Detecting Prominent Features and Classifying Network Traffic for Securing Internet of Things Based on Ensemble Methods

by

Ramu Ponneganti

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Stephen Yau, Chair
Andrea Richa
Yezhou Yang

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Rapid growth of internet and connected devices ranging from cloud systems to internet of things have raised critical concerns for securing these systems. In the recent past, security attacks on different kinds of devices have evolved in terms of complexity and diversity. One of the challenges is establishing secure communication in the network among various devices and systems. Despite being protected with authentication and encryption, the network still needs to be protected against cyber-attacks. For this, the network traffic has to be closely monitored and should detect anomalies and intrusions. Intrusion detection can be categorized as a network traffic classification problem in machine learning. Existing network traffic classification methods require a lot of training and data preprocessing, and this problem is more serious if the dataset size is huge. In addition, the machine learning and deep learning methods that have been used so far were trained on datasets that contain obsolete attacks. In this thesis, these problems are addressed by using ensemble methods applied on an up to date network attacks dataset. Ensemble methods use multiple learning algorithms to get better classification accuracy that could be obtained when the corresponding learning algorithm is applied alone. This dataset for network traffic classification has recent attack scenarios and contains over fifteen attacks. This approach shows that ensemble methods can be used to classify network traffic and detect intrusions with less training times of the model, and lesser pre-processing without feature selection. In addition, this thesis also shows that only with less than ten percent of the total features of input dataset will lead to similar accuracy that is achieved on whole dataset. This can heavily reduce the training times and classification duration in real-time scenarios.

ACKNOWLEDGMENTS

I would like to thank Professor Stephen S. Yau for providing me motivation, guidance and the opportunity to work on this research. He is a source of inspiration to me. I would also like to thank my committee members Dr. Yezhou Yang and Dr. Andrea Richa for being the part of my graduate supervisory committee, and helping throughout the process with their valuable suggestions.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

There is a popularity in the usage of internet, various devices ranging from mobile phones, internet of things, and cloud systems and computer networks associated with these systems and devices. Hence the security of these has become a prominent research area [4]. Attackers try to figure out the security weaknesses of the networks, vulnerability in the system and try to break through them to cause potential damage or steal vital information, and also attacks like denial of service to cause trouble to the service providers. Firewalls are a sort of network protection technology that is one of the earliest of all the protection measures of the network, which can exclude all the network threads that are not from the inside of the known network. As there is continuous and rapid development in the technology, new attacks are also becoming complex, difficult and diverse. Thus, firewalls may not be able to protect these other categories of complex attacks, and we cannot rely on them. Along with the firewall, another important component in network security, is an intrusion detection system IDS [1] which proactively protects the system. This helps a lot in protecting the integrity of entire security system.

There are two kinds of intrusion detection systems: feature based IDS and anomaly detection IDS [2] [3]. Feature-based IDS can be updated constantly, and it needs the model library of known intrusions. Then, this model is used to detect intrusions based on its model library. One advantage of this kind of IDS is that it reacts quickly to intrusion types in model library. However, feature-based IDS cannot detect new attacks, and it also has to

update uninterruptedly to detect more types of new attacks. Anomaly detection IDS needs to create a complete model of normal data flow to detect intrusions. The model will be able to identify new intrusions.

In this thesis, we will present an approach that gives intrusion detection with minimal misclassification. The approach uses Random Forest, an ensemble learning method, achieves 99.91% accuracy in detecting intrusions. This method also achieves similar accuracy in classifying different types of network attacks. We will show that this approach does not require any feature selection nor conversion of input features in the dataset. This reduces the preprocessing time for the network traffic in real time, and the benefit of Random Forest is that this can be run in parallel for faster processing in real time. We compare this approach with traditional machine learning approaches, other ensemble methods and also explain why this is effective both in accuracy detection and application in real-time. This thesis also shows that ensemble methods are better than deep learning approaches for this kind of intrusion detection problem in terms of both prediction accuracy and performance for training and testing the data. Ensemble methods build a classifier by combining several different independent base classifiers. The independence is theoretically enforced by training each base classifier on a training set sampled with replacement from the original training set. This technique helps in building more generalization of the classifier based on randomness and helps in reducing the variance of the classifier and is shown to be efficient in and accurate in our approach to detect intrusions.

Chapter 2

BACKGROUND

Many works are being carried out in this context to find the best parameters and results for the detection of intrusion in various kinds of systems based on the network traffic. Some recent studies have addressed intrusion detection based on network traffic, such as the work of Ahmed [7], which shows that detection is an important task and that it detects anomalous data from a given data set. The author points out that intrusion detection is an interesting area and that it has been extensively studied in statistics and machine learning. Costa et al. [8] also highlighted the importance of using intelligent tools to assist intrusion detection but in the context of computer networks. In their work, the authors employed the unsupervised Optimum-Path Forest OPF classifier [8] for intrusion detection in computer networks. The authors proposed a nature-based approach to estimate the probability density function pdf used for clustering purposes, which strongly influences the quality of the classification process. Regarding the OPF classifier, Pereira et al. [9] proposed a similar approach to the one presented by Costa et al. [10] but in the context of supervised intrusion detection [11], [12], [13].

In their 2011 work, Le et al. [26] followed the approach of organizing the network in regions. With this approach, they use a hybrid placement strategy to build a backbone of monitor nodes, one per region. The function of monitor nodes is to sniff the communication from its neighbors and define whether a node is compromised. One of the advantages of this solution is that there is no communication overhead. The detection method used is

specification-based focused on detecting RPL attacks. In the paper [27] Liu et al. propose a signature-based IDS that employs Artificial Immune System mechanisms. Detectors with attack signatures were modeled as immune cells that can classify datagrams as malicious or normal, non-self or self-element respectively. The article does not present which placement strategy should be adopted and does not introduce the way that this approach could be implemented in IoT resource constraint networks. In this approach, the computational overhead needed to run learning algorithms might be a disadvantage. Misra et al. [28] present a solution to prevent DDoS attacks over IoT middleware. This specification based detection method, use the maximum capacity of each middleware layer to detect the attacks. The system will generate an alert when the number of requests to a layer exceeds the specified threshold.

Gupta et al. [29] propose an architecture for a wireless IDS. In the architecture proposed, the normal behavior profiles for network devices would be constructed applying Computational Intelligence algorithms. Thus, there would be a specific behavior profile for each device with an IP address assigned. The placement strategy was not presented by the authors neither the type of attacks that could be detected by their solution.

In the following text, background about intrusion detection systems is given in much detail [30]. Monitoring and analyzing user information, networks, and services through passive traffic collection and analysis are useful tools for managing networks and discovering security vulnerabilities in a timely manner [35, 36]. An IDS is a tool for monitoring traffic data to identify and protect against intrusions that threaten the confidentiality, integrity, and availability of an information system [37]. The operations of an IDS can be divided

into three stages. The first stage is the monitoring stage, which relies on network-based or host-based sensors. The second stage is the analysis stage, which relies on feature extraction methods or pattern identification methods. The final stage is the detection stage, which relies on anomaly or misuse intrusion detection. An IDS captures a copy of the data traffic in an information system and then analyzes this copy to detect potentially harmful activities [38].

The concept of an IDS as an information security system has evolved considerably over the past 30 years. During these years, researchers have proposed various methods and techniques for protecting different types of systems using IDSs. In 1987, Denning presented an intrusion detection model that could compare malicious attack behavior against the normal model for the system of interest [39]. The implementation of an IDS depends on the environment. A host-based intrusion detection system HIDS is designed to be implemented on a single system and to protect that system from intrusions or malicious attacks that will harm its operating system or data [41]. A HIDS generally depends on metrics in the host environment, such as the log files in a computer system [42]. These metrics or features are used as input to the decision engine of the HIDS. Thus, feature extraction from the host environment serves as the basis for any HIDS. A network-based intrusion detection system NIDS sniffs network traffic packets to detect intrusions and malicious attacks [41]. A NIDS can be either a software-based system or a hardware-based system. For example, Snort NIDS is a software-based NIDS [42].

An IDS depends on algorithms for implementing the various stages of intrusion detection. There are a vast number of algorithms for all IDS types and methods. Principal component

analysis PCA is a lightweight algorithm that can be used for various detection techniques in IDSs. Machine learning is a subfield of computer science, and is a type of Artificial Intelligence that provides machines with the ability to learn without explicit programming. Machine learning evolved from pattern recognition and computational learning theory.

A security mechanism used to monitor the abnormal behavior of the network is an Intrusion Detection System (IDS) [48]. The IDS identifies and informs that whether the user activity is normal or not. The users activities are compared by the IDS with the already stored intrusion records to identify the intrusion. Accurate predictive models can be built for large data sets using supervised machine learning techniques, that is not possible by traditional methods. As specified by Tom Mitchell [49], machine learning based intrusion detection falls under two categories Anomaly and Misuse. IDS learns the patterns by the training data, so the misuse based method is used. Misuse based detection can detect only the known attack, new attacks cannot be identified. Anomaly based IDS observes the normal behavior and if there is a change in the behavior then it considers that behavior as anomaly. So anomaly based IDS can detect new attacks that are not learned from the training model. Till now different machine learning techniques such as Artificial neural networks [50], Support Vector Machine4andNaive Bayes [51], [52], based techniques are proposed for the intrusion detection. A new detection by combining different techniques, a hybrid detection technique is proposed by [52].

Nadiammai [53] proposed semi supervised machine learning based intrusion detection. Authors have not considered the resource consumption. Combination of different classifiers to identify the intrusion is proposed by Panda [52]. They used supervised classification or unsupervised clustering for filtering of the data. They used NSL-KDD dataset and tested with decision tree classifier. But the proposed method works only for binary class classification.

Sangkatsanee [54] proposed intrusion detection system using supervised machine learning techniques to identify the on line network data as normal or not. The proposed method identifies probe and Denial of Service attacks only, but the other attacks are not considered. A framework of machine learning approach is proposed by Yu [48] and Campos [55]. Intrusion is identified by analyzing the local features. Levent [56] proposed Naive Bayes based multiclass classifier to identify the intrusions. They suggested that intrusion detection is possible by Hidden Naive Bayes (HNB) model. Denial of Service attacks are identified with good accuracy compared to other attacks.

Li proposed [57] Intrusion detection technique using Support Vector Machine (SVM). They also used feature removal method to improve the efficiency. Using the proposed feature removal method they selected best nineteen features from the KDD-CUP99 data-set. In the proposed method the data set used is very small. A light weight IDS is proposed by Sivatha Sindhu [58]. The proposed method mainly focused on pre-processing of the data so that only important attributes can be used. The first step is to remove the redundant data so that the learning algorithms give the unbiased result.

A survey on intrusion detection systems was conducted by Butan [59] Information about IDSs such as classification, Intrusion type, computing location and infrastructure are discussed. They discussed about the Mobile Ad hoc Networks (MANET) IDS. They compared MANETIDS and the Wireless Sensor Networks (WSN) IDS. Authors suggested that for mobile applications distributed and cooperative IDS schemes are suitable. For stationary applications centralized IDSs are suitable and for cluster based applications hierarchical IDSs are suitable. Farooqi [60] proposed intrusion detection framework to detect routing attacks. Specification based approach is used to detect routing attacks. Authors claim that the proposed method has low False Positive Rate (FPR) and good intrusion detection rate. The proposed method works only for static networks. Wang [61] developed IDS for Sink, Cluster Head (CH) and for a Sensor Node (SN) separately and combined altogether to identify the intrusion in heterogeneous Cluster Based Wireless Sensor Networks (CWSN) but the detection rate for U2R, R2L and Probe attacks is very low.

Following are the supervised machine learning techniques that are traditional machine learning techniques:

Logistic Regression: Following description of this classifier is taken from Wikipedia. In statistics, the logistic model (or logit model) is a widely used statistical model that, in its basic form, uses a logistic function to model a binary dependent variable; many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model; it is a form of binomial regression. Mathematically, a binary logistic model has a dependent variable with two possible values,

such as pass/fail, win/lose, alive/dead or healthy/sick; these are represented by an indicator variable, where the two values are labeled "0" and "1". The binary logistic regression model has extensions to more than two levels of the dependent variable: categorical outputs with more than two values are modeled by multinomial logistic regression, and if the multiple categories are ordered, by ordinal logistic regression, for example the proportional odds ordinal logistic model. The model itself simply models probability of output in terms of input, and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier. The coefficients are generally not computed by a closed-form expression, unlike linear least squares. Following equation generally represents the logistic function:

$$h(X;W) = g(\frac{1}{1 + e^{-W^T X}})$$

Support Vector Machine: The following text of SVM is taken from Wikipedia. In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). A SVM model is a representation of the examples as points in space,

mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Gaussian Naïve Bayes: The Gaussian Naive Bayes algorithm is the supervised learning method. Probabilities of each attribute which belongs to each class are considered for a prediction. This algorithm is assumes that the probability of each attribute belonging to a given class value is not depends on all other attributes. If the value of the attribute is known the probability of a class value is called as the conditional probabilities. Data instances provability can be found out by multiplying all attributes conditional probabilities together. Prediction can be made by calculating the each class instance probabilities and by selecting the highest probability class value [21]. Following the popular Bayes Theorem:

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

We model this problem as supervised learning. We first describe what are the problems with traditional machine learning methods, then explain how ensemble methods overcome why Random Forest is robust and achieves great accuracy in detecting intrusions on this dataset.

In the following two sections, some problems with the traditional methods of machine learning are discussed.

Bias and Variance Tradeoff: The small part of this section is borrowed from [20]. A model is biased if it systematically under or over predicts the target variable. In machine learning, this is often the result either of the statistical assumptions made by our model of choice or of bias in the training data. Variance, on the other hand, in some sense captures the generalizability of the model. Put more precisely, it is a measure of how much our prediction would change if we trained it on different data. High variance typically means that we are overfitting to our training data, finding patterns and complexity that are a product of randomness as opposed to some real trend. Generally, a more complex or flexible model will tend to have high variance due to overfitting but lower bias because, averaged over several predictions, our model more accurately predicts the target variable. On the other hand, an underfit or oversimplified model, while having lower variance, will likely be more biased since it lacks the tools to fully capture trends in the data. This is shown in figure 1.

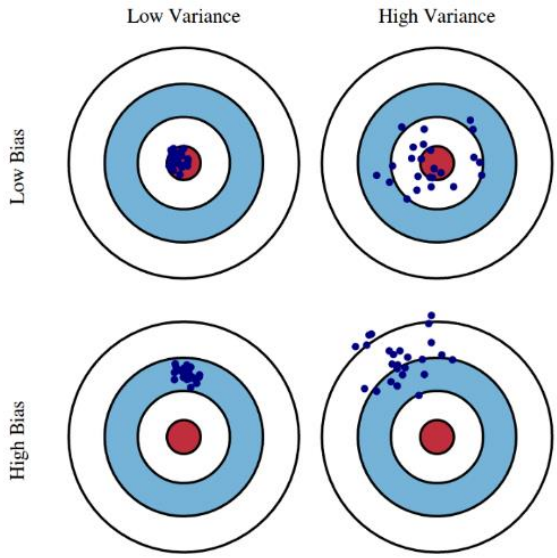

Figure 1: Bias and Variance Trade-off [20]

What we would like, ideally, is low bias-low variance. To see how to achieve this, lets first look at a typical bias squared-variance curve in Figure 2.

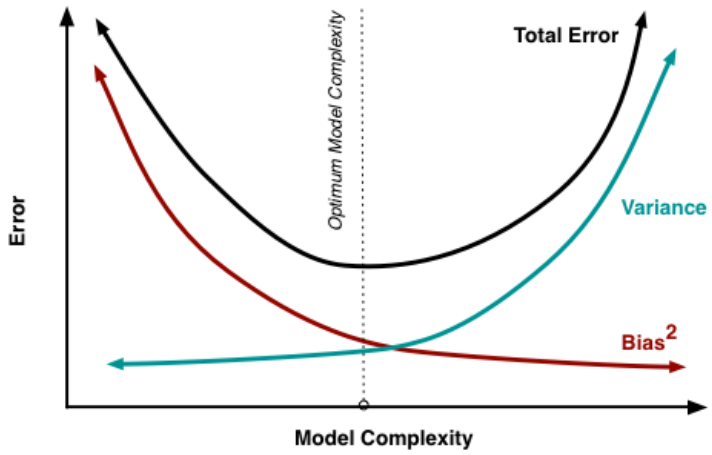

Figure 2: Bias and Variance Curve [20]

From the above figure 2, we can see that as the model complexity increases the variance of the model increases and is not generalized for new test samples, on the other hand, if the model is not well trained on the test dataset, it will be with high bias, and cannot even fit properly for the training dataset. So, in practicality, we need to choose a model that has a balance of both bias and variance, this essentially means that model should generalize to test dataset well, and should not overfit

Chapter 3

OUR APPROACH

3.1 Overview

We start with the drawbacks of machine learning methods on large datasets with imbalanced data, and explain it in section 3.2. In section 3.3 we explain the dataset, and in section 3.4 we explain the data preprocessing. We explain the algorithms used in this approach in section 3.5 and how we test the model in section 3.6.

Following are the steps in our approach:

1. Dataset collection
2. Preprocess the dataset by creating the labels to each class and split into training data set and test data set
3. Use the random forest algorithm to train the data – it handles outliers in the training set, and also bias and variance problems, without any hyper parameter tuning as described in the following sections
4. Run the model on the test dataset for evaluation

3.2 Dataset Collection

CICIDS dataset [19] was used for experiments. It contains PCAP packet capture files of network traffic data. More details are given in section 4.1

3.3 Preprocessing

Outliers [23]: Outliers are generally defined as cases that are removed from the main body of the data. Outliers are cases whose proximities to all other cases in the data are generally small. A useful revision is to define outliers relative to their class.

Data Normalization: Ensemble methods like Random Forest are robust to unscaled data, but the normalization process helps to train faster and handling small values without any overflows or datatype errors. This also helps to compare with other machine learning models that perform best with normalized data. In our approach, we normalized the data.

The ensemble methods that we used in this approach are:

1. Random Forest
2. AdaBoost
3. Gradient Boosting

Each of these methods is used for the following classification types:

1. Detecting intrusions with full features dataset
2. Classifying network traffic with full features dataset
3. Detecting intrusions with selected features and show that the top features also are enough for intrusion detection

3.4 Algorithms used in this approach

3.4.1 Random Forest

Random forests overcome these by using an ensemble method of learners and voting mechanism and this process is described in the following paragraphs in detail. We first start with ensemble methods in detail, then followed by random forest, and how it handles outliers, missing data and without a need for cross validation dataset and minimal hyper parameter tuning.

Ensemble methods [21]: Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning

algorithms alone. A machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.

Supervised learning algorithms are most commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a hopefully better hypothesis. The term ensemble is usually reserved for methods that generate multiple hypotheses using the same base learner. The broader term of multiple classifier systems also covers hybridization of hypotheses that are not induced by the same base learner.

Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. Fast algorithms such as decision trees are commonly used in ensemble methods for example, random forests, although slower algorithms can benefit from ensemble techniques as well. Random Forest boosting and bagging methods are robust to missing data, outliers and they can be used without any feature scaling and normalization. These methods do not even need hyper-parameter setting, which is one of the most difficult task in training machine learning models.

Random Forest is also considered as a very handy and easy to use algorithm, because its default hyperparameters often produce a good prediction result. The number of

hyperparameters is also not that high and they are straightforward to understand. One of the big problems in machine learning is overfitting, but most of the time this will not happen that easy to a random forest classifier. That is because if there are enough trees in the forest, the classifier wont overfit the model. Figure 3 gives overview of Random Forest.

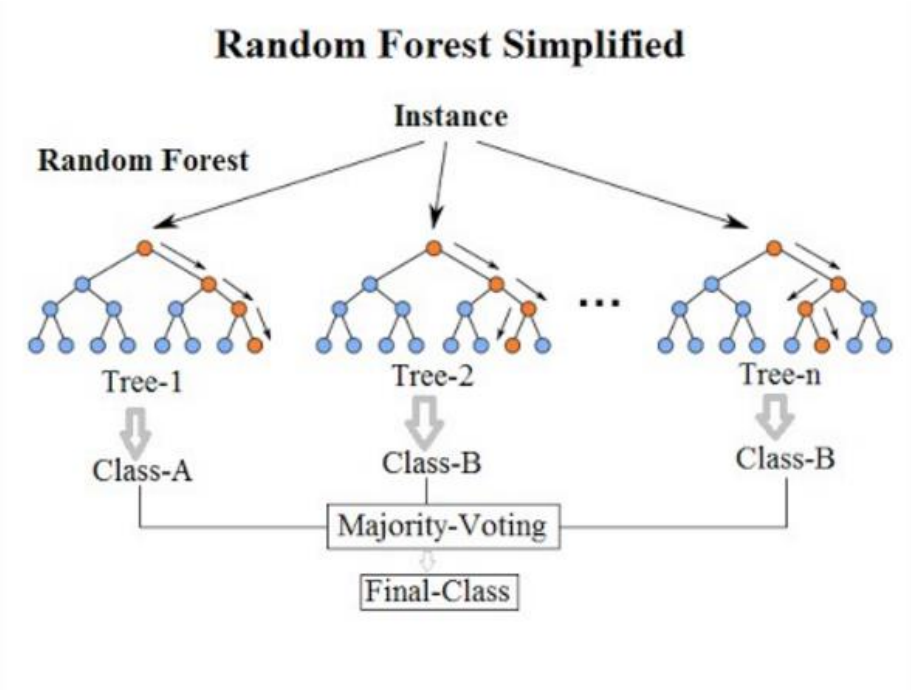

Figure 3: Overview of Random Forest [47]

Random Forest Training Algorithm: Random forest is uses a method called Bootstrap Aggregation bagging [22]. Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

The training algorithm for random forests applies the general technique of bootstrap aggregating, to tree learners. Given a training dataset $X = x_1, x_2, \dots, x_n$ with class labels $Y = y_1, y_2, \dots, y_n$, bagging approach selects a random sample repeatedly B times with replacement of the training set and fits decision trees to these samples:

For $b = 1, 2, \dots, B$:

1. Sample, with replacement, n training examples from X, Y and call these X_b, Y_b.
2. Train a classification tree f_b on X_b, Y_b.

After training, predictions for unseen samples x can be made by averaging the predictions from all the individual regression trees on x or by taking the majority vote from the decision trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees or even the same tree many times, if the training algorithm is deterministic; bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x.

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal

number of trees B can be found using cross-validation, or by observing the out-of-bag error, the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

From bagging to random forests: The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable target output, these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contribute to accuracy gains under different conditions is given by Ho.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This out-of-bag OOB data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.

After each tree is built, all of the data are run down the tree, and proximities are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

The out-of-bag (OOB) error estimate [23]: In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run, as follows: Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree. Put each case left out in the construction of the kth tree down the kth tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take j to be the class that got most of the votes every time case n was OOB. The proportion of times that j is not equal to the true class of n averaged over all cases is the OOB error estimate. This has proven to be unbiased in many tests.

Variable importance: In every tree grown in the forest, put down the OOB cases and count the number of votes cast for the correct class [23]. Now randomly permute the values of variable m in the OOB cases and put these cases down the tree. Subtract the number of votes for the correct class in the variable-m-permuted OOB data from the number of votes for the correct class in the untouched OOB data. The average of this number over all trees in the forest is the raw importance score for variable m. If the values of this score from tree to tree are independent, then the standard error can be computed by a standard computation. The correlations of these scores between trees have been computed for a number of data sets and proved to be quite low, therefore we compute standard errors in the classical way, divide the raw score by its standard error to get a z-score, ands assign a significance level to the z-score assuming normality.

If the number of variables is very large, forests can be run once with all the variables, then run again using only the most important variables from the first run. For each case, consider all the trees for which it is OOB. Subtract the percentage of votes for the correct class in the variable-m-permuted OOB data from the percentage of votes for the correct class in the untouched OOB data. This is the local importance score for variable m for this case.

Gini importance: Every time a split of a node is made on variable *m* the Gini impurity criterion for the two descendent nodes is less than the parent node [23]. Adding up the Gini decreases for each individual variable over all trees in the forest gives a fast variable importance that is often very consistent with the permutation importance measure.

Interactions [23]: The operating definition of interaction used is that variables m and k interact if a split on one variable, say m, in a tree makes a split on k either systematically less possible or more possible. The implementation used is based on the Gini values gm for each tree in the forest. These are ranked for each tree and for each two variables, the absolute difference of their ranks are averaged over all trees. This number is also computed under the hypothesis that the two variables are independent of each other and the latter subtracted from the former. A large positive number implies that a split on one variable inhibits a split on the other and conversely. This is an experimental procedure whose conclusions need to be regarded with caution. It has been tested on only a few data sets.

Interactions: The operating definition of interaction used is that variables m and k interact if a split on one variable, say m, in a tree makes a split on k either systematically less possible or more possible. The implementation used is based on the gini values g(m) for

each tree in the forest. These are ranked for each tree and for each two variables, the absolute difference of their ranks are averaged over all trees.

This number is also computed under the hypothesis that the two variables are independent of each other and the latter subtracted from the former. A large positive number implies that a split on one variable inhibits a split on the other and conversely. This is an experimental procedure whose conclusions need to be regarded with caution. It has been tested on only a few data sets.

Proximities [23]: These are one of the most useful tools in random forests. The proximities originally formed NxN matrix. After a tree is grown, put all of the data, both training and oob, down the tree. If cases k and n are in the same terminal node increase their proximity by one. At the end, normalize the proximities by dividing by the number of trees.

Users noted that with large data sets, they could not fit an NxN matrix into fast memory. A modification reduced the required memory size to NxT where T is the number of trees in the forest. To speed up the computation-intensive scaling and iterative missing value replacement, the user is given the option of retaining only the nrnn largest proximities to each case. When a test set is present, the proximities of each case in the test set with each case in the training set can also be computed. The amount of additional computing is moderate.

Missing value replacement for the training set [23]: Random forests has two ways of replacing missing values. The first way is fast. If the m variable is not categorical, the method computes the median of all values of this variable in class j, then it uses this value to replace all missing values of the m variable in class j. If the m variable is categorical,

the replacement is the most frequent non-missing value in class j. These replacement values are called fills. The second way of replacing missing values is computationally more expensive but has given better performance than the first, even with large amounts of missing data. It replaces missing values only in the training set. It begins by doing a rough and inaccurate filling in of the missing values. Then it does a forest run and computes proximities. If x(m, n) is a missing continuous value, estimate its fill as an average over the non-missing values of the m variables weighted by the proximities between the nth case and the non-missing value case. If it is a missing categorical variable, replace it by the most frequent non-missing value where frequency is weighted by proximity. Now iterate-construct a forest again using these newly filled in values, find new fills and iterate again. Our experience is that 4-6 iterations are enough.

Missing value replacement for the test set: When there is a test set, there are two different methods of replacement depending on whether labels exist for the test set. If they do, then the fills derived from the training set are used as replacements. If labels no not exist, then each case in the test set is replicated n-class times (n-class = number of classes). The first replicate of a case is assumed to be class 1 and the class one fills used to replace missing values. The 2nd replicate is assumed class 2 and the class 2 fills used on it. This augmented test set is run down the tree. In each set of replicates, the one receiving the most votes determines the class of the original case.

Balancing prediction error [23]: In some data sets, the prediction error between classes is highly unbalanced. Some classes have a low prediction error, others a high. This occurs usually when one class is much larger than another. Then random forests, trying to

minimize overall error rate, will keep the error rate low on the large class while letting the smaller classes have a larger error rate. For instance, in drug discovery, where a given molecule is classified as active or not, it is common to have the actives outnumbered by 10 to 1, up to 100 to 1. In these situations, the error rate on the interesting class (actives) will be very high. The user can detect the imbalance by outputs the error rates for the individual classes. To illustrate 20 dimensional synthetic data is used. Class 1 occurs in one spherical Gaussian, class 2 on another. A training set of 1000 class 1's and 50 class 2's is generated, together with a test set of 5000 class 1's and 250 class 2's. The final output of a forest of 500 trees on this data is:

500 3.7 0.0 78.4

There is a low overall test set error (3.73%) but class 2 has over 3/4 of its cases misclassified. The error can balancing can be done by setting different weights for the classes. The higher the weight a class is given, the more its error rate is decreased. A guide as to what weights to give is to make them inversely proportional to the class populations. So set weights to 1 on class 1, and 20 on class 2, and run again.

The output is: 500 12.1 12.7 0.0

The weight of 20 on class 2 is too high. Set it to 10 and try again, getting 500 4.3 4.2 5.2. This is pretty close to balance. If exact balance is wanted, the weight on class 2 could be jiggled around a bit more. Note that in getting this balance, the overall error rate went up. This is the usual result - to get better balance, the overall error rate will be increased.

Following are the features of Random Forest [23]:

- It is unexcelled in accuracy among current algorithms.

- It runs efficiently on large datasets.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

3.4.2 AdaBoost

AdaBoost is short form of Adaptive Boosting [43] is a machine learning algorithm used along with many other algorithms to improve the performance. The output of other algorithms is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is called adaptive because following weak learners are adjusted in

favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proved to converge to a strong learner.

During each iteration of the training process, a weight is assigned to each sample in the training set equal to the current error on that sample. These weights can be used to inform the training of the weak learner and can be grown that favor splitting sets of samples with high weights. Following explains the mathematics involved in building the model.

A general boosting classifier is of the following form

$$F_T(x) = \sum_{t=1}^{T} f_t x$$

Where, f_t is a weak learner and takes an input x and returns a value indicating the classification prediction. In a two class problem, the sign will be predicted object category. Likewise, T^{th} classifier is positive if the sample belongs to positive class and otherwise it is negative.

So, each weak learner will produce a hypothesis, hx_i, for each sample in the training dataset. At each iteration t, a weak learner is selected and assigned a coefficient such that the sum traininig error E_t of the resulting t-stage boost classifier is minimized. Following equation explains this concept.

$$E_t = \sum_i E[F_{T-1}(x_i) + a_t hx_i]$$

Where $F_{T-1}(x_i)$ is the boosted classifier that is built up to the previous stage, and $E[F]$ is an error function and $a_t hx_i$ is a weak learner that is under consideration.

At each iteration in the training process, a weight $w_{i,t}$ is assigned to each sample in the training dataset that is same as the current error on that sample. These weights can be used to inform the training of weak learner and can be grown in the favor of splitting sets of samples with high weights. Another variation of this boosting algorithm is called Gradient Boosting explained in further section 3.5.3.

Training AdaBoost Model [43]: In the first step, a weak classifier is prepared on the training data using the weighted samples. Only binary two-class classification problems are supported, so each decision classifier makes one decision on one input variable and outputs a +1.0 or -1.0 value. The misclassification rate is calculated for the trained model. Traditionally, this is calculated using the formula: E = C – N / N

Where E is the misclassification rate, C is the number of training instance predicted correctly by the model and N is the total number of training instances. This is modified to use the weighting of the training instances and is the weighted sum of the misclassification rate, where W is the weight for training instance I, and TE is the prediction error for training instance *i* which is 1 if misclassified and 0 if correctly classified.

Early Termination: This is taken from Wikipedia: A technique for speeding up processing of boosted classifiers, early termination refers to only testing each potential object with as many layers of the final classifier necessary to meet some confidence threshold, speeding up computation for cases where the class of the object can easily be determined. If 50% of negative samples are filtered out by each stage, only a very small number of objects would

pass through the entire classifier, reducing computation effort. This method has since been generalized, with a formula provided for choosing optimal thresholds at each stage to achieve some desired false positive and false negative rate.

In the field of statistics, where AdaBoost is more commonly applied to problems of moderate dimensionality, early stopping is used as a strategy to reduce overfitting. A validation set of samples is separated from the training set, performance of the classifier on the samples used for training is compared to performance on the validation samples, and training is terminated if performance on the validation sample is seen to decrease even as performance on the training set continues to improve.

Pruning: Pruning is the process of removing poorly performing weak classifiers to improve memory and execution time cost of the boosted classifier. The simplest methods, which can be particularly effective in conjunction with totally corrective training, are weight- or margin-trimming: when the coefficient, or the contribution to the total test error, of some weak classifier falls below a certain threshold, that classifier is dropped.

3.4.3 Gradient Boosting Classifier [44]:

Gradient Boosting classifier [44] is a machine learning algorithm that produces a model based on the ensemble of weak prediction models like decision trees. Gradient boosting has three components in it.

1. Weak learner
2. Loss Function
3. An additive model to weak learners to minimize the loss function

An advantage of this algorithm is that new boosting does not have to be derived for each loss function and it is generic that any loss function can be used. Decision trees are the weak learners in gradient boosting algorithm. These are constructed in a greedy way and Gini scores are used to choose the best split to minimize the loss function. Trees are added at one time, and the trees that are already in the model will not be changed. The typical gradient descent procedure is used to minimize the loss function. The output from the new tree is added to the existing sequence of trees to improve the output of the model. Another version of this algorithm stochastic gradient boosting is also present, in which a subsample is taken at random. Following explains the mathematical explanation [44] of this algorithm. Gradient boosting also combines a set of weak learners to a single strong learner in an iterative manner. It is easiest to explain in the least squares setting where the goal is to teach a model to predict values in the form of minimizing the least squares error.

At each stage $m, 1 \leq m \leq M$, of gradient boosting algorithm, we can assume that there is an imperfect model F_m. The gradient boosting algorithm improves this F_m by constructing a new model $F_{m+1}(x) = F_m(x) + h(x)$, that adds an estimator to produce a better model. To find this estimator, the gradient boosting starts with observation that a perfect estimator $h(x)$ would imply

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

Hence, gradient boosting will fit $h(x)$ to the residual function. Similar to the other boosting variants, $F_{m+1}(x)$ attempts to correct the errors of $F_m(x)$. The generalization of this to loss functions other than squared error, and to classification.

Regularization in Gradient Boosting: Fitting the training set too closely can lead to degradation of the model's generalization ability. Several so-called regularization techniques reduce this overfitting effect by constraining the fitting procedure. One natural regularization parameter is the number of gradient boosting iterations M (i.e. the number of trees in the model when the base learner is a decision tree). Increasing M reduces the error on training set, but setting it too high may lead to overfitting. An optimal value of M is often selected by monitoring prediction error on a separate validation data set. Besides controlling M, several other regularization techniques are used.

Penalizing Complexity of Tree: Another useful regularization techniques for gradient boosted trees is to penalize model complexity of the learned model. The model complexity can be defined as the proportional number of leaves in the learned trees.

Following figure outlines our approach:

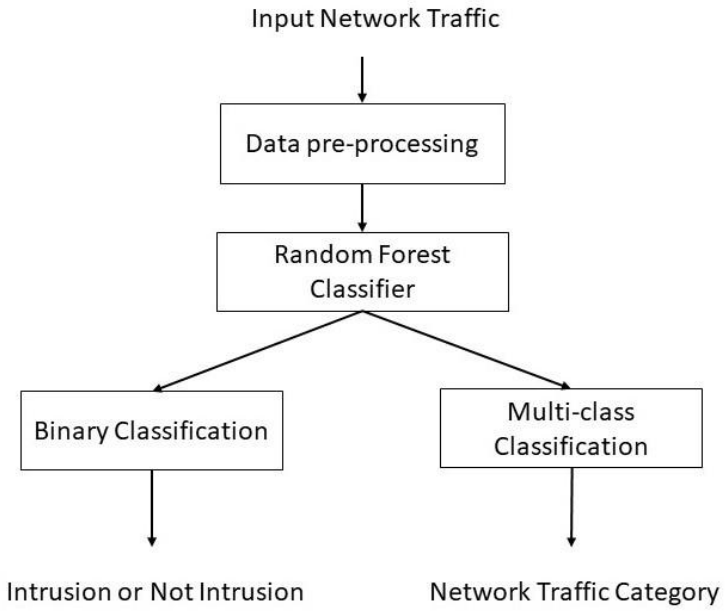

Figure 4: Our Overall Approach

Chapter 4

EXPERIMENTS AND EVALUATION

First, we describe the dataset used, followed by the libraries/tools for the experiments. Then, we describe the evaluation metrics and their importance for classification problems, especially network traffic classification. Then, we show the ensemble methods results, and compare them with different kinds of machine learning algorithms

4.1 Dataset

The CICIDS dataset [19] was used in our experiments. It contains PCAP packet capture files of network traffic data as described in the following paragraphs.

Many of the previous datasets such as DARPA98, KDD99, ISC2012, and ADFA13 are out of date and are not reliable to use as there is a rapid change in the network attacks. This dataset contains benign traffic and common attack network flows, and covers all the eleven necessary criteria with common updated attacks such as DoS, DDoS, Brute Force, XSS, SQL Injection, Infiltration, Port scan and Botnet. The dataset is completely labelled and more than 80 network traffic features extracted and calculated for all benign and intrusive flows by using CICFlowMeter software, which is publicly available in Canadian Institute for Cybersecurity website. The following sections are borrowed from the dataset website [19] and publication [19] associated with it.

Dataset generation [19]: For this dataset, they used their proposed B-Profile system which is responsible for profiling the abstract behavior of human interactions and generate a naturalistic benign background traffic. The BProfile for this dataset extracts the abstract behavior of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols. At First,

it tries to encapsulate network events produced by users with machine learning and statistical analysis techniques. The encapsulated features are distributions of packet sizes of a protocol, the number of packets per flow, certain patterns in the payload, the size of the payload, and request time distribution of protocols. Then, after deriving the B-Profiles from users, an agent which has been developed by Java is used to generating realistic benign events and simultaneously perform B-Profile on the Victim-Network for predefined five protocols.

4.1.1 Types of Attacks:

Common attack families are Brute Force Attack, Heartbleed Attack, Botnet, DoS Attack, and DDoS Attack [19], Web Attack, and Infiltration Attack which are described as follows:

Brute Force Attack: This is one of the most popular attacks that only cannot be used for password cracking, but also to discover hidden pages and content in a web application. It is basically a hit and try attack, then the victim succeeds.

Heartbleed Attack: It comes from a bug in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security TLS protocol. It is normally exploited by sending a malformed heartbeat request with a small payload and large length field to the vulnerable party usually a server in order to elicit the victims response.

Botnet: A number of Internet-connected devices used by a botnet owner to perform various tasks. It can be used to steal data, send spam, and allow the attacker access to the device and its connection.

DoS Attack: The attacker seeks to make a machine or network resource unavailable temporarily. It typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

DDoS Attack: It typically occurs when multiple systems, flood the bandwidth or resources of a victim. Such an attack is often the result of multiple compromised systems for example, a botnet flooding the targeted system with generating the huge network traffic.

Web Attack: This attack types are coming out every day, because individuals and organizations take security seriously now. We use the SQL Injection, which an attacker can create a string of SQL commands, and then use it to force the database to reply the information, Cross-Site Scripting XSS which is happening when developers dont test their code properly to find the possibility of script injection, and Brute Force over HTTP which can tries a list of passwords to find the administrators password.

Infiltration Attack: The infiltration of the network from inside is normally exploiting a vulnerable software such as Adobe Acrobat Reader. After successful exploitation, a backdoor will be executed on the victims' computer and can conduct different attacks on the victims' network such as IP sweep, full port scan and service enumerations using Nmap.

From PCAP files 80 traffic features from the dataset using CICFlowMeter. CICFlowMeter is a flow based feature extractor and can extract 80 features from a pcap file. The flow label in this application includes SourceIP, SourcePort, DestinationIP, DestinationPort and Protocol.

Criterial for this dataset [19]: Regarding to the last dataset evaluation framework published on 2016, covering eleven criteria is necessary for each dataset. None of the previous IDS available datasets could cover all of the criteria. Complete Traffic, Labelled Dataset, Complete Interaction, Complete Capture, Available Protocols, Attack Diversity, Heterogeneity, Feature Set, Meta Data. We show that random forest is the best machine-learning algorithm for classifying network traffic with high accuracy compared to other algorithms.

4.2 Experiment setup

NumPy and Pandas were used to convert the raw dataset into a format that can be given as input to the scikit-learn library. The dataset is split into train and test randomly in 80-20 ratio for intrusion detection and same ratio for the network traffic classification with stratified sampling. To choose the model parameters, we have used five-fold cross validation on ensemble methods, and also show that there is no need to hyper-parameter tuning for ensemble methods.

Numpy [45]: NumPy is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. NumPy is licensed under the BSD license, enabling reuse with few restrictions.

Pandas [46]: pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Scikit-Learn: Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

4.3 Evaluation Metrics

As the main goal of this task is intrusion detection and network classification, accuracy of the model is not just enough to show the capacity of the model. We also need few other evaluation metrics like precision, recall, confusion matrix, true positive rate, false positive rate, and area under the curve. These definitions are explained as follows:

Precision: precision is the fraction of retrieved documents that are relevant to the query

Recall: recall is the fraction of the relevant documents that are successfully retrieved.

Confusion Matrix: This is also called an error matrix. It is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning classification algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class or vice versa. This is named as a confusion matrix, because using this matrix, we can see if the system is confusing two classes and if it is commonly mislabeling one as another. Following is the terminology used in confusion matrix metrics and is important.

True positive = correctly identified

False positive = incorrectly identified

True negative = correctly rejected

False negative = incorrectly rejected

True Positive Rate: It measures the proportion of actual positives that are correctly identified out of the total positive classified instances. It indicates the percentage of true positives out of the total positive.

True Negative Rate: This measures the proportion of actual negatives that are correctly identified out of the total negative classified instances. It indicates the percentage of true negatives out of the total negatives.

4.4 Results

The experiments are conducted in four ways indicated as follows:

1. Binary classification on dataset with all features
2. Multi-class classification on dataset with all features
3. Binary classification on dataset with selected features
4. Multi-class classification on dataset with selected features

Splitting the dataset: For the purpose of testing, the dataset is split into five different datasets. Each dataset contains 20% for test and 80% for training. Each individual dataset is trained and tuned for hyper-parameters and then applied of the test dataset which is the remaining 20%. The average of all metrics is taken and indicated in the tables. The models are trained on Google Colab with the libraries mentioned above. Following table shows the results of various methods without feature selection. It has all features, and shows that

Random Forest outperforms all other methods and gives best accuracy, precision and recall.

Metric/Method	Naïve Bayes	Logistic Regression	Random Forest	AdaBoost	Gradient Boosting
Precision	0.963	0.887	1.0	0.996	0.999
Recall	0.062	0.983	0.999	0.996	0.999
F1 score	0.116	0.932	0.999	0.996	0.999
Accuracy	24%	88.62%	99.91%	99.37%	99.82%

Table 1: Comparison of Various Methods for Intrusion Detection

From the above tables, we can see that ensemble methods outperform other machine learning algorithms; especially Random Forest takes very less time compared to other ensemble methods. We can conclude that this is the efficient approach for intrusion detection on huge network traffic datasets considering the efficiency of usage of resources. Following table shows the confusion matrix for when all classes of network traffic are considered: Benign, DDoS, PortScan, Bot, Infiltration, Web Attacks, FTP and SSH, DoS, Heartbleed. In the following we show the results of different ensemble methods applied on whole dataset as multi-class classification. The accuracy of Random Forest is 99.89% and that of AdaBoost is 99.88% and Gradient Boosting is 99.72

	Benign	DoS	PortScan	DDoS	FTP	Web Attack	Bot	Infiltration	Heartbleed
Precision	0.99	0.99	0.76	0.85	0.94	0.99	0.99	0.5	0.99
Recall	0.99	0.99	0.83	1	0.99	0.99	0.99	1	0.99
F1 Score	0.99	0.99	0.80	0.92	0.96	0.99	0.99	0.66	0.99

Table 2: Random Forest Metrics for Multi-Class Classification

	Benign	DoS	PortScan	DDoS	FTP	Web Attack	Bot	Infiltration	Heartbleed
Precision	0.99	0.99	0.78	0.42	0.94	0.99	0.99	0.5	0.99
Recall	0.99	0.99	0.79	1	0.96	0.99	0.99	1	0.99
F1 Score	0.99	0.99	0.79	0.6	0.95	0.99	0.99	0.66	0.99

Table 3: AdaBoost Metrics for Multi-Class Classification

Benign	2272688
DoS	251712
PortScan	158930
DDoS	128027
Patator	13835
Web Attack	2180
Bot	1966
Infiltration	36
HeartBleed	11

Table 4: Number of Records in Each Class

It can be seen from the results, the lower precision/recall is because of the less number of samples

Feature Ranking and Importance: Feature selection is an important aspect of classification, as it reduces the training times and also eliminates duplication and feature correlation. Hence, it is important to extract features and study the results on it. Random Forest algorithm can also be used for feature selection for top features that contribute to the classification. Following list shows the list of features of according to their importance scores.

1. Average Packet Size - 0.059870
2. Packet Length Variance - 0.058659
3. Init_Win_bytes_forward - 0.052030
4. Packet Length Mean - 0.049313

5. Bwd Packet Length Mean - 0.048438
6. Max Packet Length - 0.045777
7. Avg Bwd Segment Size - 0.041463
8. Packet Length Std - 0.036997
9. Subflow Fwd Bytes - 0.036723
10. Total Length of Bwd Packets - 0.034983
11. Bwd Packet Length Max - 0.034087
12. Total Length of Fwd Packets - 0.032715
13. Bwd Packet Length Std - 0.031212
14. Subflow Bwd Bytes - 0.025415
15. Bwd Header Length - 0.023039
16. Init_Win_bytes_backward - 0.022483
17. Fwd Header Length.1 - 0.019853
18. Fwd Packet Length Max - 0.019110
19. Fwd Packet Length Mean - 0.017535
20. Fwd Header Length - 0.017184
21. Idle Max - 0.015595
22. Subflow Fwd Packets - 0.014848
23. min_seg_size_forward - 0.014448
24. Flow IAT Max - 0.013813
25. Avg Fwd Segment Size - 0.013166
26. Fwd IAT Max - 0.012664

27. Subflow Bwd Packets - 0.012458

28. Fwd IAT Std - 0.012287

29. Total Backward Packets - 0.012179

30. Fwd IAT Mean0.011560

31. Bwd Packet Length Min - 0.011324

32. Flow IAT Mean - 0.010899

33. Bwd Packets/s - 0.010371

34. Flow Duration - 0.009993

35. Fwd IAT Min - 0.009900

36. Total Fwd Packets - 0.009758

37. Fwd IAT Total - 0.009668

38. ACK Flag Count - 0.009291

39. Flow IAT Std - 0.009291

40. act_data_pkt_fwd - 0.008445

41. Idle Mean - 0.007799

42. PSH Flag Count - 0.007739

43. Fwd Packets/s - 0.007038

44. Flow IAT Min - 0.005220

45. Min Packet Length - 0.004304

46. Fwd Packet Length Min - 0.004213

47. Fwd Packet Length Std - 0.003784

48. Bwd IAT Total - 0.003096

49. Bwd IAT Max - 0.003071

50. URG Flag Count - 0.002949

51. Bwd IAT Mean - 0.001970

52. Active Max - 0.001433

53. Down/Up Ratio - 0.001284

54. Bwd IAT Min - 0.001249

55. Active Std - 0.001201

56. Active Mean - 0.001177

57. FIN Flag Count - 0.001132

58. Bwd IAT Std - 0.000999

59. Active Min - 0.000575

60. Fwd PSH Flags - 0.000483

61. SYN Flag Count - 0.000194

62. Idle Std - 0.000193

63. Fwd URG Flags - 0.000029

64. CWE Flag Count - 0.000022

65. ECE Flag Count - 0.000000

66. RST Flag Count - 0.000000

67. Bwd URG Flags - 0.000000

68. Bwd Avg Bulk Rate - 0.000000

69. Bwd Avg Packets/Bulk - 0.000000

70. Bwd Avg Bytes/Bulk - 0.000000

71. Fwd Avg Packets/Bulk - 0.000000

72. Bwd PSH Flags - 0.000000

73. Fwd Avg Bulk Rate - 0.000000

74. Fwd Avg Bytes/Bulk - 0.000000

75. Flow Bytes/s – 0.000000

76. Flow Packets/s - 0.000000

Following is the metrics for Random Forest after feature selection of top 10 features, which is nearly the square root of total features.

Accuracy	99.4%
Precision	0.995
Recall	0.997
F1-score	0.996

Table 5: Performance metrics of Random Forest after feature selection

Accuracy	95.7%
Precision	0.957
Recall	0.991
F1-score	0.974

Table 6: Performance metrics of AdaBoost after feature selection

Accuracy	96.5%
Precision	0.997
Recall	0.959
F1-score	0.978

Table 7: Performance metrics of Gradient Boosting after feature selection

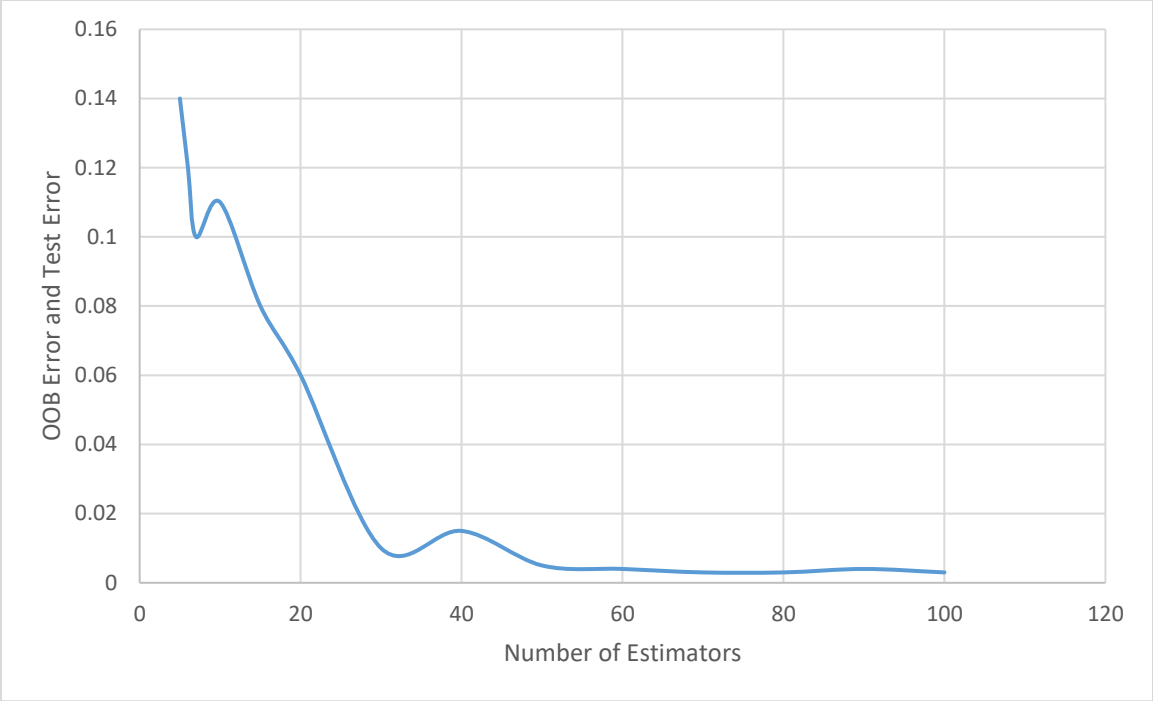

Figure 5: Decrease in Error with Change in Estimators

Following is the list of features ranking score according to Random Forest when all classes are considered for classification:

1. Packet Length Variance - 0.060135
2. Average Packet Size - 0.050755
3. Init Win bytes forward - 0.047369
4. Packet Length Std - 0.045532
5. Bwd Packet Length Std - 0.042913
6. Subflow Fwd Bytes - 0.039410
7. Max Packet Length - 0.038972
8. Bwd Packet Length Mean - 0.034768

9. Total Length of Fwd Packets - 0.033940
10. Subflow Bwd Bytes - 0.029878
11. Total Length of Bwd Packets - 0.029695
12. Fwd Packet Length Max - 0.029362
13. Avg Bwd Segment Size - 0.028650
14. Bwd Packet Length Max - 0.028421
15. Avg Fwd Segment Size - 0.025790
16. Packet Length Mean - 0.024116
17. Fwd Header Length.1 - 0.021989
18. Fwd Packet Length Mean - 0.020529
19. Bwd Packet Length Min - 0.019887
20. Idle Max - 0.019702
21. Flow IAT Max - 0.019023
22. Init_Win_bytes_backward - 0.018273
23. Fwd Header Length - 0.016255
24. Bwd Header Length - 0.015772
25. min_seg_size_forward - 0.015354
26. Fwd IAT Std - 0.015225
27. Total Fwd Packets - 0.014804
28. Bwd Packets/s - 0.013868
29. PSH Flag Count - 0.013584
30. Subflow Fwd Packets - 0.013062

31. Fwd IAT Max - 0.012878

32. Flow IAT Std - 0.012141

33. Fwd Packet Length Std - 0.011951

34. Fwd Packets/s - 0.010991

35. act_data_pkt_fwd - 0.010353

36. Fwd IAT Mean - 0.009543

37. Total Backward Packets - 0.009385

38. Fwd IAT Total - 0.008448

39. Subflow Bwd Packets - 0.007992

40. Fwd Packet Length Min - 0.007897

41. Flow IAT Mean - 0.007104

42. ACK Flag Count - 0.006895

43. Flow Duration - 0.006736

44. Fwd IAT Min - 0.006689

45. Idle Mean - 0.005290

46. Flow IAT Min - 0.004899

47. Min Packet Length - 0.004544

48. Active Min - 0.003649

49. Bwd IAT Mean - 0.002895

50. Down/Up Ratio - 0.002893

51. Active Mean - 0.002864

52. Bwd IAT Total - 0.002742

53. Active Max - 0.002588

54. Bwd IAT Max - 0.002530

55. URG Flag Count - 0.002377

56. Bwd IAT Min - 0.002104

57. FIN Flag Count - 0.001633

58. Bwd IAT Std - 0.000907

59. SYN Flag Count - 0.000714

60. Fwd PSH Flags - 0.000653

61. Active Std - 0.000361

62. Idle Std - 0.000291

63. CWE Flag Count - 0.000017

64. Fwd URG Flags - 0.000013

65. RST Flag Count - 0.000000

66. ECE Flag Count - 0.000000

67. Bwd PSH Flags - 0.000000

68. Bwd URG Flags - 0.000000

69. Bwd Avg Bulk Rate - 0.000000

70. Fwd Avg Packets/Bulk - 0.000000

71. Bwd Avg Packets/Bulk - 0.000000

72. Bwd Avg Bytes/Bulk - 0.000000

73. Fwd Avg Bulk Rate - 0.000000

74. Fwd Avg Bytes/Bulk - 0.000000

Following three tables shows the precision, recall and F1 scores of the ensemble methods used. The accuracy of Random Forest is 99.35, and that of AdaBoost is 99.29 and that of Gradient Boosting is 99.31. We see that Random Forest performs best.

	Benign	DoS	PortScan	DDoS	FTP	Web Attack	Bot	Infiltration	Heartbleed
Precision	0.99	0.98	0.98	0.75	0.90	0.99	0.96	0.98	0.99
Recall	0.99	0.99	0.38	0.6	0.06	0.99	0.97	1	0.99
F1 Score	0.99	0.99	0.55	0.66	0.12	0.98	0.97	1	0.99

Table 8: Random Forest on all classes after feature selection

	Benign	DoS	PortScan	DDoS	FTP	Web Attack	Bot	Infiltration	Heartbleed
Precision	0.99	0.98	0.60	0.75	0.8	0.96	0.97	1	0.99
Recall	0.99	0.99	0.72	0.6	0.06	0.99	0.96	.96	0.99
F1 Score	0.99	0.99	0.65	0.66	0.12	0.98	0.97	1	0.99

Table 9: AdaBoost on all classes after feature selection

	Benign	DoS	PortScan	DDoS	FTP	Web Attack	Bot	Infiltration	Heartbleed
Precision	0.99	0.98	0.95	0.5	0.71	0.96	0.98	1	0.99
Recall	0.99	0.99	0.38	0.6	0.06	0.99	0.97	1	0.99
F1 Score	0.99	0.99	0.54	0.54	0.11	0.98	0.97	1	0.99

Table 10: Gradient Boosting on all classes after feature selection

Results Discussion: From the above results, we can see that as expected, the traditional methods do not perform well, as they suffer from bias and variance and cannot generalize the models for test dataset with high accuracy. The ensemble methods perform very well, and especially random forest stands out of all the methods. Not only that random forest does well in terms of performance, but also training it takes less time compared to other ensemble methods, due to the nature of the independence of the algorithm, and also, it clearly explains that the other boosting methods are prone to a bit of overfitting when compared to bagging approaches. Random forest performs well both on intrusion detection

as binary classification and network traffic classification. On the subset of features, the ensemble methods perform well, and random forest performs the best even with only ten percent of the entire features. This has reduced a dramatic training time without compromising the accuracy. Hence, we can say that ensemble methods perform very well compared to the traditional methods in terms of both performance and training times.

Chapter 5

CONCLUSION AND FUTURE WORK

In this thesis, an approach using ensemble methods for intrusion detection with network traffic data has been presented. This approach can be generalized to any application, as the network traffic does not vary. The dataset used for training the model is comprehensive, and provides an up-to-date attack scenarios at network level traffic, with imbalanced classes as well. Previous works and datasets did not address these issues. In this thesis, and analysis of ensemble methods is also done to give new insights to the intrusion detection approach as an anomaly detection approach. The advantages of ensemble methods, especially as highlighted, they are robust to outliers, feature scaling and missing values. From the experiments, it can be seen that the ensemble methods outperform traditional methods on this kind of complex dataset and Random Forest is the best classifier in terms of accuracy and the feature selection for dataset size reduction. One interesting fact about Random Forest is that it requires minimum hyper-parameter tuning and just selecting the number of trees in the forest.

As mentioned, one of the problems with ensemble methods is they may take longer time to test as they have aggregate the results from various small classifiers. This can be handled by using distributed computing approaches like Apache Spark. The models can be trained faster, and robust to node failure and provides fault tolerant computing approach. Also, the model can be saved in the memory as the size of the model is not huge. This provides speed up of hundreds of times in real-time testing if a network traffic is intrusion or not.

REFERENCES

1. A. Mayank, P. Sanketh, B. Santosh, N Sukumar, "Intrusion detection system for PS-Poll DoS attack in 802.11 networks using real time discrete event system", *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 4, pp. 792-808, 2017.

2. E. Denning Dorothy, "An intrusion detection model", *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222-232, 1987.

3. P. Shengyi, M. Thomas, A. Uttam, "Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems", *IEEE Transactions on Smart Grid*, vol. 6, no. 6, pp. 3104-3113, 2015.

4. D. He, X. Chen, D. Zou, L. Pei and L. Jiang, "An Improved Kernel Clustering Algorithm Used in Computer Network Intrusion Detection," 2018 IEEE International Symposium on Circuits and Systems ISCAS, Florence, 2018, pp. 1-5

5. E. Kabir, J. Hu, H. Wang, G. Zhuo A novel statistical technique for intrusion detection systems Future Gener. Comput. Syst., 79 2018, pp. 303-318

6. V. Adat, B.B. Gupta, Security in internet of things: issues, challenges, taxonomy, and architecture, Telecommun. Syst., 67 3 2018, pp. 423-441

7. M. Ahmed, A.N. Mahmood, J. Hu, A survey of network anomaly detection techniques J. Netw. Comput. Appl., 60 2016, pp. 19-31

8. L.M. Rocha, F.A.M. Cappabianco, A.X. Falcão, Data clustering as an optimum-path forest problem with applications in image analysis. Int. J. Imaging Syst. Technol., 19 2 2009, pp. 50-68

9. C.R. Pereira, R.Y.M. Nakamura, K.A.P. Costa, J.P. Papa An optimum-path forest framework for intrusion detection in computer networks Eng. Appl. Artif. Intell., 25 6 2012, pp. 1226-1234

10. K.A.P. Costa, Pereira, R.Y.M. Nakamura, C.R. Pereira, J.P. Papa, A.X. Falcão, A nature-inspired approach to speed up optimum-path forest clustering and its application to intrusion detection in computer networks. Inf. Sci., 294 2015, pp. 95-108

11. J.P. Papa, A.X. Falcão, C.T.N. Suzuki, Supervised pattern classification based on optimum-path forest. Int. J. Imaging Syst. Technol., 19 2 2009, pp. 120-131

12. J.P. Papa, A.X. Falcão, V.H.C. Albuquerque, J.M.R.S. Tavares. Efficient supervised optimum-path forest classification for large datasets. Pattern Recognit., 45 1 2012, pp. 512-520
13. J.P. Papa, G.H. Rosa, L.P. Papa, A binary-constrained geometric semantic genetic programming for feature selection purposes. Pattern Recognit. Lett., 100 Supplement C 2017, pp. 59-66

14. F. Javed, M.K. Afzal, M. Sharif, B. Kim, Internet of things IoTs operating systems support, networking technologies, applications, and challenges: a comparative review. IEEE Commun. Surv. Tut. 2018, p. 1

15. B. Arrington, L. Barnett, R. Rufus, A. Esterline. Behavioral modeling intrusion detection system BMIDS using internet of things IoT behavior-based anomaly detection via immunity-inspired algorithms. 25th International Conference on Computer Communication and Networks ICCCN, IEEE 2016

16. S. Alharbi, P. Rodriguez, R. Maharaja, P. Iyer, N. Bose, Z. Ye, FOCUS: a fog computing-based security system for the internet of things. 15th IEEE Annual Consumer Communications & Networking Conference CCNC, IEEE 2018

17. D. Evans, The internet of things: how the next evolution of the internet is changing everything. Cisco White Paper 2011, pp. 1-11

18. H. Bostani, M. Sheikhan. Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on map-reduce approach. Comput. Commun., 98 Supplement C 2017, pp. 52-71

19. https://www.unb.ca/cic/datasets/ids-2018.html

20. P. Ramchandani, Random Forests and the Bias-Variance Tradeoff, Towards Data Science

21. Opitz, D.; Maclin, R. 1999. "Popular ensemble methods: An empirical study". Journal of Artificial Intelligence Research. 11: 169–198

22. Ho, Tin Kam 1995. Random Decision Forests PDF. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282

23. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#workings

24. L. Santos, C. Rabadao and R. Gonçalves, "Intrusion detection systems in Internet of Things: A literature review," 2018 13th Iberian Conference on Information Systems and Technologies CISTI, Caceres, 2018, pp. 1-7.

25. E. Cho, J. Kim, and C. Hong, “Attack model and detection scheme for botnet on 6LoWPAN,” In Management Enabling the Future Internet for Changing Business and New Computing Services, Lecture Notes in Computer Science 5787. Springer, Berlin, Heidelberg, 515-518, 2009.

26. A. Le, J. Loo, Y. Luo, and A. Lasebae, “Specification-based IDS for securing RPL from topology attacks,” In: Wireless Days WD, 2011 IFIP, pp. 1-3, 2011.

27. C. Liu, J. Yang, Y. Zhang, R. Chen, and J. Zeng, “Research on immunitybased intrusion detection technology for the Internet of Things,” In: Natural Computation ICNC, 2011 Proceedings of the Seventh International Conference, Vol. 1, pp. 212-216, 2011

28. S. Misra, P. Krishna, H. Agarwal, A. Saxena, and M. Obaidat, “A learning automata-based solution for preventing Distributed Denial of Service in Internet of Things,” In: Internet of Things iThings/CPSCom, 2011 International Conference on and Proceedings of the 4th International Conference on Cyber, Physical and Social Computing, pp. 114-122, 2011.

29. A. Gupta, O. Pandey, M. Shukla, A. Dadhich, S. Mathur, and A. Ingle, “Computational intelligence-based intrusion detection systems for wireless communication and pervasive computing networks,” In: Computational Intelligence and Computing Research ICCIC, 2013 IEEE International Conference on, pp. 1-7, 2013.

30. M. Rebbah, D. El Hak Rebbah and O. Smail, "Intrusion detection in Cloud Internet of Things environment," 2017 International Conference on Mathematics and Information Technology ICMIT, Adrar, 2017, pp. 65-70.

31. IEEE 2015 Standards, Internet of Things, IEEE P2413.

32. Bandyopadhyay D, Sen J 2011 Internet of things: Applications and challenges in technology and standardization. Wirel Pers Commun 581:49–69.

33. Han C, Jornet JM, Fadel E, Akyildiz IF 2013 A cross-layer communication module for the internet of things. Comput Netw 573:622–633.

34. Khan R, Khan S, Zaheer R, Khan S 2012 Future internet: The internet of things architecture, possible applications and key challenges In: 2012 10th International Conference on Frontiers of Information Technology, 257–260.. IEEE, Islamabad.

35. Rubio-Loyola J, Sala D, Ali AI 2008 Accurate real-time monitoring of bottlenecks and performance of packet trace collection In: 2008 33rd IEEE Conference on Local Computer Networks LCN, 884–891.. IEEE, Montreal

36. Rubio-Loyola J, Sala D, Ali AI 2008 Maximizing packet loss monitoring accuracy for reliable trace collections In: 2008 16th IEEE Workshop on Local and Metropolitan Area Networks, 61–66.. IEEE, Chij-Napoca.

37. Ghorbani AA, Lu W, Tavallaee M 2010 Network Intrusion Detection and Prevention, Advances in Information Security, vol. 47. Springer, US.

38. Anwar S, Mohamad Zain J, Zolkipli MF, Inayat Z, Khan S, Anthony B, Chang V 2017 From intrusion detection to an intrusion response system: Fundamentals, requirements, and future directions. Algorithms 102:1–24.

39. Denning DE 1987 An intrusion-detection model. IEEE Trans Softw Eng SE-132:222–232.

40. Creech G, Hu J 2014 A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. IEEE Trans Comput 634:807–819.

41. Kumar S, Gautam, Om H 2016 Computational neural network regression model for host based intrusion detection system. Perspect Sci 8:93–95.

42. Snort The Open Source Network Intrusion Detection System. https://www.snort.org. Accessed 1 Nov 2016.

43. Wikipedia Contributors, AdaBoost, from Wikipedia the free Encyclopedia, April 1, 2019 https://en.wikipedia.org/wiki/AdaBoost

44. Wikipedia Contributors, Gradient Boosting, from Wikipedia the free Encyclopedia, April 1, 2019, https://en.wikipedia.org/wiki/Gradient_boosting

45. http://www.numpy.org/

46. https://pandas.pydata.org/

47. Medium.com, Will Koehersen, Random Forest Simple Explanation, Dec 27, 2017

48. J. J. T. Zhenwei Yu, A Framework of Machine Learning Based Intrusion Detection for Wireless Sensor Networks, IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, vol. 6, no. 9, pp. 272–279, (2008).

49. T. M. Mitchell, Machine Learning, 1st ed, New York, NY, USA: McGraw-Hill, Inc., (1997).

50. G. Poojitha, K. N. Kumar and P. J. Reddy, Intrusion Detection Using Artificial Neural Network, In 2010 International Conference on Computing Communication and Networking Technologies (ICCCNT), pp. 1–7, July (2010).

51. H. Altwaijry and S. Algarny, Bayesian Based Intrusion Detection System, Journal of King Saud University – Computer and Information Sciences, vol. 24, no. 1, pp. 1–6, (2012).

52. 52. M. Panda and M. R. Patra, Semi-naive Bayesian Method for Network Intrusion Detection System, In Neural Information Processing, 16th International Conference, ICONIP 2009, Bangkok, Thailand, December 1–5, 2009, Proceedings, Part I, pp. 614–621, (2009).

53. 53. G. Nadiammai and M. Hemalatha, Effective Approach Toward Intrusion Detection System Using Data Mining Techniques, Egyptian Informatics Journal, vol. 15, no. 1, pp. 37–50, (2014).

54. P. Sangkatsanee, N. Wattanapongsakorn and C. Charnsripinyo, Practical Real-Time Intrusion Detection Using Machine Learning Approaches, Computer Communications, vol. 34, no. 18, pp. 2227–2235, (2011).

55. L. M. L. Campos, R. C. L. de Oliveira and M. Roisenberg, Network Intrusion Detection System Using Data Mining, Engineering Applications of Neural Networks: 13th International Conference, EANN 2012, London, UK, September 20–23, 2012.

56. L. Koc, T. A. Mazzuchi and S. Sarkani, A Network Intrusion Detection System Based on a Hidden Naive Bayes Multiclass Classifier, Expert Systems with Applications, vol. 39, no. 18, pp. 13 492–13 500, (2012).

57. Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai and K. Dai, An Efficient Intrusion Detection System Based on Support Vector Machines and Gradually Feature Removal Method, Expert Systems with Applications, vol. 39, no. 1, pp. 424–430, (2012).

58. S. S. S. Sindhu, S. Geetha and A. Kannan, Decision Tree Based Light Weight Intrusion Detection Using a Wrapper Approach, Expert Systems with Applications, vol. 39, no. 1, pp. 129–141, (2012).

59. I. Butun, S. D. Morgera and R. Sankar, A Survey of Intrusion Detection Systems in Wireless Sensor Networks, Communications Surveys and Tutorials IEEE, vol. 16, pp.

266–282, (2013).

60. A. H. Farooqi, F. A. Khan, J. Wang and S. Lee, A Novel Intrusion Detection Framework for Wireless Sensor Networks, Personal and Ubiquitous Computing, vol. 17, no. 5, pp. 907–919, (2013).

61. S.-S. Wang, K.-Q. Yan, S.-C. Wang and C.-W. Liu, An Integrated Intrusion Detection System for Cluster-Based Wireless Sensor Networks, Expert Syst. Appl., vol. 38, no. 12, pp. 15 234–15 243, (2011).