DETECTION OF THREATS IN CLOUD DATA SHARING USING MACHINE LEARNING

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Information Technology

by

SOMYA BRIJESH SHASTRI

16BIT0261

Under the guidance of

Prof. Gunavathi C.

School Of Information Technology and Engineering

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

May, 2020

DECLARATION

I hereby declare that the thesis entitled "DETECTION OF THREATS IN CLOUD DATA SHARING USING MACHINE LEARNING" submitted by me, for the award of the degree of Bachelor of Technology in Information Technology to VIT, is a record of bonafide work carried out by me under the supervision of Prof. Gunavathi C.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 20 May, 2020 **Signature of the Candidate**

CERTIFICATE

This is to certify that the thesis entitled "DETECTION OF THREATS IN CLOUD DATA SHARING USING MACHINE LEARNING**"** submitted by SOMYA BRIJESH SHASTRI, 16BIT0261, School of Information Technology and Engineering VIT, for the award of the degree of Bachelor of Technology in Information Technology is a record of bonafide work carried out by him/her under my supervision during the period, 01. 12. 2018 to 30.04.2019, as per the VIT code of academic and research ethics

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Project report fulfils the requirements and regulations of VIT and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: **Signature of the Guide**

Internal Examiner **External Examiner**

Head of the Department

Information Technology

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Prof. Gunavathi C, Associate professor Grade 2, School of Information Technology and Engineering, Vellore Institute of Technology, for her constant guidance, continual encouragement, and understanding; more than all, he taught me patience in my endeavour. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine learning

I would like to express my gratitude to Honorable Chancellor, Dr. G. Viswanathan; esteemed Vice-Presidents, Mr. Sankar Viswanathan; Dr. Sekar Viswanathan and Mr. G. V. Selvam; respected Vice Chancellor, Dr. Anand A. Samuel; respected Pro-Vice Chancellor Dr. S. Narayanan and Dr Balakrushna Tripathy, School of Information Technology and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr Jasmine Norman, Head of Department of Information Technology, Associate Professor Sr., all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: **Name of the student**

EXECUTIVE SUMMARY

Majority of data is shared or transacted wirelessly and it is agreed that this increases the efficiency and saves a lot of time. One major concern is security; the data shared through cloud can be compromised and can be used for unknown reasons. There is a lot of sensitive data and the leakage of this data cannot be imagined. Therefore it is important to study and address these issues. These days more and more organizations are using cloud as a platform to share data and store large amounts of data due to its versatility. But the intruders pose a major threat to the security of the data as they can access the large amount of sensitive data. The project aims to study few of the methods used till date namely KNN, Naïve Bayes, Decision Tree and Logistic regression; compare, contrast and explain their pros and cons and finally conclude with behaviour and accuracy of machine learning techniques which will be implemented by training the model with KDD dataset.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

KDD	Knowledge Discovery and Data mining tools competition
ML	Machine Learning
RAM	Random Access Memory
KNN	K-Nearest Neighbour
DoS	Denial of Service
U2R	User to Root
R2	Remote to Local
RFE	Recursive Feature Elimination
AES	Advanced Encryption Standard
SHA	Secure Hashing Algorithm
TLS	Transport Later Security
PCA	Principal Component Analysis

1. INTRODUCTION

1.1 OBJECTIVE

The main objective of this project is:

- To study and analyze various methods used to detect and prevent the threats in cloud sharing.
- Compare, train and implement major machine learning algorithms, for accuracy, precision and recall and try to make modify the better one to aim for even higher accuracy.
- Try to come up with a reason as to why a certain model performed better than others.
- Implement various ML algorithms to see and compare their performances.

1.2 MOTIVATION

Majority of data is shared or transacted wirelessly and it is agreed that this increases the efficiency and saves a lot of time. One major concern is security. The data shared through cloud can be compromised and can be used for unknown reasons. There is a lot of sensitive data and the leakage of this data cannot be imagined. Therefore it is important to study and address these issues.

1.3 BACKGROUND

Our society has become dependent on technology over the last decade. Cloud computing is one such technology which is emerging fast. Use of various services like servers, software development platforms and storage over the internet has become easily accessible, where users are able to access applications and services whenever and wherever they want. These days more and more organizations are using cloud as a platform to share data and store large amounts of data due to its versatility. With pros, comes its cons as well, security and privacy are the main concern of the cloud services. The intruders pose a major threat to the security of the data as they can access the large amount of sensitive data. There are various types of cloud threats which may pose various problems for an organization and lead to severe damage.

2. PROJECT DESCRIPTION AND GOALS

Cloud computing technology is growing at a tremendous rate. It is the most vibrant technology of the future which enables the users to use the resources efficiently without the need to invest in the infrastructure. The users can store large amount of data and can manipulate from anywhere in the world. As the dependency of the companies to exchange the data wirelessly is increasing rapidly, the security emerges as a major concern. There are various types of intrusions depending upon variation in time, data and geographical location. The major kind of attacks the cloud computing suffers from are DDoS attack, probing, R2L, U2R, spoofing etc. There is a desperate need to protect this data from the intruders as the information stored is extremely confidential and sensitive, hence the data leakage cannot be afforded.

To ensure the security of the data many machine learning algorithms are used for intrusion detection.

- Study various methods used for intrusion detect and analyze their pros and cons.
- Implement and Compare different machine learning algorithms for following performance metrics accuracy, precision and recall and try to make modify the better one to get the higher accuracy.

3. LITERATURE SURVEY

In paper [1] the authors try to provide automatic data classification which has not yet been done. The paper uses KNN algorithm and upgrades it by hybridizing ensemble learning technique. Here, on the basis of the data security parameter the data is automatically classified. With the existing RSA algorithm HMAC function has been appended when using highly confidential data. The proposed system saves more time, is more accurate and more economical. KNN gave an accuracy of 65.5% whereas KNN with ensemble learning technique increased the efficiency resulting to 73.5%

The authors in paper [2] have developed a new hybrid approach the network intrusion problem. The new approach can approximate the intrusion scope threshold degree. For training the optimal features if the dataset are made available for the threshold degree calculation. Although the hybrid approach gave a better result when seen the accuracy and the time complexity of the system, but there were problems with the true and false negative rates. The approach was then refined after which better results were obtained.

Paper [3] suggests Random Forest classified as a better classifier for the NSL-KDD dataset when compared with other traditional classification algorithms. Feature selection is applied to select the best features. The results were compared with J48 algorithms result. The results obtained had high accuracy and low false alarm of an intrusion.

In paper [4] the authors propose a solution for combine login and detection and prevention of network intrusion. The paper uses snort as an Intrusion Prevention System (IPS) for the practical working. Splunk is used for logging of dropped packets. It integrates software engineering techniques with IPS. It provides a customizable and cost effective solution to small scale organizations. The paper also states its limitations. The proposed system cannot accommodate new upcoming attacks. The diversion of an intruding packet might cause harm to the system. The limitations of the paper are also the future work that can be done.

In paper [5] S. Singh and M. Bansal have used WEKA tool on NSL-KDD dataset to evaluate the performance of Multilayer perceptron, Logistic regression, Radial Base

Function and Voted perceptron and conclude which classification algorithm proves to be the best for the chosen dataset. The paper concludes by saying that multilayer feed forward neural network gives the best accuracy and lowest error rate for intrusion detection when compared with all the other algorithms. The paper further states that, as a future work, the authors will try to combine Multilayer perceptron and fuzzy inference rules which should give better performance.

Paper [6] discusses about Advanced Persistent Threats (APT) attacks which have become an important problem to be looked upon when considering network security. The paper uses Snort tool for the problem statement chosen. The system forms association rules from the network attack behaviour and apply snort rule and apply it to IDS. Result obtained after applying extended snort rule is better than the traditional snort rule ad gives a precision of 98.3% and 0% false alarm rate.

Paper [7] discusses the basic cloud computing concepts, features, security, threats and also the solutions of the threats. It also discusses cloud architecture framework, deployment models, technologies, and attacks. It also discusses about the available research topics related to cloud security. Paper [8] identifies security issues which would be helpful to the cloud service providers as well as users of cloud. The paper aims to provide solutions to minimize the recognized threats. The paper states that emphasis should be paid not only to performance but also to the quality of service.

Paper [9] uses deep learning methods for preparing IDS using recurrent neural network (RNN IDS) and study the performance of the model. It uses binary and multiclass classification. The results of the model were compared with other algorithms like J48, support vector machine, random forest, artificial neural network, and other machine learning methods studied till now.

In paper [10] the authors propose confidentiality based data encryption technique. Using variable algorithms and variable key sizes, the paper states that using the proposed method the confidentiality and integrity of the data is maintained and the overhead and processing time is kept minimum. Algorithms used are TLS, AES and SHA. Various combination of these and different sizes of keys are used which make the algorithm more efficient and secure. Data classification is done to identify the level of confidentiality needed for a particular data. For future works, authors suggest to consider other features which could increase the confidentiality and decrease the processing speed and also occupy less space.

Paper [11] by Buczak and Guven is an excellent survey paper giving a detailed study on the many research papers available online which have used ML and DM algorithms. The paper first discusses ML and DM methods available for cyber security and intrusion detetction. The complexity of the algorithms and the challenges faced were discussed and at the end some recommendations are given. The paper states that any algorithm cannot be judged based on one parameter. One has to consider many things white stating the effectiveness of any method, like curacy, precision, when the data was collected, when a research was conducted and so on. The biggest problem when considering solution for IDS is the unavailability of proper labeled data to the researchers. The paper discusses all the above mentioned points.

In paper [12] the authors used combined algorithm approach on NSL-KDD dataset. The paper achieves detection accuracy of 89.24% when random tree and NBTree algorithm is combined using sum rule scheme. The paper, using this data set and algorithms states that using sum rule scheme when we implement combine classifier approach it can give a better result rather than individually using the algorithms individually. The paper also states that it may not be possible that if we select two individually best performing classifiers, the overall result will be the highest.

Paper [13] focuses on the efficiency of Principal Component Analysis (PCA) on this problem statement, i.e. network intrusion detection. It also determines the reduction ratio and the ideal number of principal components required for intrusion detection. The paper also finds the impact noise has on the algorithm. The paper uses two data sets, KDD CUP and UNB ISCX and uses the same algorithms on both of them. The paper, at the end, finds that 10 principal components are required for best results. The paper also proves that presence of noise degrades the performance of the algorithm and when PCA is used, the accuracy of the data is more when the data is noise free

Paper [14] uses a hybrid intelligent approach where it uses a combination of classifiers and data filtering so as to makes the decision intelligently to increase the overall performance of the resultant mode. The paper uses along with 10- fold cross validation method and 2- class classification strategy so as to get the final output. The proposed method gives a low false alarm rate and a high detection rate. The paper concludes that the proposed hybridization of Ensembles of Balanced Nested Dichotomies for Multi-class Problems and random forest of 10 trees with 0.06 as the out-of-bag estimate results almost 0% false alarm rate and 100% intrusion detection rate, thereby making the approach the most efficient.

Table 1: Literature Survey

Sr. No.	**Paper name**	**Dataset used**	**Algorithm/Tool used**	**Performance/ Result**
1	Enhance Data Security In Cloud Computing Using Machine Learning And Hybrid Cryptography Techniques By Kiran And Dr Sandeep Sharma [1]	NSL-KDD Dataset	KNN with modified ensemble learning technique	Accuracy: 73.56%
2	Anomaly-based intrusion detection system through feature selectionanalysis and building hybrid efficient model by	NSL-KDD Dataset	J48, SVM,Naïve Bayes, and hybrid of J48, Meta Pagging, RandomTree, REPTree, AdaBoostM1,DecisionStump and NaiveBayes.	Accuracy: 99.7%

	Shadi Aljawarneh et. Al. [2]			
3	Random Forest Modeling for Network Intrusion Detection System by Nabila Farnaaz and Jabbar [3]	NSL-KDD Dataset	Random Forest	Accuracy: 99.67%
4	Real Time Intrusion Detection and Prevention System by Poonam Sinai Kenkre, Anusha Pai, and Louella Colaco [4]	Created own network and data	Snort and Splunk	Not mentioned
5	Improvement of Intrusion Detection System in Data Mining using Neural Network by Sahilpreet Singh Meenakshi Bansal [5]	NSL-KDD Dataset	Multilayer Perception, Radial Base Function, Logistic Regression and Voted Perception in WEKA data mining tool	Accuracy: 94.94%
6	Research of Snort Rule Extension and APT Detection Based on APT Network Behavior Analysis by Yan Cui et. Al. [6]	custom dataset	Wireshark, Snort IDS, Association rules	Snort rules extended in this paper can be applied to the APT IDS Accuracy: 92.8% and

				96.4% (for two different organizations)
7	A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks by CHUANLONG YIN et. Al. [9]	NSL-KDD Dataset	Recurrent Neural Network	Detection Rate (Accuracy): 97.09%
8	A Secure Cloud Computing Model based on Data Classification by Lo'ai Tawalbeh et.al. [10]	Did simulations	AES, SHA, TLS	Low processing time and high encryption
9	An effective combining classifier approach using tree algorithms for network intrusion detection by Kevric et. Al. [12]	NSL-KDD Dataset	A combination of random tree and NBTree algorithms	Accuracy:89.24 %
10	Dimensionality reduction using Principal Component Analysis for network intrusion	NSL-KDD Dataset	PCA	Accuracy: 99.9%

	detection by Vasan et.al.[13]			
11	A Hybrid Intelligent Approach for Network Intrusion Detection by Panda et. Al. [14]	NSL-KDD Dataset	Various combinations of J48, Random forest, PCA	Accuracy ranging from 90.7% to 99.9%
12	Cloud security issues and challenges: a survey by Ashish Singh and Kakali Chatterjee [7]	No dataset	Study of various issues and challenges, hence no tool used.	• Counter measures to address the security. • Features of cloud computing • Challenges faces in security • Threats, attacks, issues and solutions are tabulated.
13	Analysis and Countermeasures for Security and Privacy Issues in Cloud computing by Wanu et. al. [8]	No dataset	Study of various issues and challenges, hence no tool used.	• Identified security isseus • Comparative analysis of already present

				solutions • Topics where more research needs to be done
14	A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection by Buczak et. al. [11]	No dataset	No tool	• ML and DM methods used in cyber analytics • Complexity, challenges and recommendations on ML and DM algorithms used in betwork intrusion detection.

4. TECHNICAL SPECIFICATIONS

4.1 HARDWARE REQUIREMENTS

Device should be enabled with Internet. It should at least have a storage capacity of 1 TB and 8 GB RAM since the data involved is huge and it takes time to train and test the data.

4.2 SOFTWARE REQUIREMENTS

The user's browser should be HTML5 compatible for a satisfactory user experience. The system should have Tensorflow support with Python 3.7 and have all the various libraries installed. The project is done using Google Colaboratory, an online free Jupyter notebook environment, which takes care of all the software requirements and a few hardware also (RAM).

4.3 USER REQUIREMENTS AND PRODUCT SPECIFIC SYSTEM REQUIREMENTS

Cloud computing is usually described in one of two ways. Either based on the cloud location, or on the service the cloud is offering. Based on a cloud location, it can be classified as public, private, hybrid, community cloud. Based on a service that the cloud is offering, are speaking of either IaaS (Infrastructure-as- a-Service), PaaS (Platformas-a-Service), SaaS (Software-as- a-Service) or, Storage, Database, Information, Process, Application, Integration, Security, Management, Testing-as- a-service. Be it anything, nowadays most of the domains uses or fall under one or more of these categories and hence security and authenticity of the data shared must be ensured across all domain and enterprise

There is no user requirement besides having a basic idea of cloud computing to understand the algorithm and basic working of a UI to actually see and analyze the results.

The system requirements include a decent storage capacity with at least 8GB RAM and a GTX 1050 GPU because the data involved is pretty huge and the preprocessing and the cleaning of the data has to be quick to procure fast and efficient results.

4.4 NON FUNCTIONAL REQUIREMENTS

Performance

The system must be interactive and the delays involved must be less .So in every action-response of the system, there are no immediate delays.

Safety

Information transmission should be securely transmitted without any changes in information.

Reliability

As the system provides the right tools for discussion, problem solving it must be made sure that the system is reliable in its operations and for securing the sensitive details.

4.5 ENGINEERING STANDARD REQUIREMENTS

- Economic

  The algorithm is apt and economical and doesn't need anything more than a personal system. Research has been done and the data has already been trained. What is needed is to capture the new data and test whether there is a chance of any intrusion
- Social

  Security is a concern for every social organization and there is no social circle where the data is not confidential. It is essential to protect the data which is shared.
- Ethical

  It should be ensured that any intrusion is detected and try to give out a message to not indulge in any sort of unauthorized access
- Sustainability

Every effort has been done to sustain the given technology and try to mitigate the disadvantages of the booming data sharing innovation and techniques.

- Inspectability

  The algorithm has a decent accuracy which can be used to inspect any threats and intrusions in data which is confidential.

5. DESIGN APPROACH AND DETAILS

5.1 MODULE DESCRIPTION

The project work done till now and anticipated for future is divided into the following modules:

Module 1: Analyzing various models used till now to detect threats and intrusions in cloud

Study was performed to understand the problem statement in detail, the research done till now, the limitations in the topic and the future work to be performed. The algorithms were studied to see the practical limitations.

Module 2: Choosing the correct dataset

The project will use KDD dataset because it is the most accurate dataset on which a lot of revisions have been made and has been used for a lot of researches. The dataset has been described below in a separate section for better understanding.

Module 3: Pre-processing the dataset

The chosen dataset then has to be pre-processed and feature extraction has to be performed to choose the most significant features for better and faster results. The attack class distribution is to be analyzed.

Module 4: Implementation of ML algorithms

The various machine learning algorithms to be used in this project have to be implemented, tested and then analyzed. Accuracy, precision, recall and various other parameters will have to be taken into consideration to finally say which algorithms gives the best result for this dataset.

5.2 DATASET DESCRIPTION

The project uses a refined version of KDD dataset obtained from Knowledge Discovery and Data Mining Competition called NDL-KDD dataset. KDD has training set of dimension 125973 rows, 42 columns and testing set of dimensions 22544 rows, 42 columns. The columns include the important information exchanged in the network and have 41 features such as protocol type, service, flags, etc. KDD dataset gives us the unbiased network information and is used extensively in the detection of threats in cloud. Last column is about the type of attack the information is about. Using this dataset we predict the attack class each data entry belongs to. There are 4 attack classes and one normal class (indicating no attack). The 4 classes are: Probe, Dos, U2R, and R2L.

1. Denial of Service attack (DOS): This attack occurs when the valid users are prevented by an attacker from accessing the network. It does so by consuming the memory or the computer's resources. This technique makes the system unable of handling valid requests. Few examples of DOS attacks: 'neptune,' 'teardrop,' 'ping of death (pod),' 'back', 'land', 'mail bomb' and 'smurf'.

2. Users-to-root attack (U2R): This type of attack occurs when using a valid user account an attacker gets the access to the system. It finds the system's weakness and then is able to gain access to the root component of the system. Some examples of U2R attacks are 'buffer overflow', 'rootkit', 'load-module', and 'perl'.

3. Remote-to-local attack (R2L): This attack happens when a legitimate users account is used by an attacker, who does not have an account of himself, by locally accessing it by finding the vulnerabilities of the system. Some of the R2L attacks types include 'phf', 'warezclient', 'ftp write', 'imap', 'warezmaster','spy', 'multihop', and 'guess passwd'.

4. Probing attack (PROBE): This attack type takes place when an attacker avoids the security and gains data from the computers in the network. Some of the PROBE attacks include 'nmap', 'satan', 'ipsweep , and 'portsweep'.

5.3 SYSTEM ARCHITECTURE

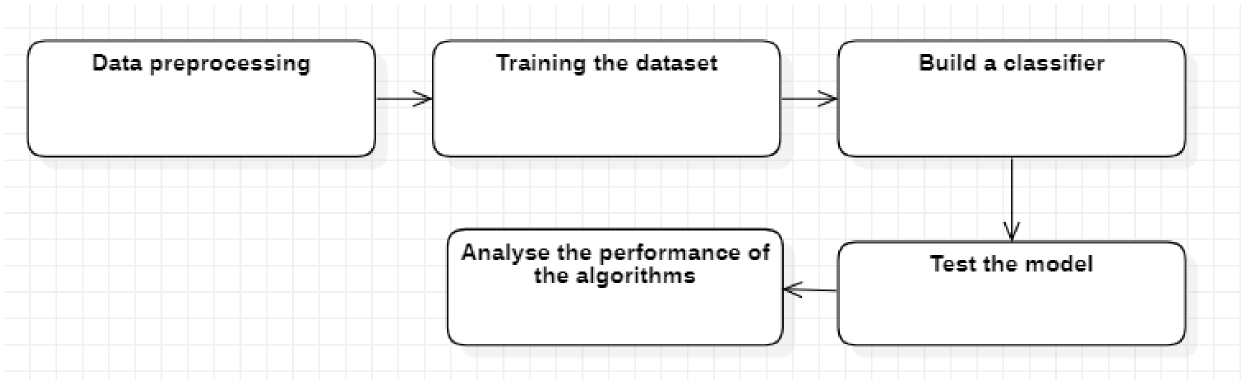

Fig. 1: System architecture

Dataset is pre-processed to remove noise and make it consistent. The training is done and a classified is built. Once the classifier is built we can test the new data and thus find the performance of the classifier and analyse.

5.4 USE CASE DIAGRAM

Actors

User: A server receives the request from the user and in turn responds by providing the required service.

Network: IP packets are carried by the network from source to destination.

IDS: IDS takes the packets from the network, analyses the packets.

System

Administrator System: IDS alerts the administrator of any suspicious activity or whenever there is detection of intrusion

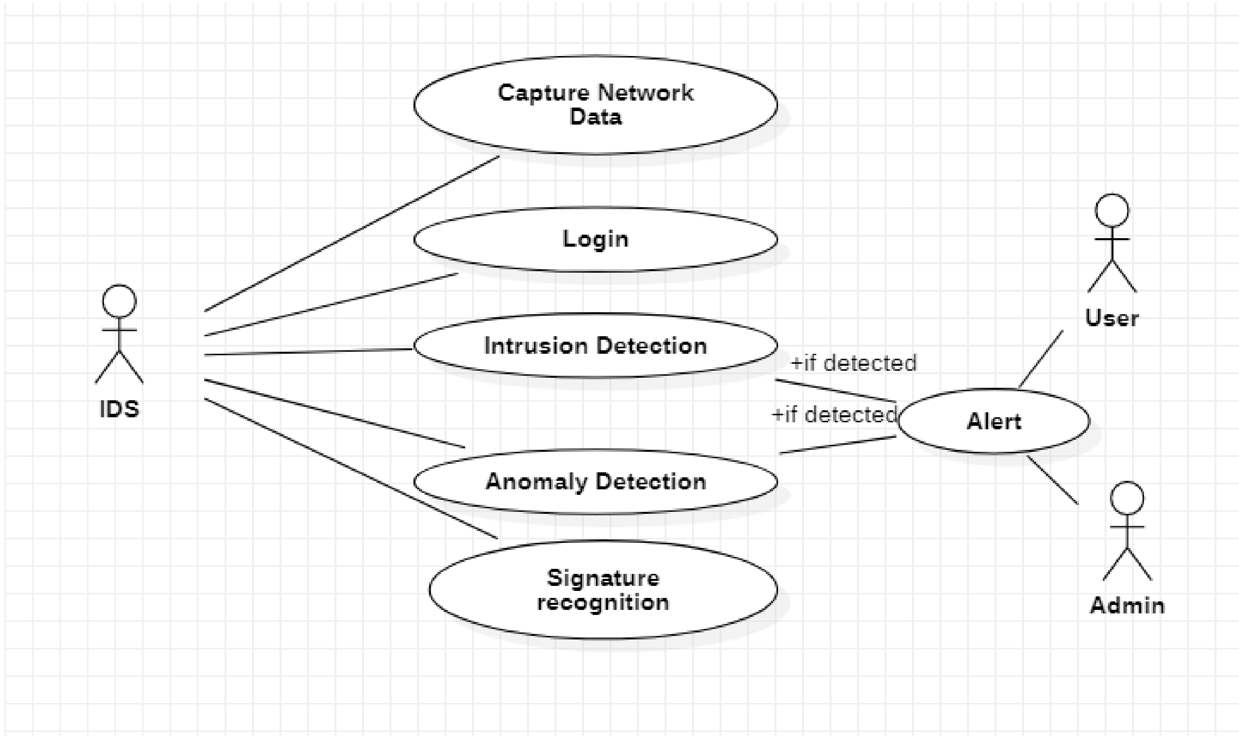

Fig 2. Use case diagram

5.5 METHODOLOGY

One the dataset is loaded and attacks are mapped to the respective attack classes a lot of pre-processing has to be done. The following steps are taken for pre-processing:

1. The mean of the dataset is made to be 0 and standard deviation to be 1.
2. Categorical data is converted into numerical data.
3. Imbalance in the dataset is removed by oversampling
4. Feature extraction is done and 10 most important features are selected.
5. Numerical data which was earlier categorical is one hot encoded so that the value of the attribute does not signify its importance.

After this 4 algorithms are applied to the dataset, KNN, Naive Bayes, Logistic regression and Decision tree. Performance is calculated by finding accuracy, f-measure, and other parameters. Given below is a flowchart of the process.

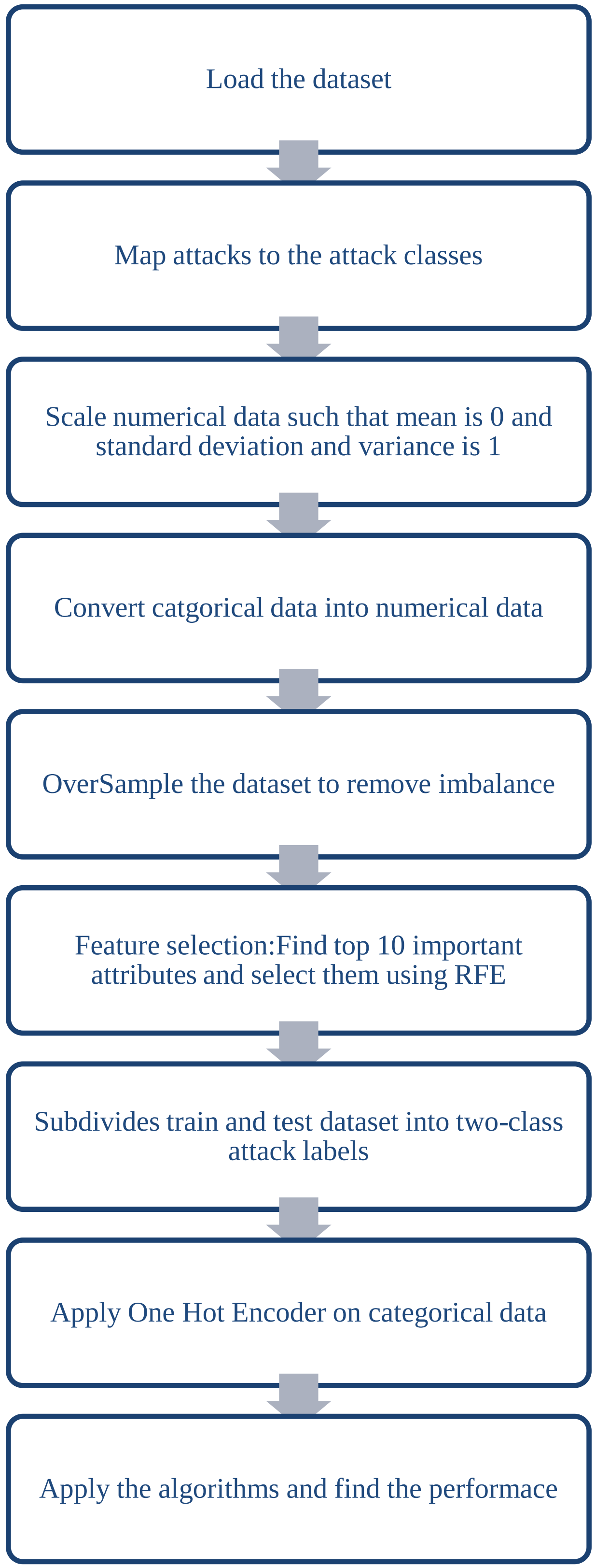

Fig 3: Flowchart representing the flow of the code

5.6 ALGORITHMS USED

Naïve Bayes:

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.

It is called *naive Bayes* or *idiot Bayes* because the calculation of the probabilities for each hypothesis is simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value P(d1, d2, d3|h), they are assumed to be conditionally independent given the target value and calculated as P(d1|h) * P(d2|H) and so on.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold. The representation for naive Bayes is probabilities.

A list of probabilities is stored to file for a learned naive Bayes model. This includes:

- **Class Probabilities**: The probabilities of each class in the training dataset.
- **Conditional Probabilities**: The conditional probabilities of each input value given each class value.

Learning a naive Bayes model from your training data is fast. Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

Decision Tree:

A decision tree models the possible outcomes of a problem statement in a tree like structure. It visually represents the data and the rules in an upside-down tree format. It represents the data in flowchart form and is based on “if.. then… else” rule. It is used to display those algorithms which contain only conditional statements. When representing data in this flowchart type tree, each node represents a condition or a test on the attribute. And each branch will show the outcome of the test performed. Each leaf node in the tree represents the final attribute assigned, i.e. the class label. The path followed to get from a node to the leaf node represents the classification rules followed to get the class label of a particular data.

Decision tree are well suited for classification problems (and also regression problems) where, given a data with all the attributes, the algorithm has to select in which class the data belongs. The algorithm works for both continuous and categorical data (input and output variables). Here we spit our population into two or more sub-populations (or homogeneous sets) based on which input variable has most significant contribution to the class label. To identify the most significant splitter, the tree uses various algorithms based on the problem statement. Tree models usually give better performance than other algorithms.

Commonly used in data mining, this algorithm is widely used in machine learning. The algorithm can easily represent even the most complex datasets (when pruned).

K Nearest Neighbours:

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique. A case, in KNN, is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

It should also be noted that all the distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset. Choosing

the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3 and 10. That produces much better results than 1NN. KNN algorithm is one of the simplest classification algorithms. Even with such simplicity, it can give highly competitive results. KNN algorithm can also be used for regression problems.
One major drawback in calculating distance measures directly from the training set is in the case where variables have different measurement scales or there is a mixture of numerical and categorical variables.

Logistic regression:

Logistic regression is the proper regression analysis to carry out when the dependent variable is binary (dichotomous). It is a predictive analysis. We use logistic regression to describe the data. It also explains the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

It is one of those techniques which are borrowed my Machine Learning from statistics. It uses logistic function, also known as the sigmoid function, to determine the result. It is used when the dependent variable is categorical in nature.

Input values (x) are combined linearly using coefficient values, or weights, (referred to as Beta) to calculate an output value (y). A significant difference between linear regression and logistic regression is that the output value being in linear regression is a binary value (0 or 1) and not a numeric value. The coefficients of the logistic regression algorithm have to be estimated from the training data. To do so we use maximum-likelihood estimation. To make predictions with a logistic regression model we just have to put the numbers into the logistic regression equation and calculate a result.

Assumptions:

1. The dependent variable has to be dichotomous in nature (e.g., dead v/s alive).
2. The data must have no outliers.

3. There must be no high correlations (multi-collinearity) among the predictors. This can be evaluated by a correlation matrix among the predictors.
4. At the center of the logistic regression analysis the task is to estimate the log odds of an event.

Voting classifier:

A Voting Classifier is a machine learning technique that trains on several models and predicts the output (class) for a given data on the basis of their probabilities. It finds the class with highest probability and returns it as the output.

This algorithm considers the output of all the classifiers passed to it and, based on the votes, the output class with maximum votes is selected and declared as output. The basic logic behind this classifier is to combine the outputs of various algorithms and predict a single output based on the highest votes thereby avoiding calculation for each separate classifier. In a way, this is not a classifier but a collection of different classifiers that are trained and tested in parallel so as to gain benefits (and advantage) of each algorithm in order to give the best and most accurate result.

This classifier would be a good choice when there we do not have a single strategy to reach the required output. It helps us in combining the strengths of various algorithms and minimizing their weaknesses/errors.

There are two types of voting:

- Hard voting: here the predicted class is the class with highest accuracy.
- Soft Voting: Here the predicted class is based on average of the probabilities of each class.

5.7 CODES AND STANDARDS

To increase the efficiency and reliability of each algorithm, approach or a service in general, IEEE has detailed standards and guidelines to design each specification.

All the issues, protocols which help in building a better product with an increased functionality have to follow a strict set of standards from IEEE. This ensures safety of customers and public health. IEEE has set up very precise standards. The detection of threat requires that the organization gives us the data which can be used to test from the given model.

The data has a very specific requirement and organizations are obviously very hesitant on sharing the data due to security concerns. For maximum operability and functionality, it should just be asked to the organization to use our models to test the data on their own so that they do not have to think about sharing the data to the developer.

Customer support and public health is ensured when the organization implements and ensures that the data of millions of customers is safe in their database. It is also necessary to ensure that use of proper standards for exchanging the network data and ensure the complete data required for the analysis of testing of the algorithm. The standards require to share all the selected data so that it can be passed into the function and can detect the threat if there’s one at all.

5.8 CONSTRATINTS, ATERNATIVES AND TRADE OFFS

The browser version should be used which have HTML5 support. The system should at least have a 1050 GTX GPU with support to Python 3.7 and RAM of at least 8GB.

Considering the fact that data being exchanged or stored is large, appropriate storage and management of data on the system should be ensured.

6. SCHEDULE, TASKS AND MILESTONE

Figure below is a Gantt chart showing how the work done during this project was distributed across time.

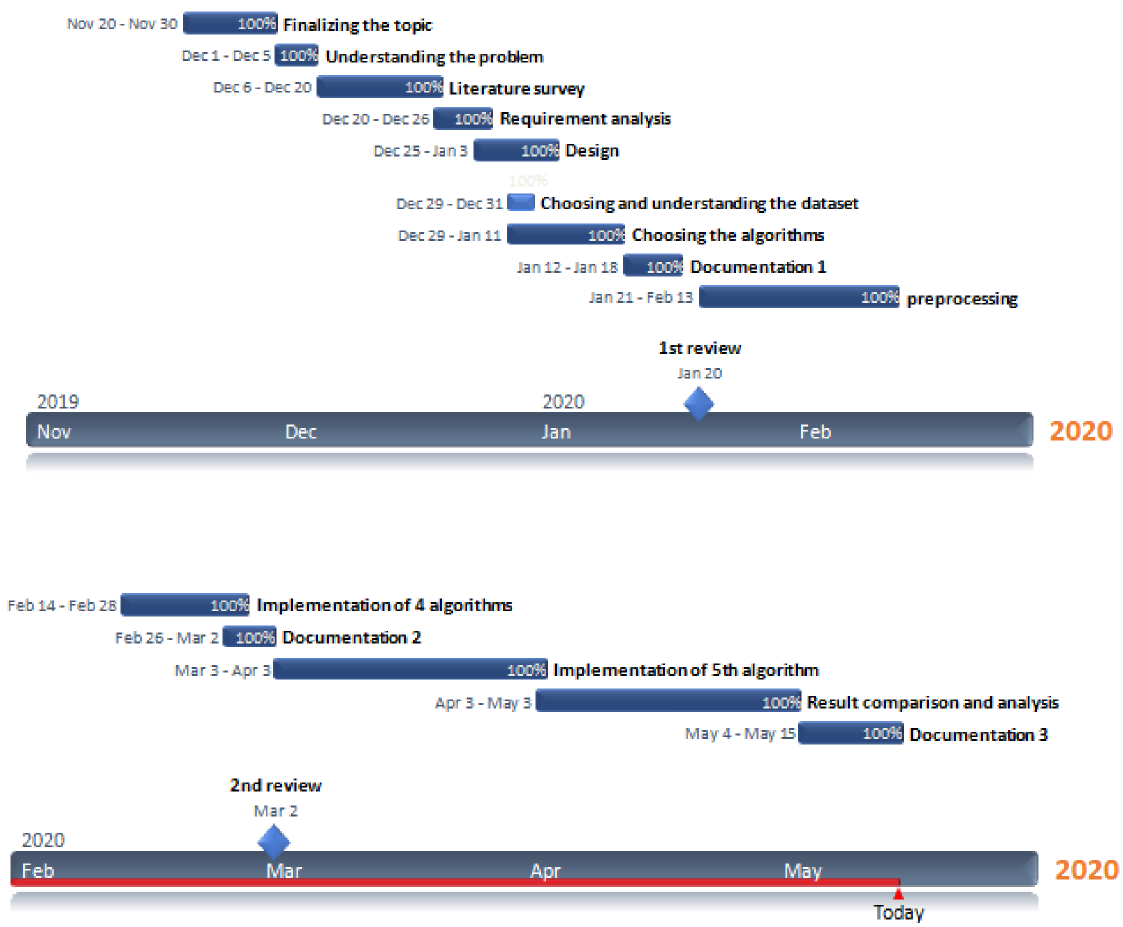

Fig 4: Gantt chart

7. PROJECT DEMONSTRATION

CODE:

```
%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
import warnings
import sys
warnings.filterwarnings('ignore')

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

# Dataset field names
datacols = ["duration","protocol_type","service","flag","sr
c_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num
_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempte
d","num_root",
    "num_file_creations","num_shells","num_access_files","n
um_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","s
error_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","d
st_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","atta
ck", "last_flag"]

# Load NSL_KDD train dataset
dfkdd_train = pd.read_table("https://raw.githubusercontent.
com/dimtics/Network-Intrusion-Detection-Using-Machine-
Learning-
Techniques/master/NSL_KDD_dataset/KDDTrain.txt", sep=",", n
ames=datacols) # change path to where the dataset is locate
d.
dfkdd_train = dfkdd_train.iloc[:,:-
1] # removes an unwanted extra field

# Load NSL_KDD test dataset
dfkdd_test = pd.read_table("https://raw.githubusercontent.c
om/dimtics/Network-Intrusion-Detection-Using-Machine-
Learning-
Techniques/master/NSL_KDD_dataset/KDDTest.txt", sep=",", na
mes=datacols)
dfkdd_test = dfkdd_test.iloc[:,:-1]

dfkdd_test.head()

# train set dimension
print('Train set dimension: {} rows, {} columns'.format(dfk
dd_train.shape[0], dfkdd_train.shape[1]))
# test set dimension
print('Test set dimension: {} rows, {} columns'.format(dfkd
d_test.shape[0], dfkdd_test.shape[1]))
mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Pro
be','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': 'R2
L','warezmaster': 'R2L','warezclient': 'R2L','imap': 'R2L',
        'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmp
guess': '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(l
ambda v: mapping[v])
dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(lam
bda v: mapping[v])
# 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()

# Attack Class Distribution
attack_class_freq_train = dfkdd_train[['attack_class']].app
ly(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.su
m()),2)
attack_class_freq_test['frequency_percent_test'] = round((1
00 * attack_class_freq_test / attack_class_freq_test.sum())
,2)

attack_class_dist = pd.concat([attack_class_freq_train,atta
ck_class_freq_test], axis=1)
attack_class_dist
```

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


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()



# extract numerical attributes and scale it to have zero me

an and unit variance

cols = dfkdd_train.select_dtypes(include=['float64','int64'

]).columns #store column names

sc_train = scaler.fit_transform(dfkdd_train.select_dtypes(i

nclude=['float64','int64']))

sc_test = scaler.fit_transform(dfkdd_test.select_dtypes(inc

lude=['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 tes
t sets
cattrain = dfkdd_train.select_dtypes(include=['object']).co
py()
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) #data wi
th string datatype but without attack class(after convertin
g to numbers)
enctest = testcat.drop(['attack_class'], axis=1)

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

from imblearn.over_sampling import RandomOverSampler  #for
imbalance in dataset
from collections import Counter
```

```
# define columns and extract encoded train set for sampling

sc_traindf = dfkdd_train.select_dtypes(include=['float64','
int64']) #numerical  data
refclasscol = pd.concat([sc_traindf, enctrain], axis=1).col
umns   #encoded string +numerical data-- col names
refclass = np.concatenate((sc_train, enctrain.values), axis
=1)    #up but actual data
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)  #whole dataset, attack
 class (both as 1d array)

print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))

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,'importan
ce':score}) #as it returns only the value, associating it w
ith corresponding column name
importances = importances.sort_values('importance',ascendin
g=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_features


#dataset partition


# 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)


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_clas
s'] == k) | (reftest['attack_class'] == v)]
            classdict[j +'_' + i].append(reftest_set)

create_classdict()

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()

#fit test data
enc.fit(Xtestcat)


# 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)

#to reshape to 1D array
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,)

rom 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

# 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='ent
ropy', random_state=0)
DTC_Classifier.fit(X_train, Y_train);

from sklearn.ensemble import VotingClassifier

# Train Ensemble Model (This method combines all the indivi
dual models above except RandomForest)
combined_model = [('Naive Baye Classifier', BNB_Classifier)
,
                  ('Decision Tree Classifier', DTC_Classifi
er),
                  ('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(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_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 Evalu
ation =============================='.format(grpclass, i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean
())  #mean of each fold on cross validating
    print()
```

```
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()

accuracy = metrics.accuracy_score(Y_train, v.predict(X_trai
n))
confusion_matrix = metrics.confusion_matrix(Y_train, v.pred
ict(X_train))
classification = metrics.classification_report(Y_train, v.p
redict(X_train))
print()
print('============================== {} {} Model Evaluatio
n =============================='.format(grpclass, i))
print()
print ("Cross Validation Mean Score:" "\n", scores.mean())
 #mean of each fold on cross validating
print()
print ("Model Accuracy:" "\n", accuracy)
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
```

```
b = np.zeros((17169,74))
b[:,:-2] = X_test
X_test=b


for i, v in models:

    accuracy = metrics.accuracy_score(Y_test, v.predict(X_t

est))

    confusion_matrix = metrics.confusion_matrix(Y_test, v.p

redict(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()
```

Some important graphs:

```
# 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);
```

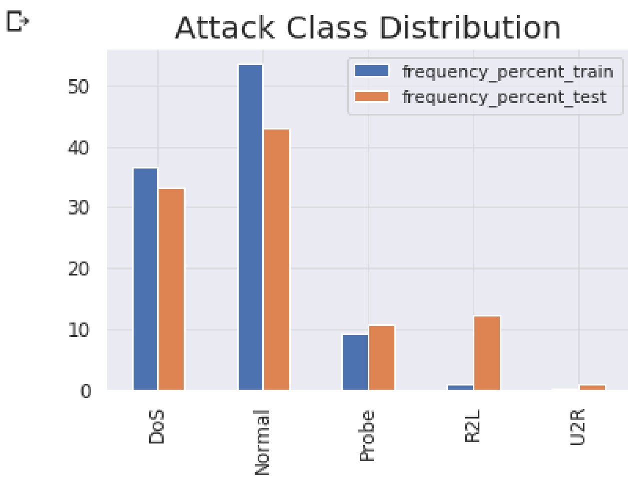

Fig. 5: Distribution chart of the types of attacks and their dominance

```
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();
```

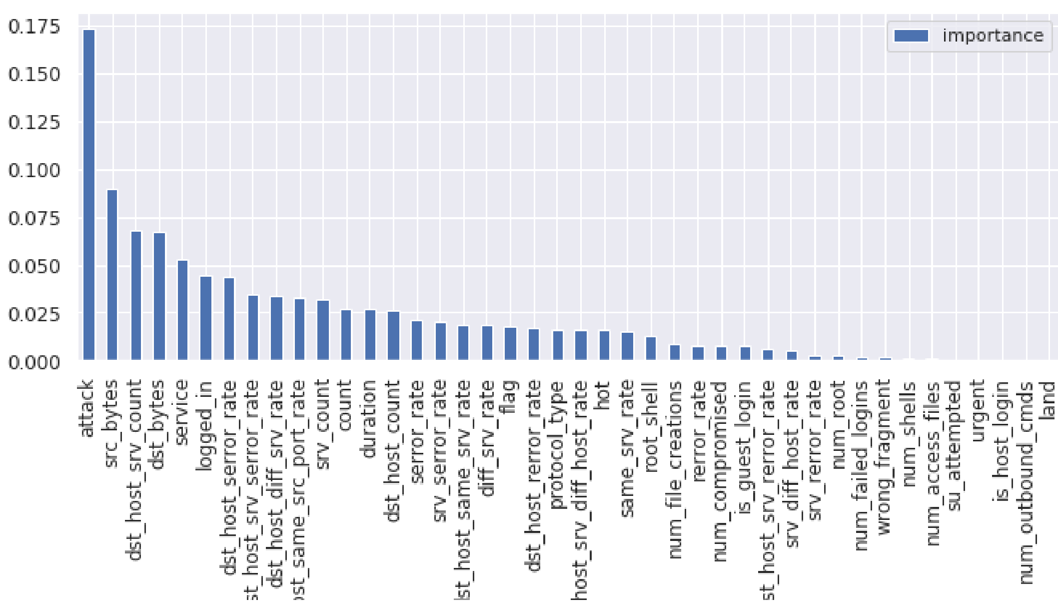

Fig 6: Graph showing importance of various variables in the dataset

```
[ ]  selected_features

['src_bytes',
 'dst_bytes',
 'logged_in',
 'count',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_serror_rate',
 'dst_host_srv_serror_rate',
 'service',
 'attack']
```

Fig 7: Selected features after importance calculation and RFE

7. RESULTS AND DISCUSSIONS

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

Cross Validation Mean Score:
 0.9737760413571536

Model Accuracy:
 0.9737686173767133

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

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

    accuracy                           0.97    134686
   macro avg       0.97      0.97      0.97    134686
weighted avg       0.97      0.97      0.97    134686
```

Fig 8: Naive Bayes Training dataset

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

Model Accuracy:
 0.8421573766672491

Confusion matrix:
 [[5574 1884]
 [ 826 8885]]

Classification report:
              precision    recall  f1-score   support

         0.0       0.87      0.75      0.80      7458
         1.0       0.83      0.91      0.87      9711

    accuracy                           0.84     17169
   macro avg       0.85      0.83      0.84     17169
weighted avg       0.84      0.84      0.84     17169
```

Fig 9: Naive Bayes Testing dataset

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

Cross Validation Mean Score:
 0.9997698368976862

Model Accuracy:
 0.9999480272634127

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

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686
```

Fig 10: Decision tree training dataset

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

Model Accuracy:
 0.815830857941639

Confusion matrix:
 [[5591 1867]
 [1295 8416]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.81      0.75      0.78      7458
         1.0       0.82      0.87      0.84      9711

    accuracy                           0.82     17169
   macro avg       0.82      0.81      0.81     17169
weighted avg       0.82      0.82      0.81     17169
```

Fig 11: Decision Tree testing dataset

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

Cross Validation Mean Score:
 0.99656981084704

Model Accuracy:
 0.9977577476500898

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

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686
```

Fig 12:KNN Traning dataset

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

Model Accuracy:
 0.8433222668763469

Confusion matrix:
 [[5355 2103]
 [ 587 9124]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.90      0.72      0.80      7458
         1.0       0.81      0.94      0.87      9711

    accuracy                           0.84     17169
   macro avg       0.86      0.83      0.84     17169
weighted avg       0.85      0.84      0.84     17169
```

Fig 13: KNN Testing dataset

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

Cross Validation Mean Score:
 0.9808072130256406

Model Accuracy:
 0.980836909552589

Confusion matrix:
 [[65532  1811]
 [  770 66573]]

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

    accuracy                           0.98    134686
   macro avg       0.98      0.98      0.98    134686
weighted avg       0.98      0.98      0.98    134686
```

Fig 14: Logistic regression Training dataset

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

Model Accuracy:
 0.841108975479061

Confusion matrix:
 [[5751 1707]
 [1021 8690]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.85      0.77      0.81      7458
         1.0       0.84      0.89      0.86      9711

    accuracy                           0.84     17169
   macro avg       0.84      0.83      0.84     17169
weighted avg       0.84      0.84      0.84     17169
```

Fig 15: Logistic Regression Testing dataset

```
============================= Normal_DoS VotingClassifier Model Evaluation ====

Cross Validation Mean Score:
 0.9801686927067431

Model Accuracy:
 0.9979879126264051

Confusion matrix:
 [[67149   194]
 [   77 67266]]

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

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686
```

Fig 16: Voting classifier Training dataset

```
============================= Normal_DoS VotingClassifier Model Test Results ===

Model Accuracy:
 0.849030228900926

Confusion matrix:
 [[5643 1815]
 [ 777 8934]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.88      0.76      0.81      7458
         1.0       0.83      0.92      0.87      9711

    accuracy                           0.85     17169
   macro avg       0.86      0.84      0.84     17169
weighted avg       0.85      0.85      0.85     17169
```

Fig. 17: Voting classifier testing dataset

Table 2: Performance measures of the algorithms (on testing dataset)

Sr No.	Algorithm	Precision	Recall	F1 score
1	Naive Bayes	0.85	0.83	0.84
2	Decision Tree	0.82	0.81	0.81
3	KNN	0.86	0.83	0.84
4	Logistic regression	0.84	0.83	0.84
5	Voting classifier	0.86	0.84	0.84

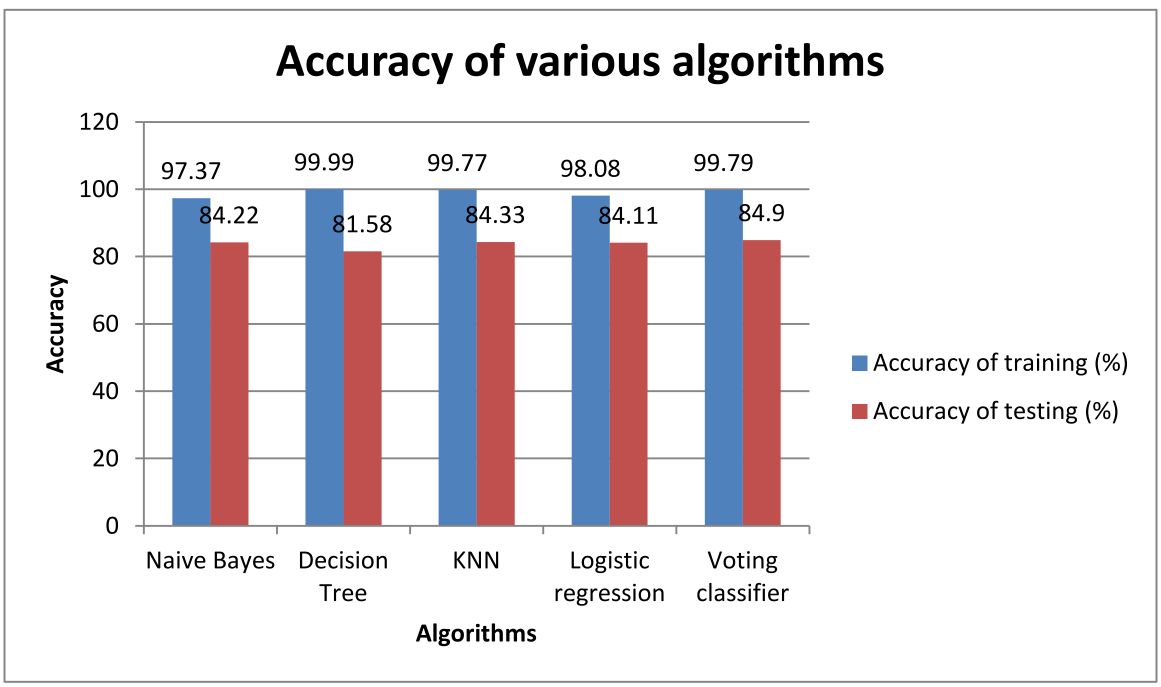

Fig 18: Graph showing accuracy of various algorithms

8. SUMMARY

After studying various research ideas and actually implementing a few of them, a conclusion is reached. It is seen that Decision tree has highest accuracy when tested on training data but lowest when tested on a new testing dataset whereas the Voting classifier made gives highest accuracy when tested on testing dataset and second highest when tested on training dataset. KNN also performs well on this dataset. Voting classifier gives the best accuracy in both the datasets. Hence we can say that, most of the times, when different algorithms are combined and every algorithm's results are taken into consideration, a better classifier is produced. One thing that must be taken care is that these results are because of extreme efficient data cleaning and important feature extraction. Limitation of this work is the introduction of new attacks coming into existence. Moreover the dataset considered here is static and not connected online. When want to use it, the code can be connected to any application dataset can easily be made dynamic.

9. REFERENCES

[1] Sharma, S. (2017). ENHANCE DATA SECURITY IN CLOUD COMPUTING USING MACHINE LEARNING AND HYBRID CRYPTOGRAPHY TECHNIQUES. *International Journal of Advanced Research in Computer Science*, *8*(9).

[2] Aljawarneh, S., Aldwairi, M., & Yassein, M. B. (2018). Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. *Journal of Computational Science*, *25*, 152-160.

[3] Farnaaz, N., & Jabbar, M. A. (2016). Random forest modeling for network intrusion detection system. *Procedia Computer Science*, *89*, 213-217.

[4] Kenkre, P. S., Pai, A., & Colaco, L. (2015). Real time intrusion detection and prevention system. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014* (pp. 405-411). Springer, Cham.

[5] Singh, S., & Bansal, M. (2013). Improvement of intrusion detection system in data mining using neural network. *International Journal of Advanced Research in computer science and software engineering*, *3*(9).

[6] Cui, Y., Xue, J., Wang, Y., Liu, Z., & Zhang, J. (2018, October). Research of Snort Rule Extension and APT Detection Based on APT Network Behavior Analysis. In *Chinese Conference on Trusted Computing and Information Security* (pp. 51-64). Springer, Singapore.

[7] Singh, A., & Chatterjee, K. (2017). Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, *79*, 88-115.

[8] Wani, A. R., Rana, Q. P., & Pandey, N. (2019). Analysis and Countermeasures for Security and Privacy Issues in Cloud Computing. In *System Performance and Management Analytics* (pp. 47-54). Springer, Singapore.

[9] Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, *5*, 21954-21961.

[10] Darwazeh, N. S., Al-Qassas, R. S., & AlDosari, F. (2015). A secure cloud computing model based on data classification. *Procedia Computer Science*, *52*, 1153-1158.

[11] Buczak, A. L., & Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*(2), 1153-1176.

[12] Kevric, J., Jukic, S., & Subasi, A. (2017). An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*, *28*(1), 1051-1058.

[13] Vasan, K. K., & Surendiran, B. (2016). Dimensionality reduction using principal component analysis for network intrusion detection. *Perspectives in Science*, *8*, 510-512.

[14] Panda, M., Abraham, A., & Patra, M. R. (2012). A hybrid intelligent approach for network intrusion detection. *Procedia Engineering*, *30*, 1-9.

APPENDIX A

Code snippets:

The project was implemented on a Google notebook. The online environment of Google is used so that any disk capacity and RAM issues are taken care of.

```
[5] dfkdd_test.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised
0	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	0	0	0
3	0	icmp	eco_i	SF	20	0	0	0	0	0	0	0	0
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	0	0	0

Fig. 19: Test dataset sample

```
[6] # 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]))

    Train set dimension: 125973 rows, 42 columns
```

```
[7] # 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]))

    Test set dimension: 22544 rows, 42 columns
```

Fig. 20: Size of the test and train dataset

```
[ ] mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint
            'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf
            'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
            'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow': 'U2R','xterm
            'sqlattack': 'U2R','httptunnel': 'U2R',
            'ftp_write': 'R2L','phf': 'R2L','guess_passwd': 'R2L','warezmaster': 'R2L','warezc
            'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','worm': 'R2L','sn
            'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',
            'normal': 'Normal'
            }
```

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

Fig. 21: Mapping of the attack to attack class

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

# extract categorical attributes from both training and test set
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) #data with st
enctest = testcat.drop(['attack_class'], axis=1)

#attack class
cat_Ytrain = traincat[['attack_class']].copy()
cat_Ytest = testcat[['attack_class']].copy()
```

Fig 22: converting categorical data into numerical data

```
from imblearn.over_sampling import RandomOverSampler  #for imbalanc
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)  #67343*42*5(data values), 6734

print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
```

Fig 23: OverSampling the data to remove the imbalance

```
Original dataset shape Counter({1: 67343, 0: 45927, 2: 11656, 3: 995, 4: 52})
Resampled dataset shape Counter({1: 67343, 0: 67343, 3: 67343, 2: 67343, 4: 67343})
```

Fig 24: Result before and after oversampling

```python
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

# 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);
```

Fig. 25: Implementing the algorithm

```python
# 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);
```

Fig. 26 Combining the models for voting classifier

```
from sklearn import metrics

models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))
```

Fig. 27: Appending all the models

```
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 ================
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())  #mean of each fol
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

Fig. 28: Finding the metrics for each algorithm