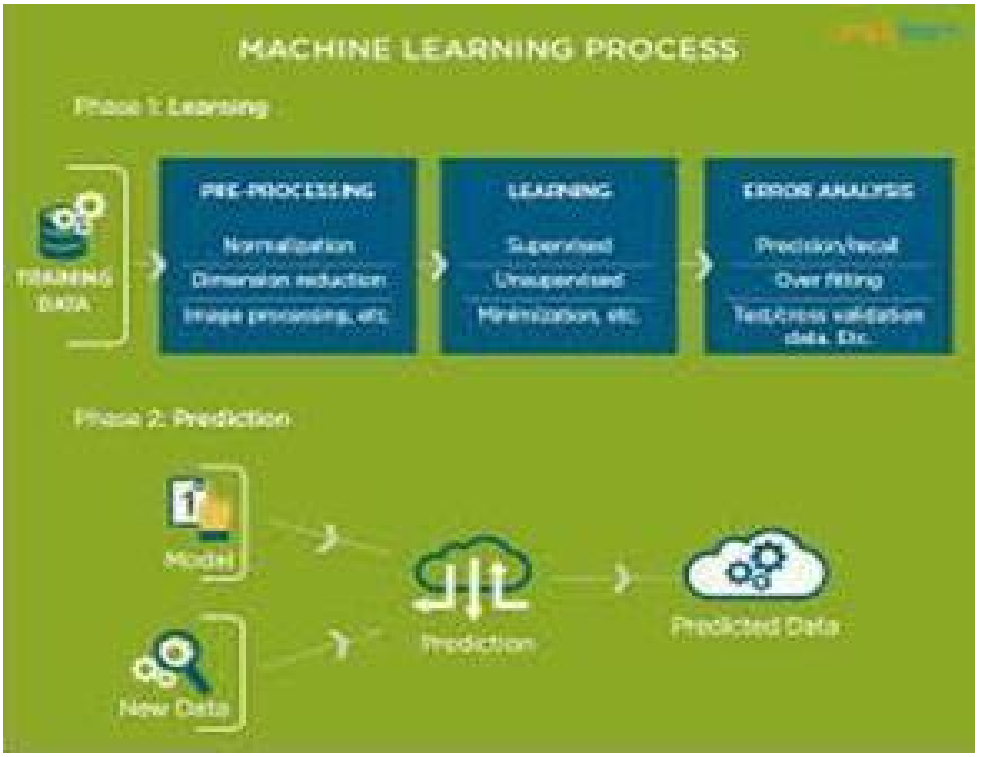
INTRODUCTION

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer system order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

IMPORTANCE OF MACHINE LEARNING

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world. Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries. With the constant evolution of the field, there has been a subsequent rise in the uses, demands and importance of machine learning.

Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques. The process flow depicted here represents how machine learning works.



2.1The Process Flow

USES OF MACHINE LEARNING

To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analy zing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results.

CHAPTER-3

TYPES OF LEARNING ALGORITHMS

3. TYPES OF LEARNING ALGORITHMS

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

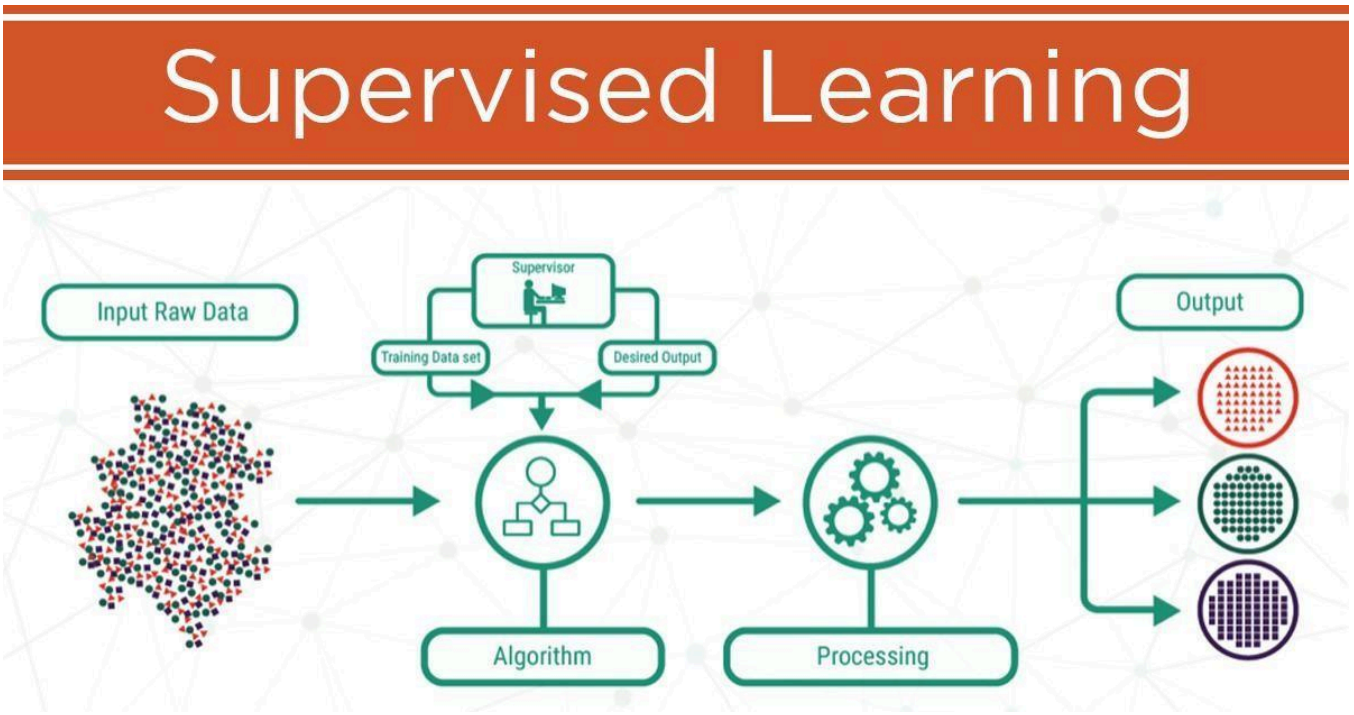
Supervised Learning

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

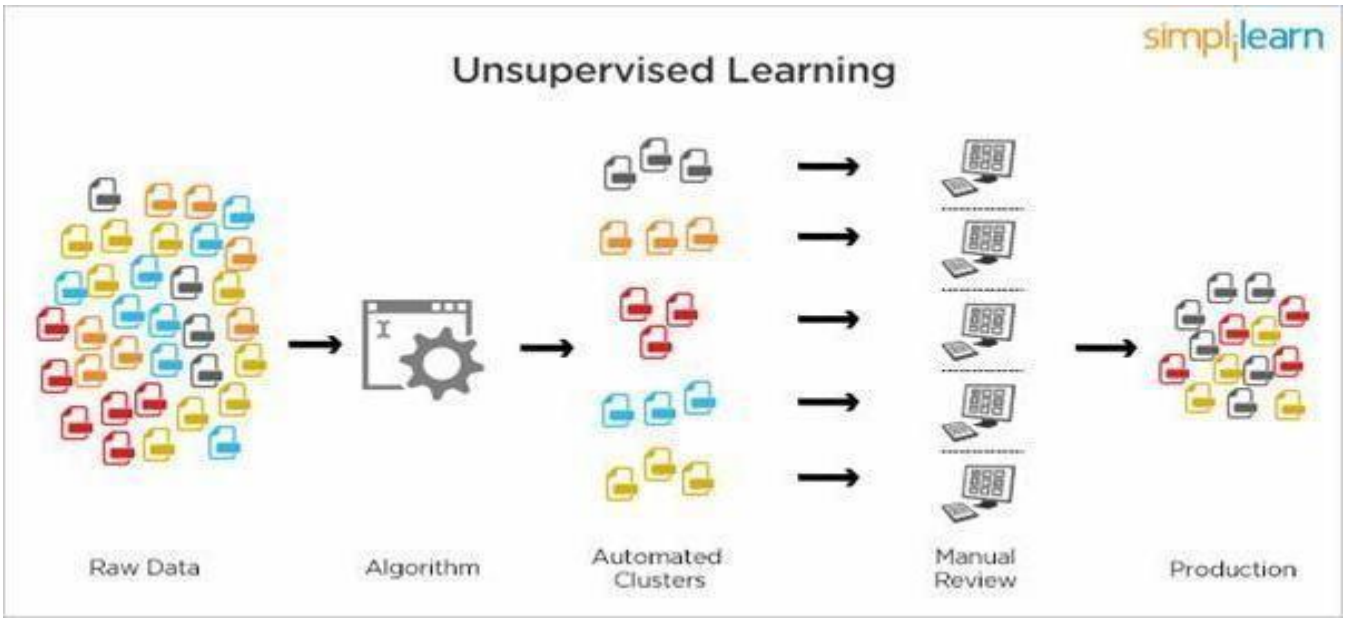
Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.



3.1Supervised Learning

Unsupervised Learning

When an algorithm learns from pla in examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

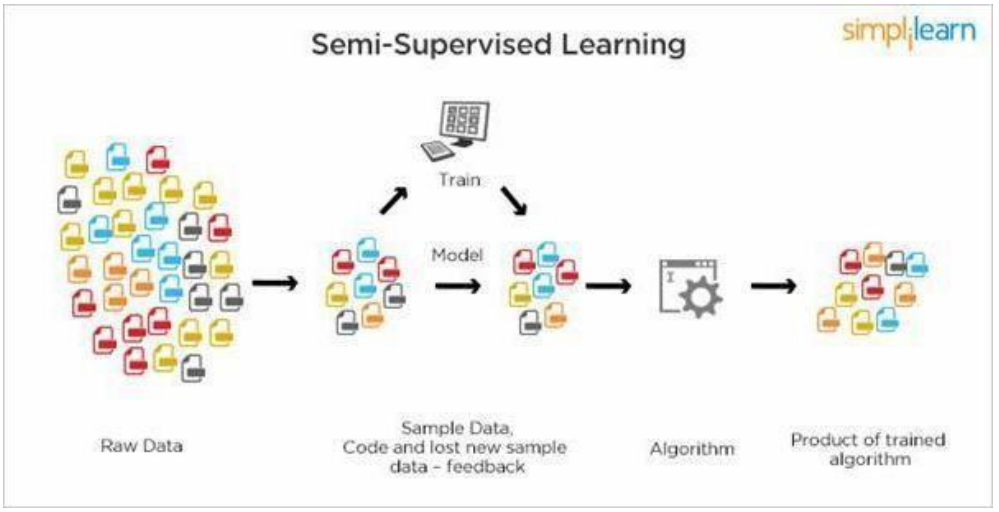


3.2 Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbour mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

Semi Supervised Learning

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.



Semi Supervised Learning

CHAPTER-4

SOFTWARE REQUIREMENT ANALYSIS (SRS)

4. Software Requirement Analysis (SRS)

Software Environment

Operating System	- Microsoft Windows 2000/XP
Scripting language	- Python, machine learning
Tool	- Jupyter Notebook, Py charm

Hardware Environment

RAM	- 4 GB
Processor	- Intel Pentium 500MHz
Floppy Disk	- 1.44 MB
Monitor	- Color Monitor (256 colors)

CHAPTER-5

ABOUT PYTHON

5. ABOUT PYTHON

Basic programming language used for machine learning is : PYTHON

INTRODUCTION TO PYHTON

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code with in objects.

HISTORY OF PYTHON

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

FEATURES OF PYTHON

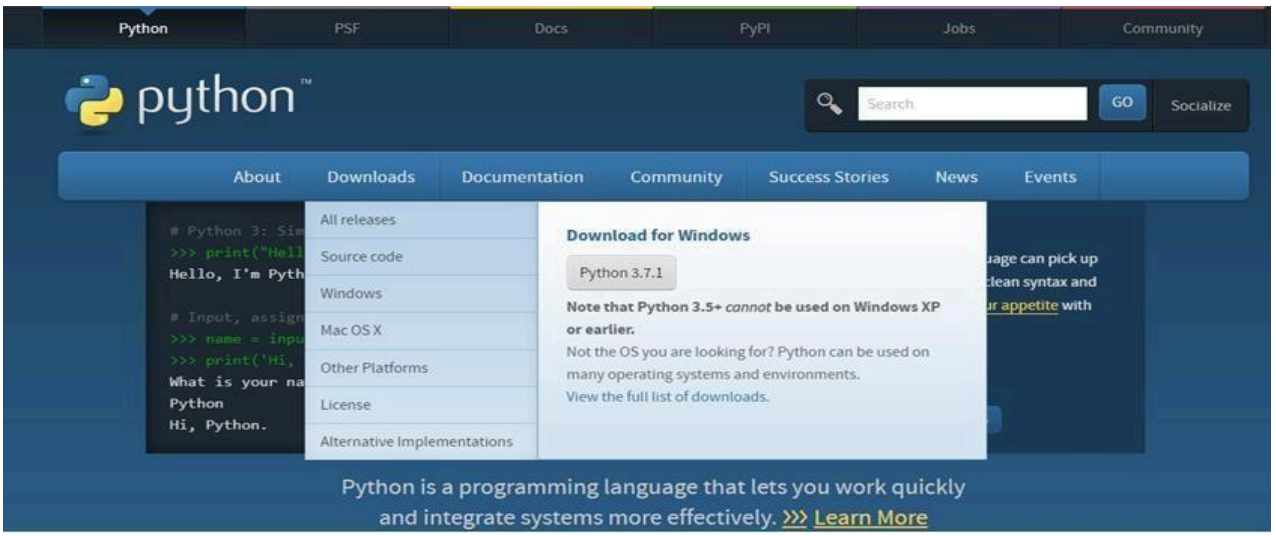
- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax , This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh.

HOW TO SETUP PYTHON

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

Installation (using python IDLE)

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.



5.5.1 Python Download

Installation (using Anaconda)

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

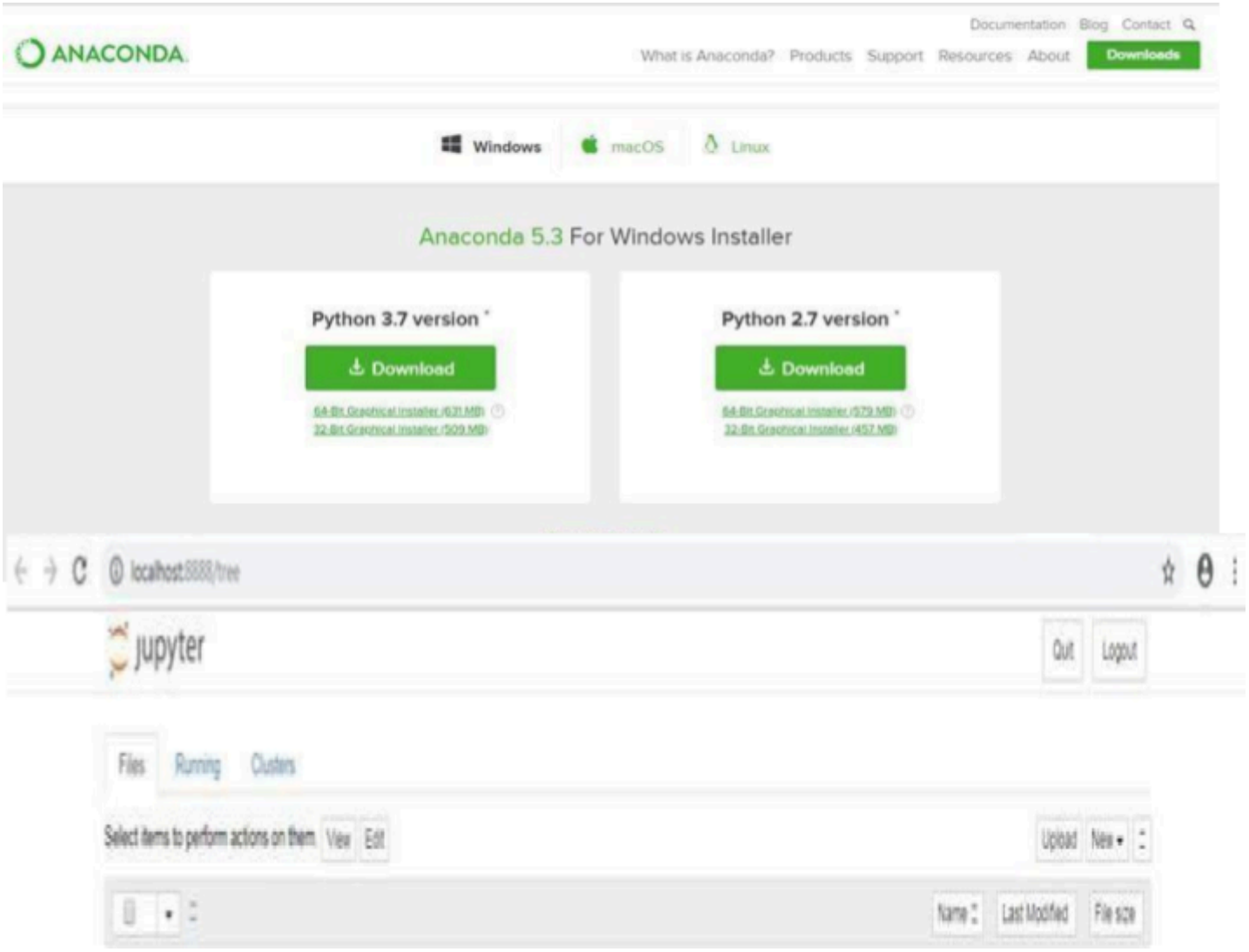
In WINDOWS

Step 1: Open Anaconda.com/downloads in web browser.

Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer) Step 3: select installation type (all users).

Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish

Step 5: Open jupyter notebook (it opens in default browser).



5.6.2 Jupyter Notebook

PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This meansthat when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types–
    1. Numbers
    2. Strings
    3. Lists
    4. Tuples
    5. Dictionary

Python Numbers

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they
- can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Python Lists

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end-1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Python Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed with in parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have
  no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

Python Dictionary

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes
- found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessedusing square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

PYTHON USING OOP's CONCEPTS

Class

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

Defining a Class

- We define a class in a very similar way how we define a function.
- Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

5.8.1. Defining a class

__Init__method in Class

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two under scores : init ().

Python offers ready-made framework for performing data mining tasks on large volumes of data effectively in lesser time. It includes several implementations achieved through algorithms such as linear regression, logistic regression, Naïve Bayes, k-means, K nearest neighbor, and Random Forest.

CHAPTER – 6
SYSTEM ANALYSIS

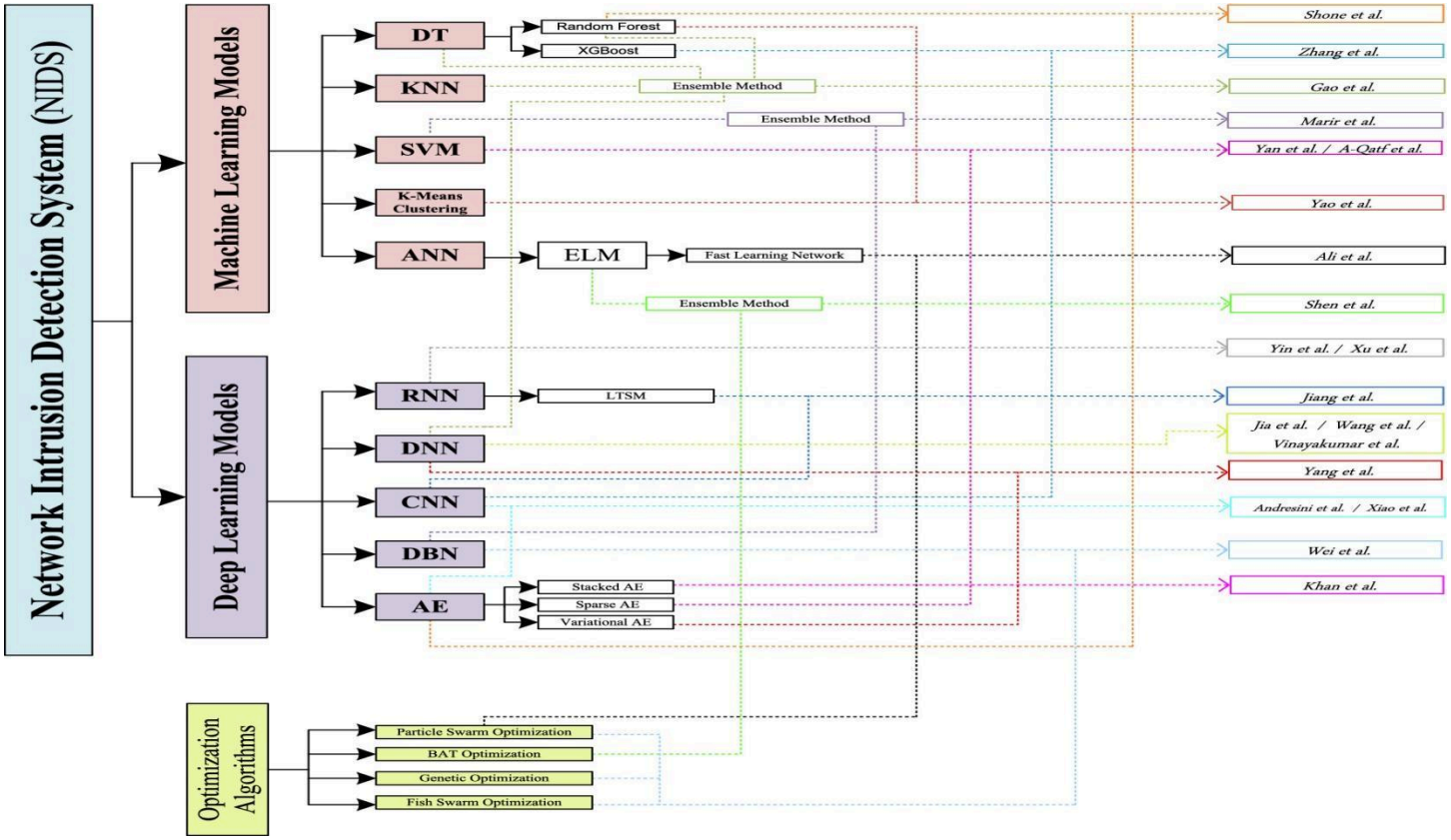
6. SYSTEM ANALYSIS

Steps for building model

The Intrusion Detection of a Network is built in 4 steps.

- Data Pre-processing
- Feature Selection
- Training Models
- Result Analysis
- Attack Prediction

In the following section, we provide an extensive overview of widely used ML and DL algorithms for NIDS systems. Further, figure highlights the taxonomy of recent ML- and DL-based techniques used for NIDS. ML is a subset of AI that includes all the methods and algorithms which enable the machines to learn automatically using mathematical models in order to extract useful information from the large datasets.[13, 55] The most common ML (also called Shallow Learning) algorithms used for IDS are Decision Tree, *K*-Nearest Neighbor (KNN), Artificial Neural Network (ANN), Support Vector Machine (SVM), *K*-Mean Clustering, Fast Learning Network, and Ensemble Methods.



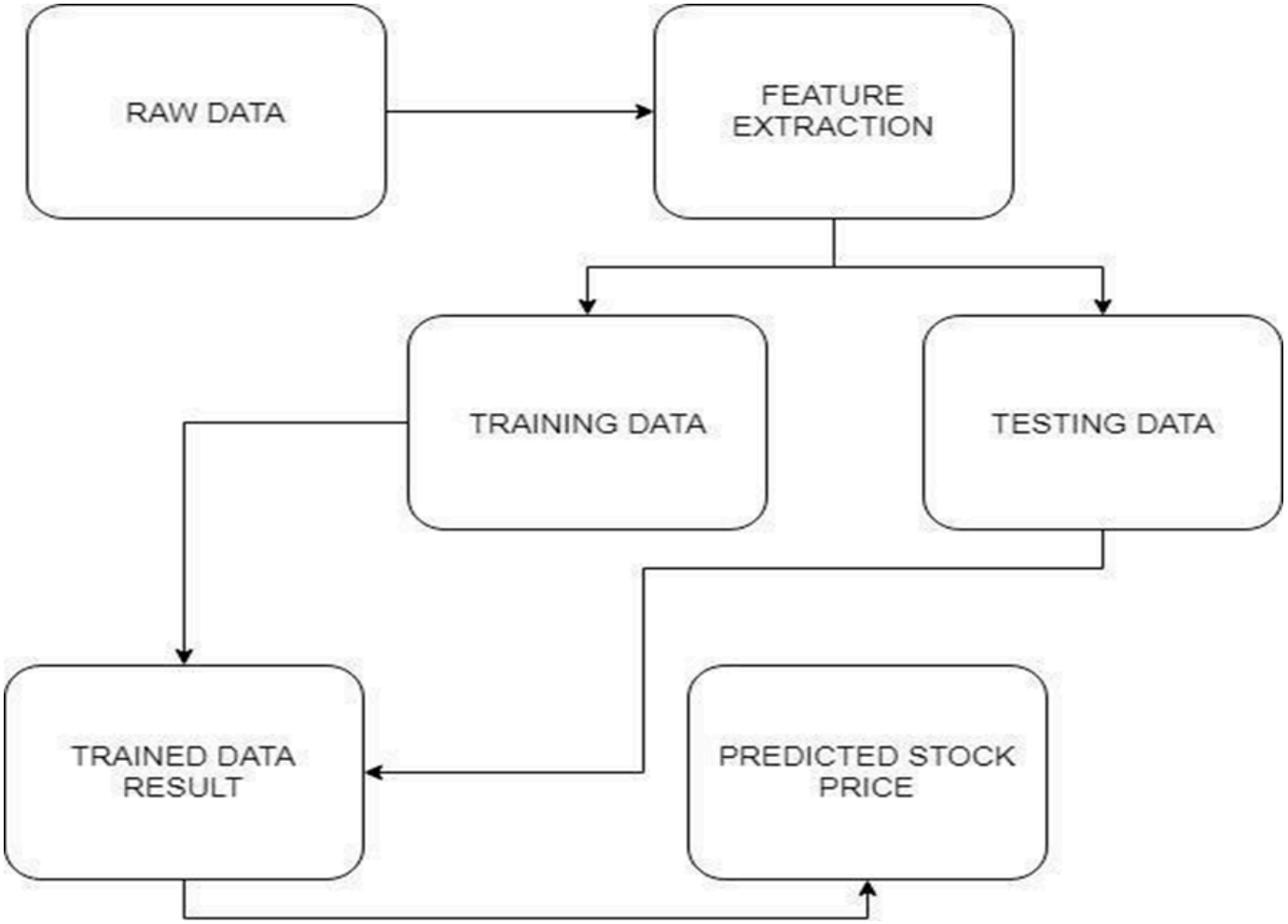
1.1 Block Diagram

CHAPTER-7

SYSTEM DESIGN

7. SYSTEM DESIGN

Architecture



7.1.1. Architecture

Module Identification

The various modules of the project would be divided into the segments as described.

1. Data Collection :

   Data collection is a very basic module and the initial step towards the project. It generally deals with the collection of the right dataset. Our data mainly consists of the details of network and attack class. Initially, we will be analysing the Kaggle dataset and according to the accuracy, we will be using the model with the data to analyse the predictions accurately.

2. Pre Processing :

   Data pre-processing is a part of data mining, which involves transforming raw data into a more coherent format. Raw data is usually, inconsistent or incomplete and usually contains many errors. The data pre- processing involves checking out for missing values, looking for categorical values and finally do a feature scaling to limit the range of variables so that they can be compared on common environs.

3. Training the Machine :

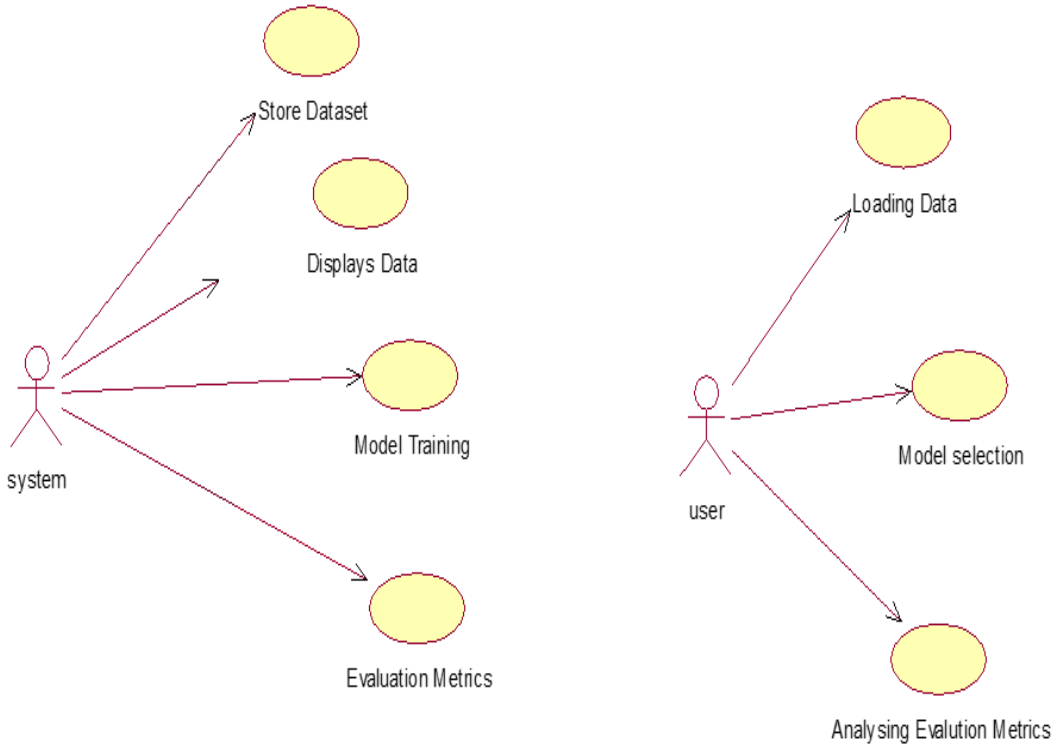
   Training the machine is similar to feeding the data to the algorithm to touch up the test data. The training sets are used to tune and fit the models. The test sets are untouched, as a model should not be judged based on unseen data. The training of the model includes cross-validation where we get a well- grounded approximate performance of the model using the training data.

   The idea behind the training of the model is that we some initial values with the dataset and then optimize the parameters which we want to in the model. This is kept on repetition until we get the optimal values. Thus, we take the predictions from the trained model on the inputs from the test dataset.

4. Data Scoring :

   The process of applying a predictive model to a set of data is referred to as scoring the data. The technique used to process the dataset is several boosting and bagging techniques. Random forest involves an ensemble method, which is usually used, for classification and as well as regression. Based on the learning models, we achieve interesting results.

7.3 Project Design Using UML Diagram



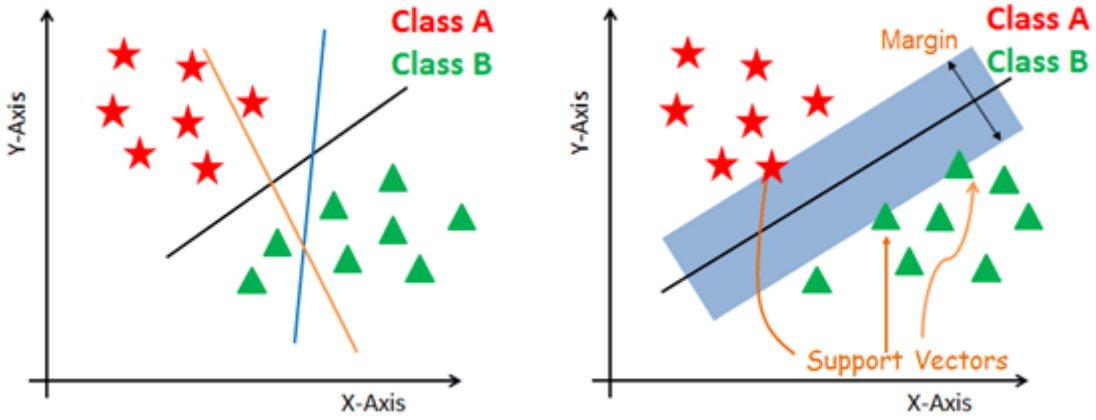
UML Diagram

CHAPTER-8

ALGORITHM

Support Vector Machine

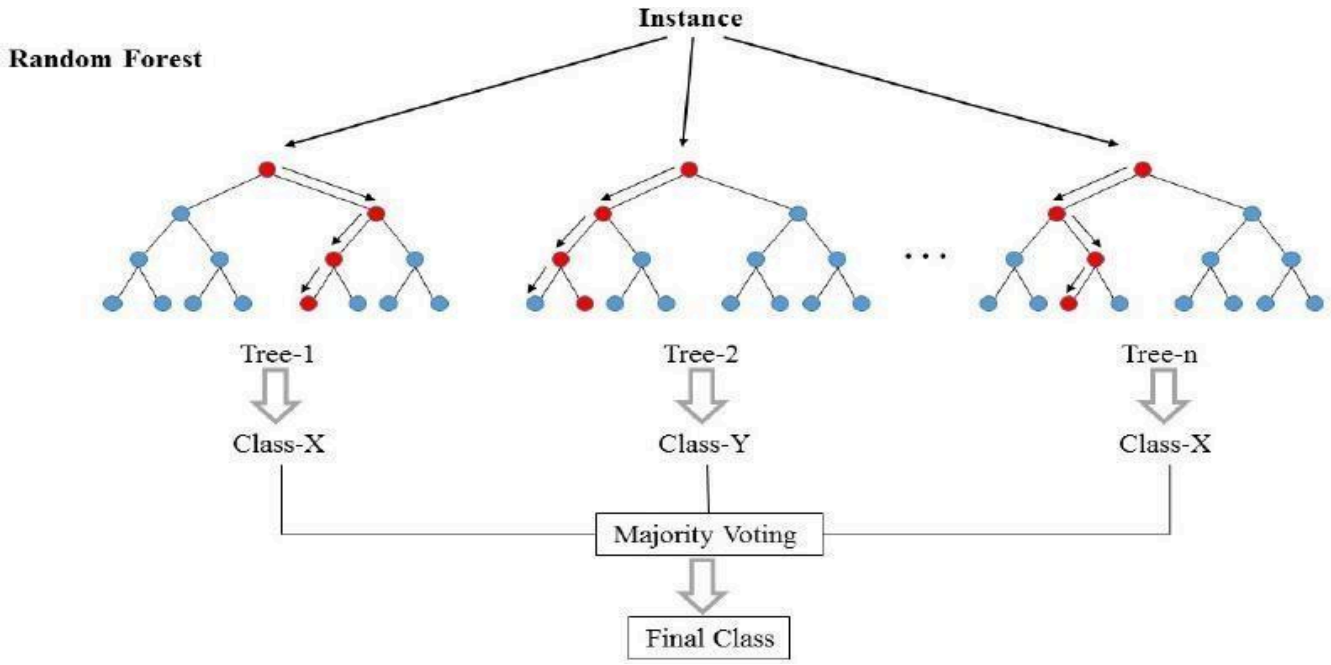
SVM is a supervised ML algorithm based on the idea of max-margin separation hyper-plane in *n*-dimensional feature space. It is used for the solution of both linear and nonlinear problems. For nonlinear problems, kernel functions are used. The idea is to first map a low dimensional input vector into a high dimensional feature space using the kernel function. Next, an optimal maximum marginal hyper-plane is obtained, which works as a decision boundary using the support vectors. For NIDS(Network Intrusion Detection System), the SVM algorithm can be used to enhance its efficiency and accuracy by correctly predicting the normal and malicious classes.



Random Forest Classsifier

Random forest is a bagging technique, which is a supervised learning algorithm, which uses ensemble learning method for classification and regression.

The basic ideas behind random forest is that it combines multiple decision trees to determine the final output, i.e., it builds multiple decision trees and merges their predictions together to get a more accurate and stable prediction.



Random Forest

There are 2 steps in random forest algorithm,

First step is random forest creation.

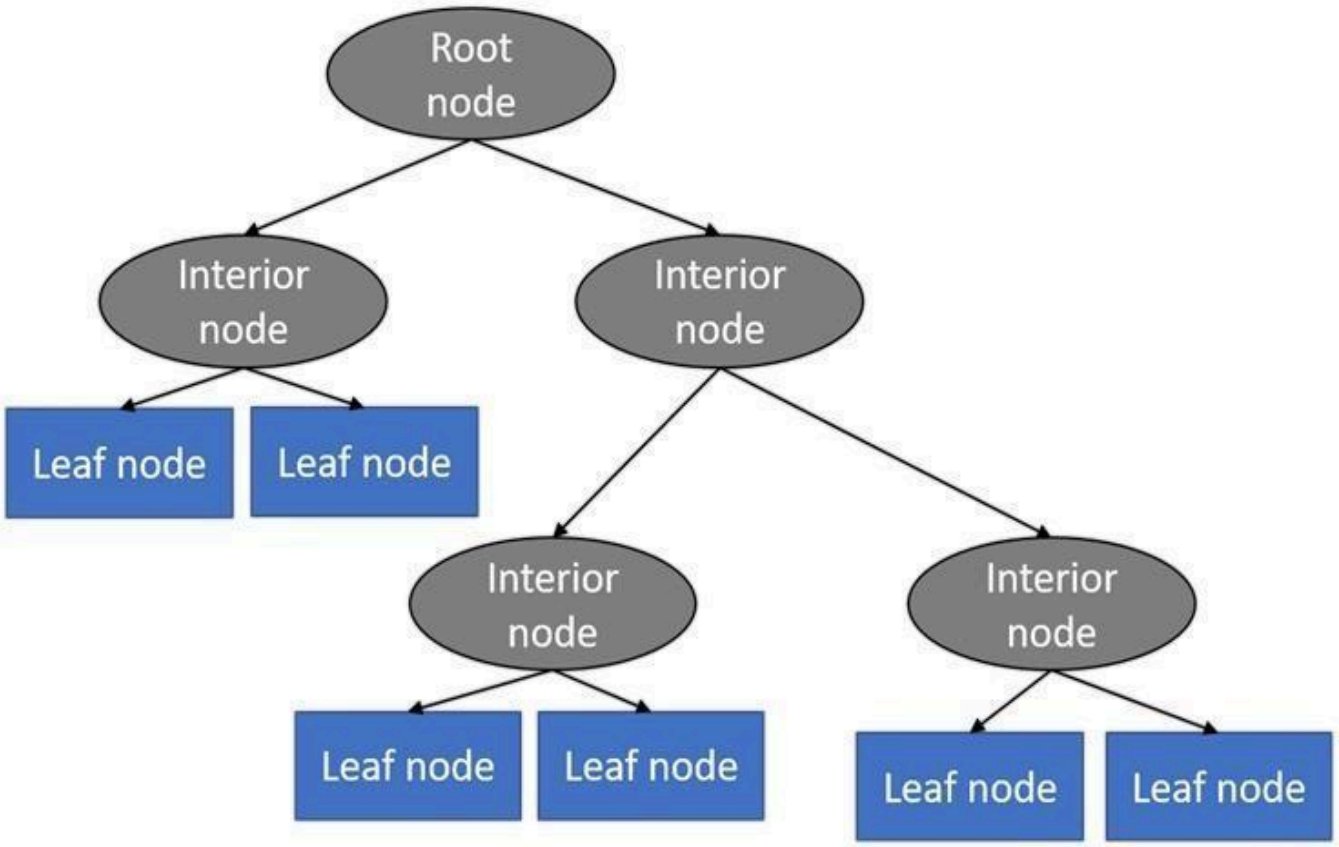
The other is to make a prediction from the random forest classifier created in the first step.

Decision Tree

Decision tree builds regression or classification models in the form of a tree structure. Decision Tree is one of the most commonly used, practical approaches for supervised learning.

It is a tree-structured classifier with three types of nodes.

- The Root Node is the initial node which represents the entire sample and may get split further into further nodes.
- The Interior Nodes represent the features of a data set and the branches represent the decision rules.
- Finally, the Leaf Nodes represent the outcome. This algorithm is very useful for solving decision-related problems.



Decision Tree

CHAPTER –9
SAMPLE CODE

9. SAMPLE CODE

# import relevant modules
import pip
pip.main(['install','seaborn'])
pip.main(['install','imblearn'])

```
# import relevant modules
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import imblearn

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.nan)
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

print("pandas : {0}".format(pd.__version__))
print("numpy : {0}".format(np.__version__))
print("matplotlib : {0}".format(matplotlib.__version__))
print("seaborn : {0}".format(sns.__version__))
print("sklearn : {0}".format(sklearn.__version__))
print("imblearn : {0}".format(imblearn.__version__))

# Dataset field names
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", "last_flag"]

# Load NSL_KDD train dataset
dfkdd_train = pd.read_table("~/NID/NSL_KDD_dataset/KDDTrain.txt", sep=",", names=datacols) #
change path to where the dataset is located.
dfkdd_train = dfkdd_train.iloc[:,:-1] # removes an unwanted extra field
dfkdd_train = dfkdd_train.fillna(0)



# Load NSL_KDD test dataset
dfkdd_test = pd.read_table("~/NID/NSL_KDD_dataset/KDDTest.txt", sep=",", names=datacols)
dfkdd_test = dfkdd_test.iloc[:,:-1]
dfkdd_test = dfkdd_test.fillna(0)
# View train data
dfkdd_train.head(3)
```

```
# train set dimension
print('Train set dimension: {} rows, {} columns'.format(dfkdd_train.shape[0],
dfkdd_train.shape[1]))

# View test data
dfkdd_test.head(3)

# test set dimension
print('Test set dimension: {} rows, {} columns'.format(dfkdd_test.shape[0],
dfkdd_test.shape[1]))



mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint':
'Probe','mscan': 'Probe',
        'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf':
'DoS','mailbomb': 'DoS',
        'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
        'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow': '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'
        }

# Apply attack class mappings to the dataset
dfkdd_train['attack_class'] = dfkdd_train['attack'].apply(mapping.get)
dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(mapping.get)

# Drop attack field from both train and test data
dfkdd_train.drop(['attack'], axis=1, inplace=True)
dfkdd_test.drop(['attack'], axis=1, inplace=True)

# View top 3 train data
dfkdd_train.head(3)

# Descriptive statistics
dfkdd_train.describe()
dfkdd_train['num_outbound_cmds'].value_counts()

dfkdd_test['num_outbound_cmds'].value_counts()

# 'num_outbound_cmds' field has all 0 values. Hence, it will be removed from both train
and test dataset since it is a redundant field.
dfkdd_train.drop(['num_outbound_cmds'], axis=1, inplace=True)
dfkdd_test.drop(['num_outbound_cmds'], axis=1, inplace=True)

# Attack Class Distribution
attack_class_freq_train = dfkdd_train[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_test = dfkdd_test[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_train['frequency_percent_train'] = round((100 * attack_class_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)
attack_class_dist
```

```
# Attack class bar plot
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);

dfkdd_train.head()

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance
cols = dfkdd_train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(dfkdd_train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(dfkdd_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)


from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

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

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

testcat = cattest.apply(encoder.fit_transform)

# separate target column from encoded data
enctrain = traincat.drop(['attack_class'], axis=1)
enctest = testcat.drop(['attack_class'], axis=1)

cat_Ytrain = traincat[['attack_class']].copy()
cat_Ytest = testcat[['attack_class']].copy()

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# define columns and extract encoded train set for sampling
sc_traindf = dfkdd_train.select_dtypes(include=['float64','int64'])
refclasscol = pd.concat([sc_traindf, enctrain], axis=1).columns
refclass = np.concatenate((sc_train, enctrain.values), axis=1)
X = refclass

# reshape target column to 1D array shape
c, r = cat_Ytest.values.shape
y_test = cat_Ytest.values.reshape(c,)


c, r = cat_Ytrain.values.shape
y = cat_Ytrain.values.reshape(c,)

# apply the random over-sampling
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_sample(X, y)
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
Original dataset shape Counter({1: 67343, 0: 45927, 2: 11656, 3: 995, 4: 52})
Resampled dataset shape Counter({0: 67343, 1: 67343, 2: 67343, 3: 67343, 4: 67343})
```

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

# fit random forest classifier on the training set
rfc.fit(X_res, y_res);
# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':refclasscol,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();

from sklearn.feature_selection import RFE
import itertools
rfc = RandomForestClassifier()

# create the RFE model and select 10 attributes
rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_res, y_res)

# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), refclasscol)]
selected_features = [v for i, v in feature_map if i==True]

selected_feature

# define columns to new dataframe
newcol = list(refclasscol)
newcol.append('attack_class')

# add a dimension to target
new_y_res = y_res[:, np.newaxis]

# create a dataframe from sampled data
res_arr = np.concatenate((X_res, new_y_res), axis=1)
res_df = pd.DataFrame(res_arr, columns = newcol)


# create test dataframe
reftest = pd.concat([sc_testdf, testcat], axis=1)
reftest['attack_class'] = reftest['attack_class'].astype(np.float64)
reftest['protocol_type'] = reftest['protocol_type'].astype(np.float64)
reftest['flag'] = reftest['flag'].astype(np.float64)
reftest['service'] = reftest['service'].astype(np.float64)

res_df.shape
reftest.shape

from collections import defaultdict
classdict = defaultdict(list)

# create two-target classes (normal class and an attack class)
attacklist = [('DoS', 0.0), ('Probe', 2.0), ('R2L', 3.0), ('U2R', 4.0)]
normalclass = [('Normal', 1.0)]

def create_classdict():
    '''This function subdivides train and test dataset into two-class attack labels'''
    for j, k in normalclass:
        for i, v in attacklist:
            restrain_set = res_df.loc[(res_df['attack_class'] == k) |
(res_df['attack_class'] == v)]
            classdict[j +'_' + i].append(restrain_set)
```

```
            # test labels
            reftest_set = reftest.loc[(reftest['attack_class'] == k) |
(reftest['attack_class'] == v)]
            classdict[j +'_' + i].append(reftest_set)

create_classdict()

for k, v in classdict.items():
    k

pretrain = classdict['Normal_DoS'][0]
pretest = classdict['Normal_DoS'][1]
grpclass = 'Normal_DoS'


from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()

Xresdf = pretrain
newtest = pretest

Xresdfnew = Xresdf[selected_features]
Xresdfnum = Xresdfnew.drop(['service'], axis=1)
Xresdfcat = Xresdfnew[['service']].copy()

Xtest_features = newtest[selected_features]
Xtestdfnum = Xtest_features.drop(['service'], axis=1)
Xtestcat = Xtest_features[['service']].copy()


# Fit train data
enc.fit(Xresdfcat)

# Transform train data
X_train_1hotenc = enc.transform(Xresdfcat).toarray()

# Transform test data
X_test_1hotenc = enc.transform(Xtestcat).toarray()

X_train = np.concatenate((Xresdfnum.values, X_train_1hotenc), axis=1)
X_test = np.concatenate((Xtestdfnum.values, X_test_1hotenc), axis=1)

y_train = Xresdf[['attack_class']].copy()
c, r = y_train.values.shape
Y_train = y_train.values.reshape(c,)

y_test = newtest[['attack_class']].copy()
c, r = y_test.values.shape
Y_test = y_test.values.reshape(c,)

from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
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_train, Y_train);

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);
```

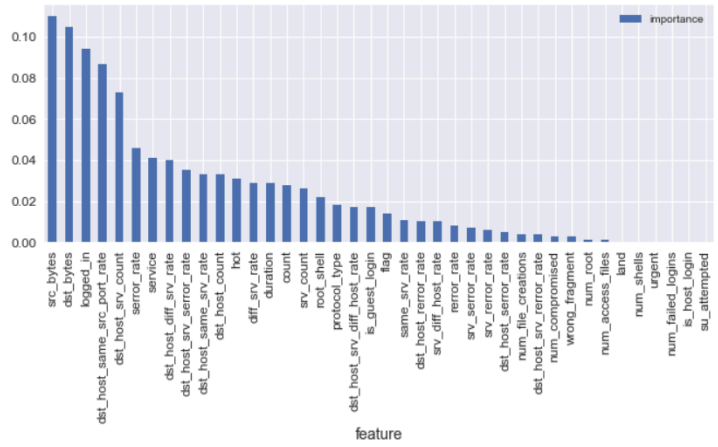
```
# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
DTC_Classifier.fit(X_train, Y_train);

# Train RandomForestClassifier Model
#RF_Classifier = RandomForestClassifier(criterion='entropy', n_jobs=-1, random_state=0)
#RF_Classifier.fit(X_train, Y_train);

# Train SVM Model
#SVC_Classifier = SVC(random_state=0)
#SVC_Classifier.fit(X_train, Y_train)

## Train Ensemble Model (This method combines all the individual models above except
RandomForest)
#combined_model = [('Naive Baye Classifier', BNB_Classifier),
#                  ('Decision Tree Classifier', DTC_Classifier),
#                  ('KNeighborsClassifier', KNN_Classifier),
#                  ('LogisticRegression', LGR_Classifier)
#                 ]
#VotingClassifier =  VotingClassifier(estimators = combined_model,voting = 'soft',
n_jobs=-1)
#VotingClassifier.fit(X_train, Y_train);


from sklearn import metrics
models = []
#models.append(('SVM Classifier', SVC_Classifier))
models.append(('Naive Baye Classifier', BNB_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_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} {} Model Evaluation
==============================='.format(grpclass, 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()




for i, v in models:
```

```
    accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
    confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
    classification = metrics.classification_report(Y_test, v.predict(X_test))
    print()
    print('============================== {} {} Model Test Results
==============================='.format(grpclass, i))
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

CHAPTER –10
RESULT ANALYSIS

Random Forest Algorithm:

In [19]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(X_res, y_res);
# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':refclasscol,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();
```



Random Forest

Evaluate Models

```
from sklearn import metrics

models = []
#models.append(('SVM Classifier', SVC_Classifier))
models.append(('Naive Baye Classifier', BNB_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_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} {} Model Evaluation ==============================='.format(grpclass, 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()
```

```
============================== Normal_DoS Naive Baye Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.973776072139

Model Accuracy:
 0.973768617377

Confusion matrix:
 [[65346  1997]
 [ 1536 65807]]

Classification report:
              precision    recall  f1-score   support

        0.0       0.98      0.97      0.97     67343
```

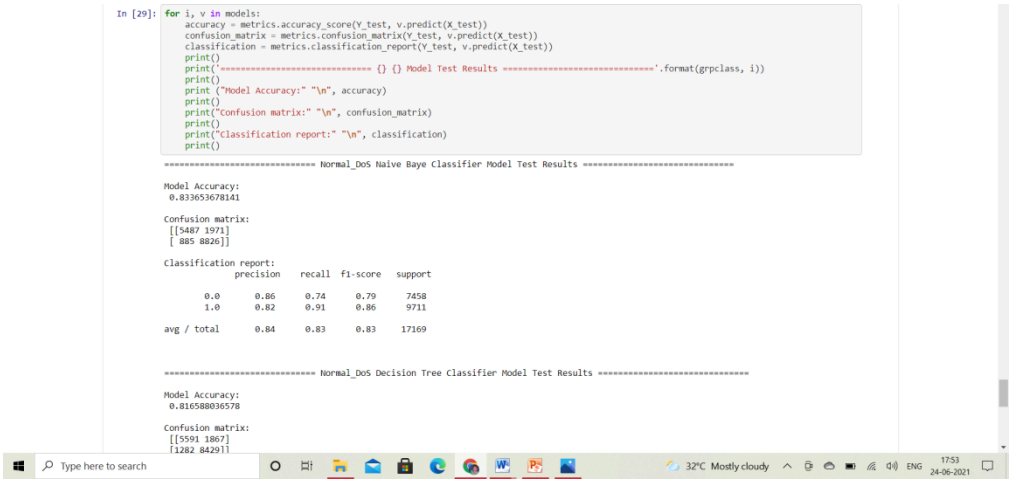
```
Classification report:
             precision    recall  f1-score   support

        0.0       0.98      0.97      0.97     67343
        1.0       0.97      0.98      0.97     67343

avg / total       0.97      0.97      0.97    134686


============================== Normal_DoS Decision Tree Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.999769836897

Model Accuracy:
 0.999948027263

Confusion matrix:
 [[67343     0]
 [    7 67336]]

Classification report:
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00     67343
        1.0       1.00      1.00      1.00     67343

avg / total       1.00      1.00      1.00    134686


============================== Normal_DoS KNeighborsClassifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.996569816347

Model Accuracy:
 0.99775774765

Confusion matrix:
 [[67287    56]
 [  246 67097]]
```



```
============================== Normal_DoS KNeighborsClassifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.996569816347

Model Accuracy:
 0.99775774765

Confusion matrix:
 [[67287    56]
 [  246 67097]]

Classification report:
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00     67343
        1.0       1.00      1.00      1.00     67343

avg / total       1.00      1.00      1.00    134686
```

```
============================== Normal_DoS LogisticRegression Model Evaluation ==============================

Cross Validation Mean Score:
 0.980822083372

Model Accuracy:
 0.980777512139

Confusion matrix:
 [[65527  1816]
 [  773 66570]]

Classification report:
             precision    recall  f1-score   support

        0.0       0.99      0.97      0.98     67343
        1.0       0.97      0.99      0.98     67343

avg / total       0.98      0.98      0.98    134686
```

Test Models



```
In [29]: for i, v in models:
             accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
             confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
             classification = metrics.classification_report(Y_test, v.predict(X_test))
             print()
             print('============================== {} {} Model Test Results ==============================='.format(grpclass, i))
             print()
             print ("Model Accuracy:" "\n", accuracy)
             print()
             print("Confusion matrix:" "\n", confusion_matrix)
             print()
             print("Classification report:" "\n", classification)
             print()

============================== Normal_DoS Naive Baye Classifier Model Test Results ==============================

Model Accuracy:
 0.833653678141

Confusion matrix:
 [[5487 1971]
 [ 885 8826]]

Classification report:
              precision    recall  f1-score   support

         0.0       0.86      0.74      0.79      7458
         1.0       0.82      0.91      0.86      9711

avg / total       0.84      0.83      0.83     17169


============================== Normal_DoS Decision Tree Classifier Model Test Results ==============================

Model Accuracy:
 0.816588036578

Confusion matrix:
 [[5591 1867]
 [1282 8429]]
```

```
        0.0       0.81      0.75      0.78      7458
        1.0       0.82      0.87      0.84      9711

avg / total       0.82      0.82      0.82     17169


============================== Normal_DoS KNeighborsClassifier Model Test Results ==============================

Model Accuracy:
 0.866620071058

Confusion matrix:
 [[5787 1671]
 [ 619 9092]]

Classification report:
              precision    recall  f1-score   support

        0.0       0.90      0.78      0.83      7458
        1.0       0.84      0.94      0.89      9711

avg / total       0.87      0.87      0.86     17169


============================== Normal_DoS LogisticRegression Model Test Results ==============================

Model Accuracy:
 0.842157376667

Confusion matrix:
 [[5963 1495]
 [1215 8496]]

Classification report:
              precision    recall  f1-score   support

        0.0       0.83      0.80      0.81      7458
        1.0       0.85      0.87      0.86      9711

avg / total       0.84      0.84      0.84     17169
```

output

CHAPTER-11

CONCLUSION

11.1. CONCLUSION

This paper provides an extensive review of the network intrusion detection mechanisms based on the ML and DL methods to provide the new researchers with the updated knowledge, recent trends, and progress of the field. A systematic approach is adopted for the selection of the relevant articles in the field of AI-based NIDS. Firstly, the concept of IDS and its different classification schemes is elaborated extensively based on the reviewed articles. Then the methodology of each article is discussed and the strengths and weaknesses of each are highlighted in terms of the intrusion detection capability and complexity of the model. Here we state that random forest and support vector machine perform better in detecting intrusion when compared to KNN and K Farthest Neighbour Algorithms.

CHAPTER-12
REFERENCES

12. REFERENCES

1. https://www.ijeat.org/wp-content/uploads/papers/v8i4/D6321048419.pdf
2. https://www.pantechsolutions.net/stock-market-prediction-using-machine-learning
3. https://ieeexplore.ieee.org/document/8212715
4. http://ijarcsse.com/docs/papers/Volume_7/1_January2017/V7I1-0112.pdf