Comparison of Intrusion Detection Systems for Use in Low-Powered Devices

A project report submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfillment of the Requirement for the

Award of the Degree

of

Bachelor of Technology

in

Computer and Communication Engineering

by

Pratyay Amrit

Reg. No. 140953430

Under the guidance of

Ms. Ipsita Upasana
Assistant Professor
Department of I & CT
Manipal Institute of Technology
Manipal, India

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

INSPIRED BY LIFE

MAY 2018

I dedicate my thesis to my friends and family.

DECLARATION

I hereby declare that this project work entitled **Comparison of Intrusion Detection Systems for Use in Low-Powered Devices** is original and has been carried out by me in the Department of Information and Communication Technology of Manipal Institute of Technology, Manipal, under the guidance of **Ms. Ipsita Upasana**, **Assistant Professor**, Department of Information and Communication Technology, M. I. T., Manipal. No part of this work has been submitted for the award of a degree or diploma either to this University or to any other Universities.

Place: Manipal

Date : 05-05-18

Pratyay Amrit

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

CERTIFICATE

This is to certify that this project entitled **Comparison of Intrusion Detection Systems for Use in Low-Powered Devices** is a bonafide project work done by **Mr. Pratyay Amrit (Reg. No.: 140953430)** at Manipal Institute of Technology, Manipal, independently under my guidance and supervision for the award of the Degree of Bachelor of Technology in Computer and Communication Engineering.

Ms. Ipsita Upasana
Assistant Professor
Department of I & CT
Manipal Institute of Technology
Manipal, India

Dr.Balachandra
Professor & Head
Department of I & CT
Manipal Institute of Technology
Manipal, India

ACKNOWLEDGEMENTS

My sincere thanks to Ms. Ipsita Upasana, and all faculty members of the Department of Information & Communcation Technology for providing necessary guidance and feedback throughout the course of the project.

ABSTRACT

Intrusion Detection Systems have become an essential part of computer network security. It acts as a first-line defense from cyber-attacks by analyzing the various attributes of a data packet and identifying it as a malicious, or an ordinary one. Such systems have come a long way and in the present time, promise acceptable performance. However, the algorithms that run at the core of these systems are computationally expensive, making them fairly accurate, but almost unimplementable in very low powered devices, such as nodes of a WSAN, or in an IoT based setup. This work attempts to rank various machine learning and data mining techniques on the basis of their accuracy and time required to train and classify network data.

[**Security and Privacy**]: Intrusion/Anomaly Detection and Malware Mitigation–Intrusion detection systems

Contents

List of Tables

List of Figures

ABBREVIATIONS

IDS	Intrusion Detection System
UNSW-NB15	University of New South Wales, Network Based 2015 Dataset
KDD'99	Knowledge Discovery and Data Mining Competition 1999 Dataset
SIEM	System Information and Event Management
KNN	K-Nearest Neighbors
NB	Gaussian Naive Bayes
DT	Decision Tree
DT_n	Decision Tree with max-depth = n
RF_n	Random Forests with max-depth = n
ET_n	Extra Trees with max-depth = n
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TTF	Time to Fit
TTP	Time to Predict
TS	Training Score
PS	Prediction Score

Chapter 1

Introduction

1.1 Background

An Intrusion Detection System (IDS) is an application that can be installed on a system or a network. The application may be a piece of software, or a device that can be plugged into the interface as a module. The IDS, once plugged into the system, scans for activity, and based on various features pertaining to the activity, classifies it as normal, or malicious. It must be noted that the IDS itself does not provide any access control, and is only responsible for the *detection* of an attack. As a result, IDSs must be bundled with other tools to prevent an attack. Typically, all detected malicious activity is reported to a Security Information and Event Management (SIEM) system, which combines reports from multiple sources and uses alarm filtering techniques to distinguish malicious activity from false alarms.

Many attempts have been made to classify various kinds of IDSs. According to [1], on the basis of method of detection, two types of IDSs have been identified:

- **Knowledge Based**: Knowledge Based IDSs accumulate knowledge about attacks and look for similar data to detect an attack. Since these IDSs look for very specific signatures, they provide very accurate detection of common attacks. However, in a case where an attack has a signature that has never been seen before, this kind of system fails. This type of IDS is also called **misuse detection** IDS.

- **Behavior Based**: Behavior Based IDSs work with the assumption that attacks can be detected if the behavior of the system or the users deviate from the normal expected pattern. Many sources are combined to gather enough data to teach a system what normal behavior looks like and to create a model. The IDS then compares the current activity with this model and reports if it detects an anomaly. These IDSs are also called **anomaly detection** IDSs. Since these IDSs are based on predictive models, they tend to have lower accuracy, and higher false alarm rate when compared to a knowledge based IDS. However, they are effective against novel attacks as any kind of attack is detected as an anomaly, or a deviation from the norm.

A third type of IDS is also often seen in literature, called the **hybrid detection** IDS. This is a combination of the two types. Typically, an activity is first passed through a knowledge based system. If the first filter claims the activity to be an attack, the system terminates and the activity is reported. If the activity passes the first filter, it is passed through a behavior based system, where even if the signature could not be found in the dictionary of attacks, an anomaly is detected and the activity is reported. The signature of this activity is then recorded in the knowledge based system for future reference. An activity is not reported as malicious only if it passes both of the filters. Although the process of identification is often sped up compared to the other two types of IDSs, the accuracy is bottle-necked by the behavior based system used. Running such a system with a poor behavior based system also possess the risk of saving false signatures in the initial filter. If an activity passes through the first filter, and the second filter raises a false alarm, the signature of the (in reality) normal activity is recorded as malicious, and is reported in subsequent instances of similar activities.

Thus, choosing the right behavior based model becomes an important part of building an IDS. Knowledge based systems are not future proof, and it is often too late for a system or a network to be affected by an attacker even once, regardless of how well the IDS reports subsequent activities of the same. Hybrid systems may provide quick classification, but are bottle-necked by the high false positive rate of the behavior based model used.

1.2 Objectives

The broad objective of this study was to compare different machine learning and data mining models on the basis of their accuracy as well as execution time.
Specifically, the objectives can be broken down to the following:

- To determine useful features in classifying network data.
- To find appropriate measures to score the different models.
- To study the causes of different models performing differently.
- To open scope for future studies to improve the given score.

1.3 Limitations of Study

- The study has been conducted with a static database for training and testing. Although the dataset has been curated fairly recently (2015), it is at best, a rough approximate of live data flowing through a network.

- Many complex algorithms (such as deep neural networks, multi-layer perceptron etc.) which have proved to be very accurate have not been tested simply because the execution time of these algorithms were too high for it to fetch a comparable score.

- Although [2] suggests a faster and efficient way to implement a multi-layer perceptron with binary weights, it has also claimed a reduction of accuracy by up to 4 times for the same, and hence has not been considered in the comparison.

Chapter 2

Literature Review

2.1 Taxonomy for Intrusion Detection Systems

In [1], an attempt has been made to standardize a terminology for IDSs. Many different types of common in use IDSs are introduced in the paper, along with some upcoming possibilities.

Broadly, IDSs have been classified on the basis of 5 different features (Figure 2.1).

1. On the basis of detection method:
   - Behavior based
   - Knowledge based
2. On the basis of behavior on detection:
   - Passive filtering
   - Active filtering
3. On the basis of audit source location:
   - Host log files
   - Network packets

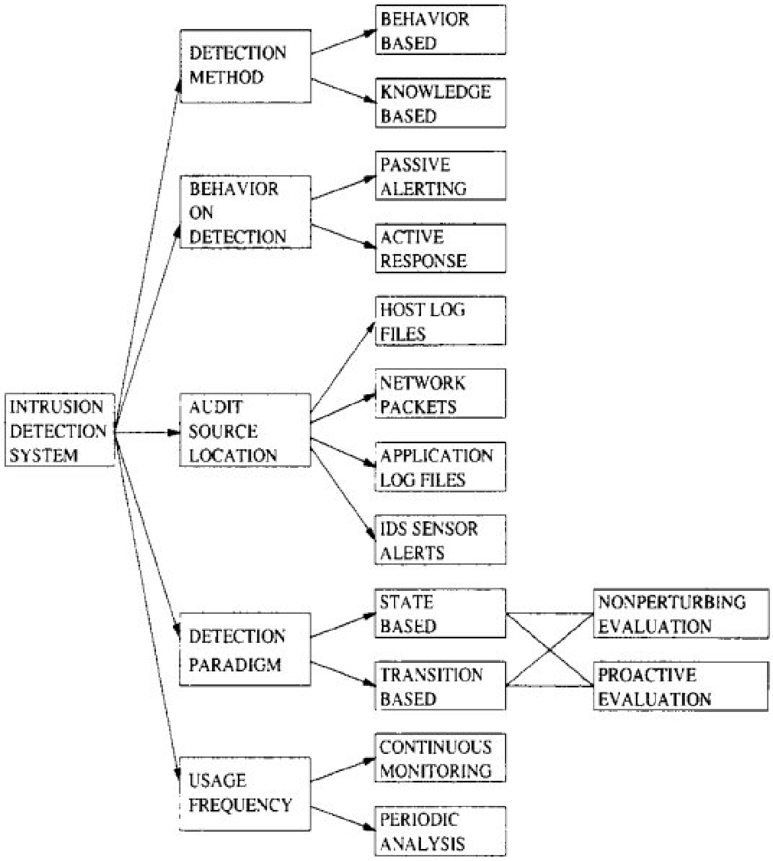

Figure 2.1: Types of IDS as mentioned in [1]

- Application log files
- IDS sensor alerts

4. On the basis of detection paradigm:

   - State based
   - Transition based

5. On the basis of usage frequency:

   - Continuous monitoring
   - Periodic analysis

This work focuses only on behavior based IDSs. Various methods have been proposed using all kinds of machine learning models, some of which have been tested and compared in this work.

2.2 Datasets

When it comes to training and testing IDSs, the KDD'99 [3] and UNSW-NB15 [4] datasets are usually considered.

- **KDD'99**: The KDD'99 dataset was curated in 1999 and was used for The Third International Knowledge Discovery and Data Mining Tools Competition 1999. It holds its popularity till date for training and testing models based on data mining and machine learning algorithms for extracting information out of network packet data. The dataset contained 41 features along with a label mentioning the type of attack detected. The dataset is a raw dump of network traffic, and barely processed for use in machine learning.

- **UNSW-NB15**: The UNSW-NB15 dataset was generated in 2015, and similar to KDD'99, is a dump of network traffic. However, it has been proccessed to fix the problems of KDD'99 [5]. The dataset has a total of 49 features, including 2 class labels and over 2 million records. Since the dataset was generated more recently (2015), it is a more accurate example of network data traffic that an IDS might face in the current times. The class labels have classified 9 different kinds of attacks, some of which were not present in the KDD'99 dataset.

The UNSW-NB15 seemed to be a better dataset to use of this work, owing to its novelty and problems in KDD'99 that it addresses [5].

2.3 UNSW-NB15

The UNSW-NB15 dataset was released in 2015. It was generated using tcpdump tool to capture about 100GB of raw data. It has the following 9 types of attacks:

1. Fuzzers: Fuzzing, or Fuzz Testing is a black box testing technique which aims at finding implementation bugs by injecting malformed or semi-malformed data in an automated fashion. Potential attackers may run fuzz tests on the network to find vulnerabilities that can be exploited.

2. Analysis: Traffic analysis is a special type of inference attack technique that looks at communication patterns between entities in a system. Knowing who's talking to whom, when, and for how long, can sometimes clue an attacker in to information of which you'd rather she not be aware.

3. Backdoor: A backdoor is a malware type that negates normal authentication procedures to access a system.
   Webserver backdoors are used for a number of malicious activities, including:

   - Data theft
   - Website defacing
   - Server hijacking
   - The launching of distributed denial of service (DDoS) attacks
   - Infecting website visitors (watering hole attacks)
   - Advanced persistent threat (APT) assaults

4. DoS: Denial of Service (DoS) is an attack which aims to make a host machine unavailable to its intended users by temporarily or indefinitely

disrupting the host machines connection to the internet.

Broadly, DoS attacks can be classified into 3 types:

- Volume Based Attacks: Includes UDP floods, ICMP floods, and other spoofed-packet floods. The attacks goal is to saturate the bandwidth of the attacked site, and magnitude is measured in bits per second (Bps).
- Protocol Attacks: Includes SYN floods, fragmented packet attacks, Ping of Death, Smurf DDoS and more. This type of attack consumes actual server resources, or those of intermediate communication equipment, such as firewalls and load balancers, and is measured in packets per second (Pps).
- Application Layer Attacks: Includes low-and-slow attacks, GET/POST floods, attacks that target Apache, Windows or OpenBSD vulnerabilities and more. Comprised of seemingly legitimate and innocent requests, the goal of these attacks is to crash the web server, and the magnitude is measured in Requests per second (Rps).

5. Exploit: An exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic.

6. Generic: This subset contains any general type of attack such as:

    - Probe, which is a program or a device inserted into the network to monitor the network and collect data.
    - User to Root Attacks (U2R), in which an attacker or a hacker tries to get the access rights from a normal host in order, for instance, to gain the root access to the system.

- Remote to Local Attacks (R2L), in which a remote user sends data packets to a machine over the internet, which he/she does not have the access to, in order to expose vulnerabilities and exploit privileges which a local user would have on the computer.

7. Reconnaissance: Active reconnaissance is a type of computer attack in which an intruder engages with the targeted system to gather information about vulnerabilities.

8. Shellcode: A shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability.

9. Worm: A computer worm is a standalone malware computer program that replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it.

The dataset contains 2,540,044 records with 49 features categorized into 6 categories.

1. **Flow Features:**

   1. srcip: Source IP address.
   2. sport: Source port number.
   3. dstip: Destinations IP address.
   4. dsport: Destination port number.
   5. proto: Protocol type, such as TCP, UDP.

2. **Basic Features:**

   6. state: The states and its dependent protocol e.g., CON.
   7. dur: Row total duration.
   8. sbytes: Source to destination bytes.
   9. dbytes: Destination to source bytes.

10. sttl: Source to destination time to live.
11. dttl: Destination to source time to live.
12. sloss: Source packets retransmitted or dropped.
13. dloss: Destination packets retransmitted or dropped.
14. service: Such as http, ftp, smtp, ssh, dns and ftp-data.
15. sload: Source bits per second.
16. dload: Destination bits per second.
17. spkts: Source to destination packet count.
18. dpkts: Destination to source packet count.

3. **Content Features:**

19. swin: Source TCP window advertisement value.
20. dwin: Destination TCP window advertisement value.
21. Stcpb: Source TCP base sequence number.
22. dtcpb: Destination TCP base sequence number.
23. smeansz: Mean of the packet size transmitted by the srcip.
24. dmeansz: Mean of the packet size transmitted by the dstip.
25. trans_depth: The connection of http request/response transaction.
26. res_bdy_len: The content size of the data transferred from http.

4. **Time Features:**

27. sjit: Source jitter.
28. djit: Destination jitter.
29. stime: Row start time.
30. ltime: Row last time.
31. sintpkt: Source inter-packet arrival time.
32. dintpkt: Destination inter-packet arrival time.
33. tcprtt: Setup round-trip time, the sum of synack and ackdat.
34. synack: The time between the SYN and the SYN_ACK packets.
35. ackdat: The time between the SYN_ACK and the ACK packets.

36. is_sm_ips_ports: If srcip (1) = dstip (3) and sport (2) = dsport (4), assign 1 else 0.

5. **Additional Generated Features:**

   37. ct_state_ttl: No. of each state (6) according to values of sttl (10) and dttl (11).

   38. ct_flw_http_mthd: No. of methods such as Get and Post in http service.

   39. is_ftp_login: If the ftp session is accessed by user and password then 1 else 0.

   40. ct_ftp_cmd: No of flows that has a command in ftp session.

   41. ct_srv_src: No. of rows of the same service (14) and srcip (1) in 100 rows.

   42. ct_srv_dst: No. of rows of the same service (14) and dstip (3) in 100 rows.

   43. ct_dst_ltm: No. of rows of the same dstip (3) in 100 rows.

   44. ct_src_: ltm No. of rows of the srcip (1) in 100 rows.

   45. ct_src_dport_ltm: No of rows of the same srcip (1) and the dsport (4) in 100 rows.

   46. ct_dst_sport_ltm: No of rows of the same dstip (3) and the sport (2) in 100 rows.

   47. ct_dst_src_ltm: No of rows of the same srcip (1) and the dstip (3) in 100 records.

6. **Labeled Features:**

   48. attack_cat: The name of each attack category.

   49. label: 0 for normal and 1 for attack records.

2.4 Algorithms Used

K-Nearest Neighbors is an algorithm used for classification or regression. It takes the features and labels as inputs to train the model. Once trained, the classification is done on the basis of a majority vote. An object is classified as the class of the majority vote from the nearest k vectors in the feature space. The parameter k must be adjusted to specific use case scenarios. KNN is implemented in the scikit-learn library [6] as sklearn.neighbors.KNeighborsClassifier.

Naive Bayes algorithm builds a conditional probability model with the assumption that all features are independent and have no correlation. Despite this, rather unrealistic assumption and seemingly oversimplified design, studies have shown that the algorithm is quite optimal [7].

Decision Tree is a method that generates tree-like model of decisions and their consequences. Various metrics, such as information gain, gini impurity etc. are used to decide which feature to split at every level in the tree. Scikit-learn library [6] implements decision tree in the class sklearn.tree.DecisionTreeClassifier. The choice of the metric to split the tree is given as a parameter in the class constructor.

Random Forest [8] is an ensemble method in which many small decision trees are created at training time, and classification is done on the basis of a majority vote among the individual trees to decide on an object. This method addresses the issue of over-fitting in decision trees.

Extra Trees is another ensemble method very similar to Random Forests. Studies suggest [9] that Extra Trees tend to perform a bit worse when there is a high number of noisy features.

Chapter 3

Methodology

3.1 Environment

Since the time of execution has been focused on in this work, it is important to note the environment in which the tests have been conducted. Table 3.1 shows the system configuration in which the tests have been conducted. Since the focus is on low powered devices, GPU acceleration has not been used in any of the tests.

Table 3.1: System Specifications

OS	Arch Linux
CPU	Intel Core i5-7200U @ 4x 3.1GHz
GPU	NVIDIA GeForce 940MX
RAM	8GB

Python language was used to conduct the tests using the interactive environment provided by **IPython**. The **scikit-learn** library has been extensively used for training and testing the models. **Pandas** library has been used alongwith **numpy** to clean and prepare the dataset for processing.

3.2 Workflow

To achieve the objectives of the project, the following workflow was planned:

1. Find a dataset that meets the present standards
2. Import, segment and clean the dataset as per requirements
3. Find important features in the datasets
4. Decide on performance parameters to use for comparison
5. Run tests on the dataset with multiple algorithms
6. Compare the results

3.3 Dataset

Two popular datasets were found for the purpose of training and testing IDSs, the KDD'99 [3] and the UNSW-NB15 [4]. KDD'99 presented various problems, such as redundancy, unbalanced sets etc., as discussed thoroughly in [5]. UNSW-NB15, besides being a more recent dataset, hence meeting the present standards in types of data packet and more accurate labels, also addressed issues in the KDD'99 dataset. Hence, the choice of UNSW-NB15 was made for this work.

3.4 Cleaning the Dataset

The UNSW-NB15 dataset contains over 2.5 million records, and is split into 4 subsets, each containing 700,000 records. The first subset has been used for training the models, and the second for testing throughout the work. Each subset goes through the following steps before it can be used for building the models:

1. Import dataset using python pandas library
2. Fill empty values (attack_cat = ” for normal data) with 0
3. Transform nominal data values to numeric
4. Add feature names to the pandas dataframe (only for readability while debugging)

3.5 Feature Selection

Decision Tree algorithms in scikit-learn library assign an importance score to each feature on the basis of how much they contribute in the classification. The ExtraTreesClassifier class was used to fetch these importance scores (Figure 3.1)

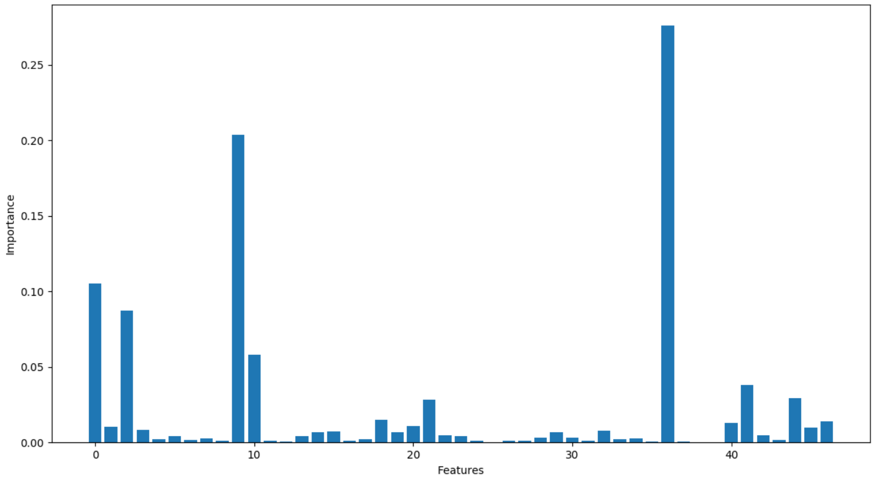

Figure 3.1: Feature importance

It was observed that 2 features had radically high importance scores (sttl and ct_state_ttl). The first one is the source to destination time to live, which makes sense as it is often manipulated to run various kinds of attacks, and the second is an additional generated feature dependent on state, sttl and dttl.

The two other peaks in the graph (Figure 3.1), at x = 0 and 2 are source and destination IP addresses. It makes sense that they have a high score as attacks or normal data are likely to originate from the same source and go to the same destination in the dataset curated at a lab at a university, but it is not very useful to build a generic model to classify an attack as it is a property of the source, and not the kind of data packet.

3.6 Performance Parameters

A confusion matrix is used to define accuracy of the models. A confusion matrix has 4 parameters:

- True Positive (TP): Number of correctly classified attack records
- True Negative (TN): Number of correctly classified normal records
- False Positive (FP): Number of misclassified attack records
- False Negative (FN): Number of misclassified normal records

Using these 4 parameters, the accuracy may be calculated as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

To factor in the time of execution, two new parameters are introduced. Since the objective is to maximize accuracy and minimize time of execution, the final score of a model is defined as:

$$score \propto \frac{accuracy}{execution\_time} \tag{3.2}$$

With the proportionality constant as 1, the Training Score (TS) and Prediction Score (PS) are defined as:

$$TS = \frac{accuracy}{time\_to\_fit/train} \tag{3.3}$$

$$PS = \frac{accuracy}{time\_to\_predict/classify} \tag{3.4}$$

Chapter 4

Results and Observations

The KNN Classifier was tested with the value of k = 1 to reduce the number of computations as much as possible to fasten the process. All decision tree based classifiers have been tested with max_depth = 2 and 4. This parameter defines how deep the tree can build itself. In the case of DecisionTreeClassifier, max_depth = None has also been considered. The execution time and accuracy for each model is presented in Table 4.1 and Table 4.2

Table 4.1: Execution times of different models with all features

Model	**TTF (sec)**	**TTP (sec)**	**Accuracy**
KNeighborsClassifier(n_neighbors = 1)	78.721	54.251	0.9519
GaussianNB()	0.752	0.77	0.9181
DecisionTreeClassifier()	3.76	0.3	0.9746
DecisionTreeClassifier(max_depth = 2)	3.463	0.295	0.9873
DecisionTreeClassifier(max_depth = 4)	3.781	0.32	0.9876
RandomForestClassifier(max_depth = 2)	4.666	0.472	0.9885
RandomForestClassifier(max_depth = 4)	7.144	0.521	0.99
ExtraTreesClassifier(max_depth = 2)	1.167	0.489	0.9276
ExtraTreesClassifier(max_depth = 4)	1.604	0.551	0.9617

Table 4.2: Execution times of different models with 2 selected features

Model	**TTF (sec)**	**TTP (sec)**	**Accuracy**
KNeighborsClassifier(n_neighbors = 1)	0.909	0.192	0.9895
GaussianNB()	0.073	0.039	0.9860
DecisionTreeClassifier()	0.057	0.013	0.9895
DecisionTreeClassifier(max_depth = 2)	0.075	0.023	0.98733
DecisionTreeClassifier(max_depth = 4)	0.195	0.013	0.9895
RandomForestClassifier(max_depth = 2)	0.902	0.192	0.9885
RandomForestClassifier(max_depth = 4)	1.027	0.219	0.9895
ExtraTreesClassifier(max_depth = 2)	0.55	0.198	0.9826
ExtraTreesClassifier(max_depth = 4)	0.654	0.221	0.9895

To understand the difference in the performance of these models, a better visualization of the data in Table 4.1 and Table 4.2 is presented in Figure 4.1, Figure 4.2 and Figure 4.3.

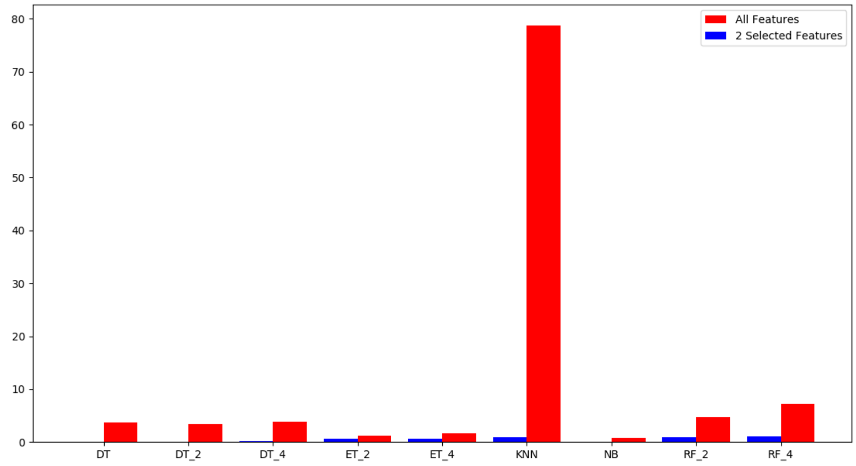

Figure 4.1: Change in TTF of different models (lower is better)

It is clear from these results that selecting the 2 features in Section 3.5 resulted in radically reduced execution times, especially for KNN (77.812 sec-

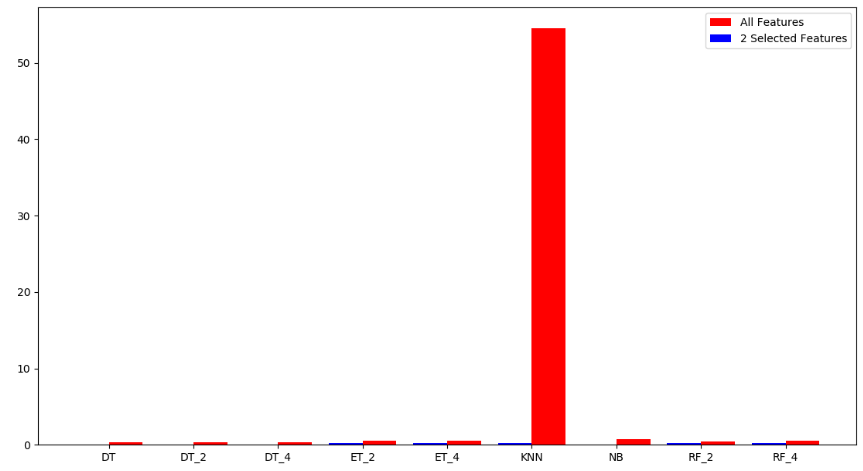

Figure 4.2: Change in TTP of different models (lower is better)

onds reduced), without negatively affecting to the accuracy of the models. Most models see only slight change in accuracy, but the change itself is usually positive. The reduction in execution time resulted in significant improvement in the Training and Prediction Scores (Table 4.3).

The data in Table 4.3 is presented in Figure 4.5 and Figure 4.4 for visualizing the change in the Training and Prediction Scores. It is evident from these results that the two selected features in the dataset are indeed contributing the most in the classification. Accuracy and execution time improved upon ignoring the other features. The time reduction is simply because there are less dimensions in the feature space for the models to process, while the accuracy improvement may suggest that the other features had a slight negative contribution in the classification. A closer look at the confusion matrix results (Table **??**) suggests that, although the overall accuracy has improved, the false positive rate in most cases has increased.

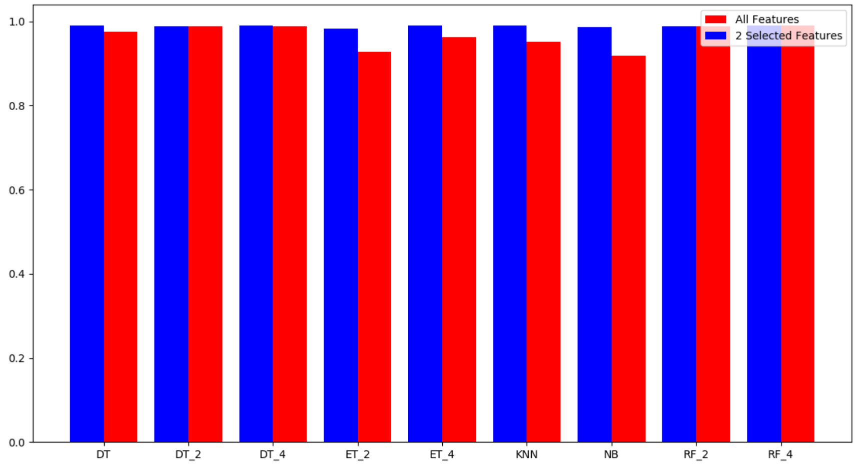

Figure 4.3: Change in Accuracy of different models (higher is better)

Table 4.3: Training and Prediction Scores

	All Features		2 Features	
Model	**TS**	**PS**	**TS**	**PS**
KNeighborsClassifier(n_neighbors = 1)	0.012	0.017	1.088	5.153
GaussianNB()	1.22	1.192	13.507	25.282
DecisionTreeClassifier()	0.259	3.248	17.359	76.115
DecisionTreeClassifier(max_depth = 2)	0.285	3.346	13.164	42.927
DecisionTreeClassifier(max_depth = 4)	0.261	3.086	5.074	76.115
RandomForestClassifier(max_depth = 2)	0.211	2.094	1.095	5.148
RandomForestClassifier(max_depth = 4)	0.138	1.9	0.963	4.518
ExtraTreesClassifier(max_depth = 2)	0.794	1.896	1.786	4.962
ExtraTreesClassifier(max_depth = 4)	0.599	1.745	1.513	4.477

Table 4.4: Confusion Matrix for the models

Model	**TN**	**FN**	**FP**	**TP**
KNN	639191	25586	8060	27163
KNN	640364	463	6887	52286
NB	641671	51709	5580	1040
NB	637461	0	9790	52749
DT	642239	12715	5012	40034
DT	640364	462	6887	52287
DT_2	638630	248	8621	52501
DT_2	638630	248	8621	52501
DT_4	638740	146	8511	52603
DT_4	640364	463	6887	52286
RF_2	639520	278	7731	52471
RF_2	639537	284	7714	52465
RF_4	642116	1837	5135	50912
RF_4	640364	463	6887	52286
ET_2	645241	48667	2010	4082
ET_2	639942	4829	7309	47920
ET_4	644140	23672	3111	29077
ET_4	640371	468	6880	52281

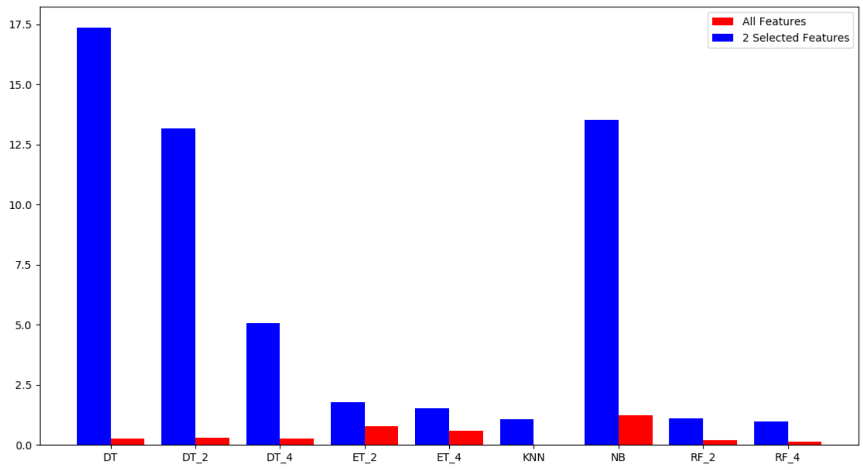

Figure 4.4: Change in TS of different models (higher is better)

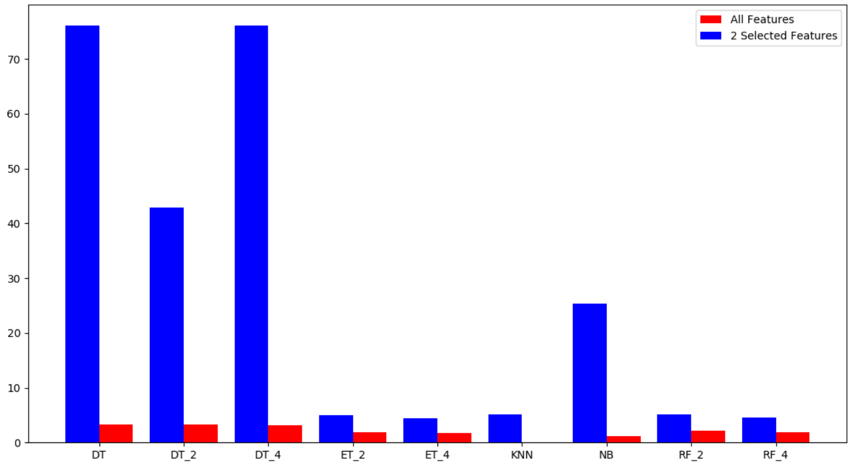

Figure 4.5: Change in PS of different models (higher is better)

Chapter 5

Conclusion

5.1 Useful features in Intrusion Detection

It is evident from this study that the basic features, such as source to destination time to live, destination to source time to live and state of the data packet are few of the most important features for intrusion detection. However, relying on them completely may lead to a system which may not respond to growing trends in cyber security. This may be because most of the attacks classified in UNSW-NB15 dataset use exploits related to these basic features. If other features are completely ignored and attackers figure out ways to exploit those features, it will become impossible to detect those intrusions in the future. It is important to update the list of important features by running periodic tests overr fresh data.

5.2 Notes about the different models

A few interesting cases came up durnig the tests.

- The GaussianNB classifier, when tested with all features, had the lowest execution time, and the lowest accuracy. It may be because GaussianNB works under the assumption that the features are not related to each other. There is minimal computation resulting in lower time signatures and since the assumption is not realistic, the accuracy might have suffered. When tested with the 2 selected features, however, while the execution time remained comparable to other models, the accuracy increased significantly. This increase in accuracy may be attributed to the low dimensionality of the dataset with just 2 features making the assumption more realistic than before.

- Decision Tree classifier with max-depth 2, gave the exact same predictions when tested with all features and when tested with 2. This may be because the optimization done with the feature selection is used automatically by the Decision Tree algorithm. The extra time used with all features must be due to the process of selecting these features. Since there are only 2 features which are selected, max-depth of 2 gives an optimal classification result. (Figure 5.1)

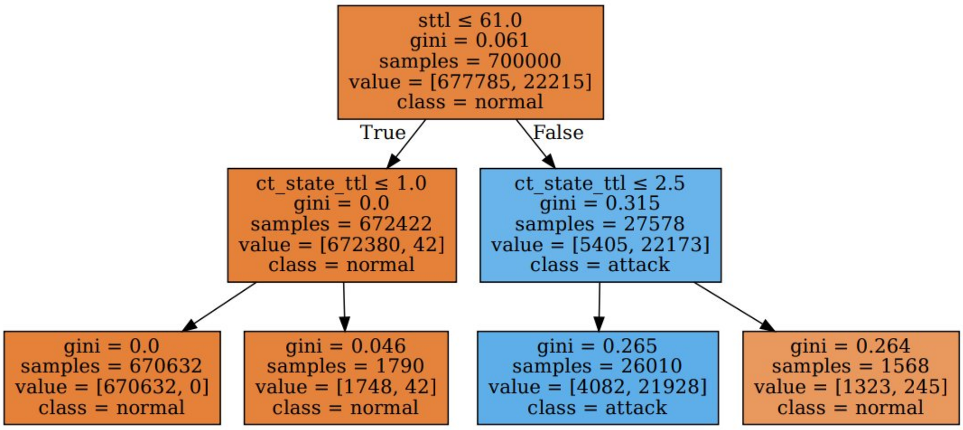

Figure 5.1: Decision Tree with max-depth = 2 tested with 2 features

5.3 Scope

Given that the UNSW-NB15 is a realistic dataset, the results from these models suggest Decision Trees as the most optimal model to implement an IDS in a low powered environment. More tests must be conucted to find an optimal value for the internal parameters of Decision Trees, such as max-depth. The radical improvement in performance from the two selected features is very convincing, however choosing the right features will be dependant on the type of data going in as input into the system and the way they are distibuted. The correct features may vary from network to network, and extreme care must be taken while implementing such a model.

Tests must be conducted on live data and small devices, which are the focus of this work. The results from this work may act as a starting point to further research in this domain.

Appendices

Appendix A

Code

A.1 Feature Selection

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from data_preprocessing_unsw import import_and_clean

ds = import_and_clean("UNSW-NB15_3.csv")

model = ExtraTreesClassifier()

model.fit(ds.iloc[:, :-2], ds.iloc[:, -1])

plt.bar(np.arange(47), model.feature_importances_)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.show()
```

A.2 Data Preprocessing

```
import pandas as pd
import numpy as np

features = pd.read_csv("../dataset/UNSW-NB15_features.csv")

def import_and_clean(dataset):
    print("importing", dataset, "...")
    data = pd.read_csv("../dataset/" + dataset, low_memory=False)
    data.columns = features.iloc[:, 1]
    print("filling blank entries...")
    data = data.fillna(0)

    print("converting nominal data to numeric...")
    cols = data.select_dtypes('object').columns
    data[cols] = data[cols].apply(lambda x: x.astype('category').cat.codes)

    return data
```

A.3 Evaluation Script

```
def evaluate(classification_result, labels):
    comparison = list(zip(classification_result, labels))

    result = {
        (0, 0): 0,
        (0, 1): 0,
        (1, 0): 0,
        (1, 1): 0
    }

    for res in comparison:
        result[res] = result[res] + 1

    print("in", classification_result.shape[0], "records, the model predicts")
    print("True Negative:\t", result[(0, 0)])
    print("False Negative:\t", result[(0, 1)])
    print("False Positive:\t", result[(1, 0)])
    print("True Positive:\t", result[(1, 1)])
    accuracy = (result[(1, 1)] + result[(0, 0)]) / (result[(0, 0)] + result[(0, 1)] + result[(1, 0)] + resu
    print("accuracy = \t", accuracy)
```

A.4 General script format for testing with all features

```
import time
import sys
import graphviz
from data_preprocessing_unsw import import_and_clean
from evaluation import evaluate
from sklearn import tree
```

```
train = import_and_clean("UNSW-NB15_1.csv")
test = import_and_clean("UNSW-NB15_2.csv")

dtc = tree.DecisionTreeClassifier(max_depth = int(sys.argv[1]))

print("training...")
start = time.time()
dtc.fit(train.iloc[:, :-2], train.iloc[:, -1])
end = time.time()

ttf = (end - start)

print("testing...")
start = time.time()
y_dtc = dtc.predict(test.iloc[:, :-2])
end = time.time()

ttp = (end - start)

print("evaluating...")
evaluate(y_dtc, test.iloc[:, -1])

print("ttf = \t\t", ttf)
print("ttp = \t\t", ttp)

dot_data = tree.export_graphviz(dtc, out_file=None, feature_names=train.columns[:-2], class_names=[’normal’
graph = graphviz.Source(dot_data)
graph.render("./results/dtc_clf_md" + str(sys.argv[1]) + "_all_features")
```

A.5 General script format for testing with the two selected features

```
import time
import sys
import graphviz
from data_preprocessing_unsw import import_and_clean
from evaluation import evaluate
from sklearn import tree

train = import_and_clean("UNSW-NB15_1.csv")
test = import_and_clean("UNSW-NB15_2.csv")
```

```
dtc = tree.DecisionTreeClassifier(max_depth = int(sys.argv[1]))


print("training...")

start = time.time()

dtc.fit(train.iloc[:, [9, 36]], train.iloc[:, -1])

end = time.time()


ttf = (end - start)


print("testing...")

start = time.time()

y_dtc = dtc.predict(test.iloc[:, [9, 36]])

end = time.time()


ttp = (end - start)


print("evaluating...")

evaluate(y_dtc, test.iloc[:, -1])


print("ttf = \t\t", ttf)

print("ttp = \t\t", ttp)


dot_data = tree.export_graphviz(dtc, out_file=None, feature_names=[’sttl’, ’ct_state_ttl’], class_names=[’n

graph = graphviz.Source(dot_data)

graph.render("./results/dtc_clf_md" + str(sys.argv[1]) + "_2_features")
```

Appendix B

Trace Files

B.1 General Trace Format

```
$ python decision_tree_all_features.py 2
    importing UNSW-NB15_1.csv ...
    filling blank entries...
    converting nominal data to numeric...
    importing UNSW-NB15_2.csv ...
    filling blank entries...
    converting nominal data to numeric...
    training...
    testing...
    evaluating...
    in 700000 records, the model predicts
    True Negative:  638630
    False Negative:  248
    False Positive:  8621
    True Positive:  52501
    accuracy =   0.98733
    ttf =   3.5196142196655273
    ttp =   0.29450440406799316
```

References

[1] H. Debar, M. Dacier, and A. Wespi, “A revised taxonomy for intrusion-detection systems,” *Annales Des Télécommunications*, vol. 55, no. 7, pp. 361–378, Jul 2000. [Online]. Available: https://doi.org/10.1007/BF02994844

[2] P. V. S. Alpao, J. R. I. Pedrasa, and R. Atienza, “Multilayer perceptron with binary weights and activations for intrusion detection of cyber-physical systems,” in *TENCON 2017 - 2017 IEEE Region 10 Conference*, Nov 2017, pp. 2825–2829.

[3] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, July 2009, pp. 1–6.

[4] N. Moustafa and J. Slay, “Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set),” in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov 2015, pp. 1–6.

[5] ——, “The significant features of the unsw-nb15 and the kdd99 data sets for network intrusion detection systems,” in *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, Nov 2015, pp. 25–31.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Pas-

sos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] H. Zhang, “The optimality of naive bayes,” 01 2004.

[8] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, Aug 1995, pp. 278–282 vol.1.

[9] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1

Table B.1: Project Detail

Student Details

Student Name	Pratyay Amrit		
Registration Number	140953430	Section/Roll No.	B/61
Email Address	p.amrit@live.com	Phone No.(M)	7795443730

Project Details

Project Title	Comparison of Intrusion Detection Systems for Use in Low-Powered Devices		
Project Duration	4 Months	Date of Reporting	08-05-2018

Internal Guide Details

Faculty Name	Ms. Ipsita Upasana
Full Contact Address with PIN Code	Department of Information and Communication Technology, Manipal Institute of Technology, Manipal-576104
Email Address	ipsita.upasana@manipal.edu