

Louisiana Tech University

**Louisiana Tech Digital Commons**

---

Doctoral Dissertations

Graduate School

---

Spring 5-2020

**Multi-Model Network Intrusion Detection System Using  
Distributed Feature Extraction and Supervised Learning**

Sangam Mulmi

Follow this and additional works at: <https://digitalcommons.latech.edu/dissertations>

---

**MULTI-MODEL NETWORK INTRUSION DETECTION SYSTEM  
USING DISTRIBUTED FEATURE EXTRACTION  
AND SUPERVISED LEARNING**

by

Sangam Mulmi, M.S., B.S.

A Dissertation Presented in Partial Fulfillment  
of the Requirements of the Degree  
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE  
LOUISIANA TECH UNIVERSITY

May 2020

LOUISIANA TECH UNIVERSITY

GRADUATE SCHOOL

April 20, 2020

Date of dissertation defense

We hereby recommend that the dissertation prepared by

**Sangam Mulmi, M.S., B.S.**

entitled **Multi-Model Network Intrusion Detection System Using Distributed  
Feature Extraction and Supervised Learning**

be accepted in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy in Computational Analysis & Modeling**



Dr. Sumeet Dua  
Supervisor of Dissertation Research

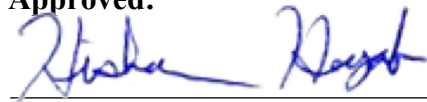


Dr. Weizhong Dai  
Head of Computational Analysis & Modeling

**Doctoral Committee Members:**

Dr. Pradeep Chowriappa  
Dr. Weizhong Dai  
Dr. Jinko Kanno  
Dr. Ramu Ramachandran

**Approved:**



Hisham Hegab  
Dean of Engineering & Science

**Approved:**



Ramu Ramachandran  
Dean of the Graduate School

## ABSTRACT

Intrusion Detection Systems (IDSs) monitor network traffic and system activities to identify any unauthorized or malicious behaviors. These systems usually leverage the principles of data science and machine learning to detect any deviations from normalcy by learning from the data associated with normal and abnormal patterns. The IDSs continue to suffer from issues like distributed high-dimensional data, inadequate robustness, slow detection, and high false-positive rates (FPRs). We investigate these challenges, determine suitable strategies, and propose relevant solutions based on the appropriate mathematical and computational concepts.

To handle high-dimensional data in a distributed network, we optimize the feature space in a distributed manner using the PCA-based feature extraction method. The experimental results display that the classifiers built upon the features so extracted perform well by giving a similar level of accuracy as given by the ones that use the centrally extracted features. This method also significantly reduces the cumulative time needed for extraction. By utilizing the extracted features, we construct a distributed probabilistic classifier based on Naïve Bayes. Each node counts the local frequencies and passes those on to the central coordinator. The central coordinator accumulates the local frequencies and computes the global frequencies, which are used by the nodes to compute the required prior probabilities to perform classifications. Each node, being evenly

trained, is capable of detecting intrusions individually to improve the overall robustness of the system.

We also propose a similarity measure-based classification (SMC) technique that works by computing the cosine similarities between the class-specific frequential weights of the values in an observed instance and the average frequency-based data centroid. An instance is classified into the class whose weights for the values in it share the highest level of similarity with the centroid. SMC contributes alongside Naïve Bayes in a multi-model classification approach, which we introduce to reduce the FPR and improve the detection accuracy. This approach utilizes the similarities associated with each class label determined by SMC and the probabilities associated with each class label determined by Naïve Bayes. The similarities and probabilities are aggregated, separately, to form new features that are used to train and validate a tertiary classifier. We demonstrate that such a multi-model approach can attain a higher level of accuracy compared with the single-model classification techniques.

The contributions made by this dissertation to enhance the scalability, robustness, and accuracy can help improve the efficacy of IDSs.

## APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation. It is understood that “proper request” consists of the agreement, on the part of the requesting party, that said reproduction is for their personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation. Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation.

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation.

Date May 8, 2020

Author 

## **DEDICATION**

This dissertation is for my beloved parents, Mrs. Sarda Mulmi and Mr. Badri Lal Mulmi, who are my greatest supporters. Without their hard work, dedication, and sacrifice, my world the way it is would not exist.

## TABLE OF CONTENTS

ABSTRACT .....	iii
APPROVAL FOR SCHOLARLY DISSEMINATION .....	v
DEDICATION .....	vi
LIST OF FIGURES .....	xiii
LIST OF TABLES .....	xv
ACKNOWLEDGMENTS .....	xvii
CHAPTER 1 INTRODUCTION.....	1
1.1    Intrusion Detection System (IDS).....	1
1.1.1    Knowledge-based IDS.....	2
1.1.2    Behavior-based IDS .....	2
1.2    Centralized and Distributed IDSs .....	3
1.3    Current Issues & Challenges .....	4
1.4    Objectives and Intended Approaches.....	5
1.5    Conclusions .....	6
CHAPTER 2 PRELIMINARIES .....	7
2.1    General Types of Data.....	7
2.2    Data Transformation .....	8
2.2.1    Normalization .....	8
2.2.2    Discretization by Binning.....	8
2.3    Feature Selection and Extraction .....	9
2.4    Distance Measures .....	11



2.5	Supervised Learning.....	11
2.6	Model Validation .....	12
2.7	Performance Evaluation .....	13
2.7.1.1	Accuracy .....	13
2.7.1.2	Precision .....	14
2.7.1.3	Recall.....	14
2.7.1.4	Specificity .....	14
2.7.1.5	F1 Score.....	14
2.8	Utilized Datasets .....	15
2.8.1	NSL-KDD.....	15
2.8.2	CICIDS2017 .....	17
2.9	Configuration and Tools.....	19
2.10	Conclusions .....	20
<b>CHAPTER 3 DISTRIBUTED FEATURE EXTRACTION FOR IDS CLASSIFIER CONSTRUCTION .....</b>		<b>21</b>
3.1	Background.....	21
3.2	Related Works.....	24
3.3	Methodology.....	25
3.3.1	Network Topology .....	25
3.3.2	Data Distribution.....	26
3.3.3	Features Analysis .....	26
3.3.4	Data Transformation.....	26
3.3.4.1	Data Normalization .....	27
3.3.4.1.1	Local Computations .....	27
3.3.4.1.2	Global Computations .....	28
3.3.4.1.3	Normalization .....	29

3.3.4.2	Class Relabeling.....	29
3.3.5	Distributed Feature Extraction.....	30
3.3.5.1	Local Eigen-Decomposition .....	30
3.3.5.2	Global Aggregation.....	30
3.3.5.3	Local Extraction.....	31
3.3.6	Classification.....	31
3.4	Experimental Procedure & Observations.....	32
3.4.1	Data Splitting, Distribution, and Normalization.....	32
3.4.2	Eigen-Decomposition.....	33
3.4.3	Local Feature Extraction .....	36
3.4.4	Classification Model-Building.....	41
3.5	Results and Discussion.....	41
3.5.1	Time Analysis of Feature Extraction.....	41
3.5.2	Classification with Original Features.....	43
3.5.3	Classification with Extracted Features.....	44
3.5.4	FPR Analysis .....	47
3.6	Conclusions .....	47
CHAPTER 4 DISTRIBUTED CONSTRUCTION OF A PREDICTION MODEL.....		50
4.1	Background.....	50
4.2	Related Works.....	52
4.3	Methodology.....	53
4.3.1	Data Preparation & Transformation.....	53
4.3.1.1	Standardization.....	54
4.3.1.2	Discretization by Binning.....	54
4.3.2	Features Analysis and Selection .....	55

4.3.3	Naïve Bayes Classifier .....	56
4.3.4	Distributed Model-Building.....	56
4.3.4.1	Data Separation by Class.....	57
4.3.4.2	Local Frequency Counting .....	57
4.3.4.3	Global Frequency Counting.....	57
4.3.4.3.1	Class Frequencies.....	58
4.3.4.3.2	Class-Specific Value Frequencies.....	58
4.3.5	Local Prior-Probabilities Computation.....	58
4.3.6	Classification.....	59
4.3.7	Validation .....	59
4.4	Experimental Procedure & Observations .....	60
4.4.1	Data Transformation & Splitting .....	60
4.4.2	Features Analysis and Selection .....	60
4.4.3	Data Separation by Class.....	62
4.4.4	Frequency Counting .....	62
4.4.5	Model Construction.....	63
4.5	Results and Discussion.....	63
4.5.1	Training and Detection Durations Analysis .....	63
4.5.2	Classification Performance.....	64
4.5.2.1	Centralized vs. Distributed Predictor Performance.....	65
4.5.3	FPR Analysis .....	66
4.6	Conclusions .....	67
CHAPTER 5 SIMILARITY MEASURE-BASED LEARNING AND MULTI-MODEL BINARY CLASSIFICATION .....		69
5.1	Background.....	69
5.2	Related Works.....	71

5.3	Methodology.....	72
5.3.1	Data Selection and Integration.....	72
5.3.2	Data Transformation.....	73
5.3.2.1	Normalization .....	74
5.3.2.2	Discretization by Binning.....	74
5.3.3	Features Analysis, Ranking, and Selection.....	74
5.3.4	Similarity Measure-based Classification (SMC) .....	74
5.3.4.1	Frequency Analysis .....	75
5.3.4.1.1	Value-Frequency in Dataset .....	76
5.3.4.1.2	Value-Frequency in Class.....	76
5.3.4.2	Frequency-based Data Centroid.....	77
5.3.4.3	Frequential-Weight Determination .....	77
5.3.4.4	Similarity Measurement .....	78
5.3.4.5	Classification.....	79
5.3.5	Distributed SMC .....	79
5.3.6	Multi-Model Binary Classification .....	80
5.3.6.1	Partially-Dependent Multi-Model (PDMM).....	82
5.3.6.2	Fully-Dependent Multi-Model (FDMM).....	83
5.4	Experimental Procedure & Observations.....	83
5.4.1	Preparation.....	83
5.4.2	Features Selection .....	83
5.4.3	SMC-based Model Construction.....	85
5.4.3.1	Class Frequencies .....	85
5.4.3.2	Frequential Value-Weight Determination .....	86
5.4.3.3	Data Centroids.....	90

5.4.4	Similarity Measurements .....	91
5.4.5	SMC-based Classification .....	92
5.4.6	Multi-Model Classification.....	94
5.5	Results and Discussion.....	94
5.5.1	Classification Performances .....	94
5.5.1.1	Single-Model Performances .....	95
5.5.1.2	Multi-Model Performances.....	96
5.5.1.3	FPR Analysis .....	97
5.6	Conclusions .....	97
CHAPTER 6 CONCLUSIONS AND FUTURE WORK.....		99
6.1	Conclusions .....	99
6.1.1	Distributed Feature Extraction.....	99
6.1.2	Distributed Classifier Construction.....	100
6.1.3	SMC and Multi-Model Approach for Binary Classification .....	101
6.1.4	Final Discussion .....	101
6.2	Future Work.....	102
BIBLIOGRAPHY .....		104

## LIST OF FIGURES

<b>Figure 1-1:</b> A basic illustration of the centralized and distributed IDS architectures. ....	4
<b>Figure 2-1:</b> An illustration of the feature selection and extraction processes with four original features. Both processes aim to reduce the number of features. ....	10
<b>Figure 2-2:</b> The iterations for k-fold cross-validation with $k = 3$ . ....	12
<b>Figure 2-3:</b> The list of descriptors available in the NSL-KDD dataset. Out of the total 43 descriptors, the 42 <sup>nd</sup> one is the class label. ....	16
<b>Figure 2-4:</b> The list of descriptors available in the CICIDS2017 dataset. There are 79 attributes in this dataset, and the last one is the class label. ....	18
<b>Figure 3-1:</b> A simulated network topology where each of the $j$ nodes is connected to the central coordinator for bi-directional information exchanges. ....	26
<b>Figure 3-2:</b> The pre-normalized standard deviation and arithmetic mean for the numerical attributes in the NSL-KDD dataset show that only a few features have an extremely high variance. ....	32
<b>Figure 3-3:</b> The pre-normalized standard deviation and average for the numerical attributes in the CICIDS2017 dataset show that several features have a high variance. ....	33
<b>Figure 3-4:</b> The first versus second PCs extracted with 1, 3, and 5 nodes in the NSL-KDD dataset. The variances explained by PC 1 and PC 2, respectively, in 1-node, 3-node, and 5-node extractions are the same. ....	37
<b>Figure 3-5:</b> The plots of the first versus second PCs extracted with 10, 25, and 50 nodes in the NSL-KDD dataset. The variances explained by PC 1 and PC 2, respectively, in these extractions are consistent. ....	38
<b>Figure 3-6:</b> The first versus second PCs extracted with 1, 3, and 5 nodes in the CICIDS2017 dataset. PC 1 and PC 2 in these extractions look somewhat correlated. ....	39
<b>Figure 3-7:</b> The first versus second PCs extracted with 10, 25, and 50 nodes in the CICIDS2017 dataset. The plots morph more rapidly with the increase in the number of nodes in this dataset. ....	40

<b>Figure 3-8:</b> The comparison between the time taken to extract the features from the a) NSL-KDD and b) CICIDS2017 datasets with a various number of nodes. 50-node parallel extraction is much faster than centralized extraction.....	42
<b>Figure 4-1:</b> The relevance of each feature determined by the chi-square test of independence in the NSL-KDD dataset using a varying number of nodes. PC 1 and PC 2 are consistently identified as the two most relevant features.....	61
<b>Figure 4-2:</b> The relevance of each feature determined by the chi-square test of independence in the CICIDS2017 dataset using a varying number of nodes. PC 4 is most-frequently identified as the most relevant feature. ....	61
<b>Figure 4-3:</b> The comparison between training and detection speeds when using various number of nodes for model construction and intrusion detection. The duration for both training and detection reduced as the number of nodes increased. ....	64
<b>Figure 4-4:</b> The performance comparison based on accuracy between the predictors constructed in centralized and distributed manners. When 3, 5, and 50 nodes were used for the CICIDS2017 dataset, the distributed classifier performed better than the centralized classifier. ....	66
<b>Figure 5-1:</b> Data integration performed to combine the previously unused categorical data and the numerical PCs extracted in CHAPTER 3.....	73
<b>Figure 5-2:</b> A high-level illustration of a multi-model classifier for binary classification. SMC, Naïve Bayes, and a tertiary classification model collaborate to make decisions. ....	81
<b>Figure 5-3:</b> The relevancy of features in the NSL-KDD-based integrated dataset. <i>service</i> is identified as the most relevant feature, followed by PC 1 and PC 2.....	84
<b>Figure 5-4:</b> The relevancy of features in the CICIDS2017-based integrated dataset. None of the original categorical features were among the 16 most relevant features. ....	84
<b>Figure 5-5:</b> The average value-frequency-based centroids in the NSL-KDD and CICIDS2017 datasets. Each point represents the average value-frequency in the respective feature.....	91
<b>Figure 5-6:</b> The plots showing the classifications in the a) NSL-KDD and b) CICIDS2017 datasets when using 12 features. The points closer to the x-axis are classified as <i>Attack</i> , and the ones closer to the y-axis are classified as <i>Normal</i> . ....	93

## LIST OF TABLES

<b>Table 2-1:</b> The general types of data with description.....	7
<b>Table 2-2:</b> The count of attributes based on their data type in the NSL-KDD dataset. ....	15
<b>Table 2-3:</b> An overview of the instances in the NSL-KDD dataset. This dataset is available in separate training and testing parts. ....	16
<b>Table 2-4:</b> The observed data types in the CICIDS2017 dataset. ....	17
<b>Table 2-5:</b> An overview of the instances in the CICIDS2017 dataset. The dataset contains eight different CSV files with the data spanning over five consecutive days.....	19
<b>Table 2-6:</b> An overview of the system configuration utilized for the experiments.....	19
<b>Table 3-1:</b> The class relabeling in the NSL-KDD and CICIDS2017 datasets. ....	29
<b>Table 3-2:</b> A comparison of the eigenvalues computed by a various number of nodes whose cumulative EV exceeds the threshold of 95% in the NSL-KDD dataset. ....	34
<b>Table 3-3:</b> A comparison of the eigenvalues computed by a various number of nodes whose cumulative EV exceeds the threshold of 95% in the CICIDS2017 dataset.....	35
<b>Table 3-4:</b> The number of new dimensions observed after feature extraction from the NSL-KDD and CICIDS2017 datasets. ....	36
<b>Table 3-5:</b> The comparison of performances between different classifiers built with the original features in the NSL-KDD dataset. k-NN performs the best with an accuracy of 97.94%. ....	43
<b>Table 3-6:</b> The comparison of performances between different classifiers built with the original features in the CICIDS2017 dataset. k-NN performs the best with an accuracy of 98.71%. ....	43
<b>Table 3-7:</b> The comparison of performances between different classifiers built with the features extracted using a various number of nodes for the NSL-KDD dataset. The overall performance of the k-NN-based classifier is the best. ....	45
<b>Table 3-8:</b> The comparison of performances between different classifiers built with the features extracted using a various number of nodes for the CICIDS2017 dataset.	



The observed performance is comparable to the classifiers that use the centrally extracted features.....	46
<b>Table 3-9:</b> The best FPR achieved by each of the tested classifiers on the respective datasets. The lowest FPRs were achieved by the k-NN. ....	47
<b>Table 4-1:</b> The performance comparisons between the classifiers constructed in a distributed way using a varying number of nodes.....	65
<b>Table 4-2:</b> The best FPR achieved in each dataset by the distributed Naïve Bayes.....	66
<b>Table 5-1:</b> The categorical attributes in the NSL-KDD and CICIDS2017 datasets utilized for the similarity measure-based classification. These attributes were ignored in the previous chapters. ....	73
<b>Table 5-2:</b> The number of rows representing each class in the training sets. ....	85
<b>Table 5-3:</b> The computed three most significant weights for the values in the top 16 features for the <i>Attack</i> class in the NSL-KDD dataset. ....	87
<b>Table 5-4:</b> The computed three most significant weights for the values in the top 16 features for the <i>Normal</i> class in the NSL-KDD dataset. ....	88
<b>Table 5-5:</b> The computed three most significant weights for the values in the top 16 features for the <i>Attack</i> class in the CICIDS2017 dataset. ....	89
<b>Table 5-6:</b> The computed three most significant weights for the values in the top 16 features for the <i>Normal</i> class in the CICIDS2017 dataset. ....	90
<b>Table 5-7:</b> A sample instance from the testing set of the NSL-KDD dataset with the observed values for features <code>service</code> , <code>PC 1</code> , <code>PC 2</code> , and <code>flag</code> .....	91
<b>Table 5-8:</b> The types of classifiers constructed to evaluate the performances of the single-model and multi-model classifiers. ....	94
<b>Table 5-9:</b> The performance comparisons of the SMC-based classifiers constructed using a varying number of features. ....	95
<b>Table 5-10:</b> The performance comparisons of the single-model Naïve Bayes and k-NN classifiers. ....	96
<b>Table 5-11:</b> The observed performances when using multi-model approaches based on SMC, Naïve Bayes, and k-NN. ....	96
<b>Table 5-12:</b> The best FPR achieved by each classifier in all tested datasets. ....	97

## ACKNOWLEDGMENTS

I am grateful to many who have offered their valuable inputs throughout this dissertation research. Foremost, I would like to thank my research advisor, Dr. Sumeet Dua, for guiding me to pursue the correct path that appropriately led to the finish line. Additionally, my sincere appreciation goes to the honorable advisory committee members — Dr. Pradeep Chowriappa, Dr. Weizhong Dai, Dr. Jinko Kanno, and Dr. Ramu Ramachandran — for providing honest critique and keeping me on track. My thanks also extend to every educator who has ever shared their knowledge and wisdom, so I could continually learn and flourish.

My family and friends have always been there. Their everlasting support has made it possible to reach this point. I thank them from the bottom of my heart. Especially, thanks to my dear brother, Mr. Bipan Mulmi, for a constant push; thanks to my beautiful fiancé, Ms. Taylor Poland, for all the motivation and keeping me focused; and finally, thanks to my respected uncle, Mr. Yogesh Shrestha, for inspiring me to explore the wonders of science and mathematics.

# CHAPTER 1

## INTRODUCTION

### 1.1 Intrusion Detection System (IDS)

Network intrusions are unauthorized activities in a computing network that compromise its security, resources, and data. All networking infrastructures, including the internet and intranet, are prone to intrusions. It is vital to detect intrusions promptly to mitigate the risks posed by them. An intrusion detection system (IDS) aides in identifying intrusions. An IDS typically is software that examines and analyzes network data packets to identify anything suspicious [1]. Such a system learns from the usual pattern of the network and flags the activities that do not appear reasonable. To build a classification model that is capable of adequately distinguishing between the normal and abnormal network traffic, the IDS must learn from the known instances of the network behavior. Such learning heavily relies on data analysis and machine learning techniques [2]. Traditionally, IDSs are implemented centrally. This type of IDS architecture that utilizes only a single central node requires all the data essential for detection to be passed through it for screening. Depending on the way an IDS is constructed and implemented, most IDSs can be categorized into one of the following two main categories — Knowledge-based IDS and Behavior-based IDS.

### 1.1.1 Knowledge-based IDS

A knowledge-based IDS utilizes previously known attacks and system vulnerabilities to build the rules or signatures. The signatures are the known patterns that define an attack, which can be represented as a set of rules [3]. The new questionable data is compared against the previously formulated signatures. If any match is found, meaning there exists a signature that matches the properties of the current data in question, then it is identified as an attack. The knowledge-based IDSs tend to be fast and accurate as they work by performing comparisons between their observations and the predetermined set of rules [4]. They, however, fail to detect any new attacks because even a minor deviation from the original attack causes the new attack to mismatch with all the previously created signatures. Due to this, such IDSs are unable to detect zero-day attacks. The zero-day attacks are the attacks that are being observed for the very first time; therefore, their signature is not present in the system yet [5]. The signature dictionary, consequently, requires frequent updating to ensure the signatures of the latest threats are available. With the rapid evolvement of new types of attacks, basing a network's security solely on a knowledge-based IDS is not preferable.

### 1.1.2 Behavior-based IDS

The behavior-based IDS relies on a proper understanding of the network traffic patterns. To build this type of IDS, the network traffic-related data is statistically analyzed, and a prediction model that differentiates between the normal and abnormal traffic is developed. The prediction model is often based on a clustering or classification technique. Usually, a set of data containing both normal and abnormal traffic patterns are used to build a prediction model. It is also possible to train the system with just the

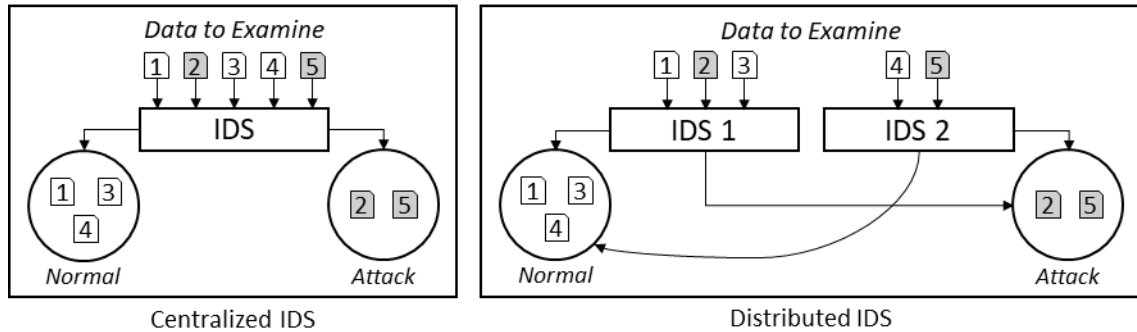
normal traffic data such that whenever a new type of traffic that does not adhere to the regular traffic pattern is detected, then it is flagged as an attack. When an IDS is implemented to identify abnormal traffic, it is categorized as an anomaly-based IDS. The main benefit of the behavior-based IDS is its ability to detect new types of attacks. The issue, however, is that the behavior-based IDSs tend to suffer from slower detection and higher false alarms.

## **1.2 Centralized and Distributed IDSs**

In a centrally implemented IDS, the central node that is responsible for running an IDS undertakes all the training, testing, and detecting tasks; therefore, all the data are passed through it. Since this central node has access to the entirety of the data, it can build a detection model that is representative of all the previously observed instances. With the growth in the implementation of a distributed computing environment for the modern network infrastructures, the traditional centrally located IDS is gradually becoming obsolete. The present-day's massive volume of network data transfers can become overwhelming to the single central IDS. Due to these, the interest has grown to design, develop, and implement a distributed IDS architecture.

One of the first proposed distributed IDS performs traffic monitoring in a distributed manner but performs data analysis centrally [6]. Many common forms of distributed IDS employ a central node that helps aggregate the data from each node. In such architecture, each node can detect attacks based on the patterns that are learned collaboratively with the help of the central coordinator. Because the detection happens locally on distributed nodes, the incoming data can be distributed among them for inspection, which reduces the load on a single system. This type of IDS is more robust

and requires limited data throughput in nodes [7]. **Figure 1-1** illustrates high-level architectures of centralized and distributed IDSs.



**Figure 1-1:** A basic illustration of the centralized and distributed IDS architectures.

### 1.3 Current Issues & Challenges

The process of construction and utilization of the IDS has crossed many milestones since its inception. The modern-day IDSs have evolved into sophisticated systems powered by the advanced artificial intelligence capabilities; however, they continue to suffer from some of the common issues like high-dimensional data, slow detection speed, poor robustness, and high false alarms [8]. False alarms are high when the normal traffic is incorrectly detected as an attack. Because of such inaccuracy, many healthy connections could get affected. If all the traffic flagged as an attack were to be reviewed manually for verification, then falsely flagging many could overload the queue containing the suspicious traffic data to be reviewed.

In a centralized IDS, all the necessary data passes through the central node that is responsible for monitoring the network traffic, triggering the need for high processing power and bandwidth connection on that node. Additionally, the privacy of the data owned by each node in a network is diminished because all the raw data destined to or

originating from them are visible to the central node. Having a single central IDS also makes the entire network vulnerable to a single point of failure [9]. In an event when the primary system responsible for operating the IDS goes down, the attacks in any part of the network may go unnoticed.

This dissertation aims to dissect these issues, investigate potential solutions, and propose appropriate approaches to help overcome them.

#### **1.4 Objectives and Intended Approaches**

The IDSs continue to encounter several challenges. The general objective of this dissertation is to explore some of those challenges and present potential remedies.

The feature extraction can be done in a distributed manner to handle high-dimensional distributed data. In such an approach, each node sends some information about the data to the central coordinator for aggregation. The nodes use the aggregated data for feature extraction. To be considered useful, the features extracted distributedly must perform as effectively as the features extracted centrally. Their effectiveness can be verified by separately building the classifiers with both centrally and distributively extracted sets of features and comparing their performances in terms of accuracy and other measures.

The IDS classifier construction and implementation can also be done in a distributed fashion to improve the robustness and detection speed. Numerous nodes in a network can collaboratively construct a classifier, which can be used by each node individually to detect intrusions. Such IDS architecture would be robust, mitigating the risks posed by a single point of failure. Since the workload is distributed across multiple

nodes, the distributed system would be able to process more data in a shorter amount of time. As a result, the distributed IDS would be able to detect attacks more rapidly.

Finally, a multi-model architecture with ensembled classifiers can be utilized to improve the performance of an IDS in terms of detection accuracy. Such an improvement could also reduce the false-positive rates (FPRs). The information produced by multiple lightweight prediction models can be passed into another classifier as input features to identify whether an instance being investigated is indeed an attack. The traffic that is flagged as an attack is usually examined manually by the network security experts to confirm its maliciousness. Improving the classification accuracy and consequently reducing the number of falsely flagged traffic by using a multi-model approach can help limit the amount of manual monitoring and analysis needed to keep the network systems secure.

CHAPTER 3, CHAPTER 4, and CHAPTER 5, sequentially, discuss the intended approaches in detail while outlining significant findings and observations.

## **1.5 Conclusions**

This chapter commenced with an overview of the IDSs. It introduced their types in terms of the way they are constructed and implemented. It also gave an overview of how the machine learning and data science powers the modern IDSs. Additionally, it presented some common issues that IDSs continue to encounter; then, it laid out the objectives and some intended approaches to address those issues.



## CHAPTER 2

### PRELIMINARIES

#### 2.1 General Types of Data

Each column in a structured dataset represents some specific descriptor. Different columns may have different types of values stored in them; however, a specific column only holds a specific type of data. Depending on the type of values stored in a column, the type of data could be quantitative or qualitative. The two main categories of data types are numerical and categorical. The numerical data are represented by some numbers. They can be differentiated as a discrete, continuous, interval, or ratio type. The categorical data are generally represented by some texts and are usually categorized as nominal or ordinal [10]. The ordinal data type has a specific order but lacks the extent of the difference between the values. **Table 2-1** shows the different types of data.

**Table 2-1:** The general types of data with description.

Type	Subtype	Description	Example
Numerical	Discrete	Whole-number values.	The number of nodes.
	Continuous	Any value between whole numbers.	Size of data packets.
	Interval	Measured along a scale; no true zero.	The temperature.
	Ratio	Like interval, but with true zero.	Distance between nodes.
Categorical	Nominal	Categories with no specific order.	Hostname, port number.
	Ordinal	Categories with a specific order.	The level of risk.

## 2.2 Data Transformation

Data transformation is a data preprocessing procedure that is often necessary to change the data in one form to another to make them more appropriate to construct and implement the predictive models. Depending on the situation, different transformations could be necessary. The following are some common types of transformations.

### 2.2.1 Normalization

A significant difference in values between the features in a dataset is common. Using such a dataset to construct a predictive model can be problematic because the larger values may have a stronger influence. The values need to be scaled such that they become suitable [11]. The z-score normalization technique is one of the standardization techniques that help normalize the data and put the values into the same scale. This technique uses the mean and standard deviation in such a way that the arithmetic mean of the resulting normalized values becomes 0, and their standard deviation becomes 1. The z-score normalization is given by

$$z = \frac{x - \mu}{\sigma}, \quad \text{Eq. 2-1}$$

where  $x$  is the currently observed value,  $\mu$  is the population mean, and  $\sigma$  is the population standard deviation. In an event when  $\mu$  and  $\sigma$  are unavailable, the sample mean,  $\bar{x}$ , and sample standard deviation,  $s$ , can be used.

### 2.2.2 Discretization by Binning

Binning is a form of mapping that puts the numeric values into bins or buckets for discretization, such that the continuous values are grouped into some discrete bins. Converting continuous values into categorical values makes them compatible with the algorithms that only handle categorical values. The common types of binning include

equal-width binning and equal-frequency binning [12]. The equal-width binning method determines the width of bins using

$$width = \frac{v_{max} - v_{min}}{number\ of\ bins}, \quad \text{Eq. 2-2}$$

where  $v_{max}$  and  $v_{min}$  are the maximum and minimum values to be binned. The number of bins is pre-defined. The computed width is used to generate the ranges for bins. All the values falling into a specific range are put into the same bin. In equal-frequency binning, a set number of values are put into the same bin while ensuring that each bin has an equal number of values. For this dissertation, the equal-width binning technique is used to discretize continuous values, whenever necessary.

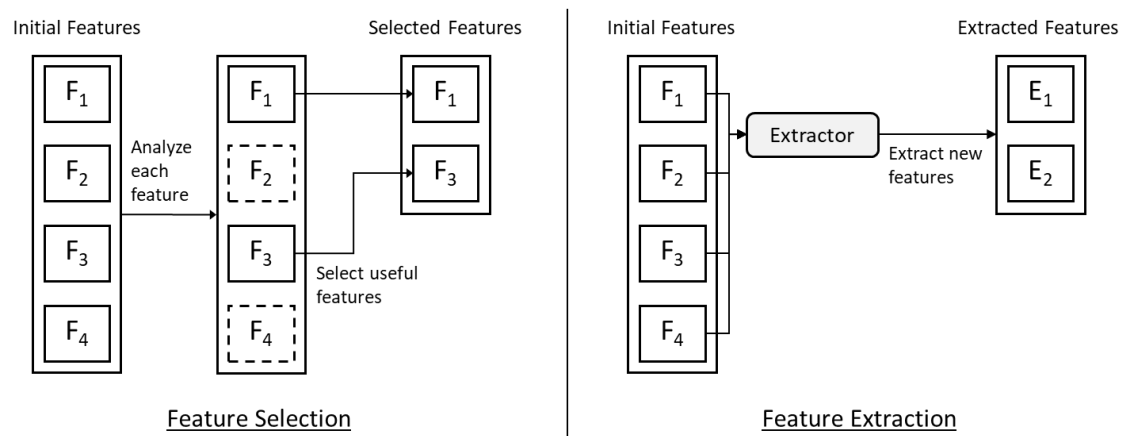
### 2.3 Feature Selection and Extraction

Often confused as the same process, the feature selection and extraction are two different processes. When the feature space is reduced through a proper feature selection or extraction, the classification performance can improve [13].

The feature selection deals with the identification and removal of the unnecessary, irrelevant, and duplicate attributes. It can be done in a supervised or unsupervised manner. The feature selection aims to reduce the number of features while improving the classification accuracy [14]. One typical example of the feature selection technique is the chi-square test of independence, which examines each attribute's degree of independence from the target variable. This method is useful for categorical data. Another technique, Analysis for Variance (ANOVA), computes the amount of variance within and between the samples by analyzing their means [15]. This technique is suitable when the input variables are numeric, and the target variable is categorical. Similarly, the method based

on information gain identifies suitable features based on the mutual information between two variables [16].

In contrast, the feature extraction techniques analyze the available descriptors and use them to generate new features while ensuring that the desired amount of information is preserved. Feature extraction results in dimensionality reduction, making it easier to tackle the curse of dimensionality. In a dataset containing a target variable, feature extraction can be done without its consideration. Principal Component Analysis (PCA) is one of the conventional and widely-used unsupervised feature extraction techniques that projects the data in a higher dimension into the lower dimension while ensuring each feature is orthogonal to one another [17]. Such projection ensures independence between attributes while reducing the number of dimensions. **Figure 2-1** demonstrates the feature selection and extraction processes.



**Figure 2-1:** An illustration of the feature selection and extraction processes with four original features. Both processes aim to reduce the number of features.

## 2.4 Distance Measures

Distance measures compute how far two points are from one another. They can be used to determine the degree of dissimilarity or similarity between the data points. The data points are similar if the distance between them is short. There are different types of distance measures. Some of the notable ones in data science are Euclidean Distance, Manhattan Distance, and Minkowski Distance [18]. Most distance measures only deal with numerical values. When the data is of nominal or ordinal type, then the available options for distance measures are limited. Jaccard similarity coefficient and cosine similarity are two commonly used techniques to measure the degree of similarity between the two data points represented by categorical values.

## 2.5 Supervised Learning

Supervised learning is a branch of machine learning where the dataset has a labeled target variable containing class labels. In this type of learning, the model is built by tuning it to predict the class labels accurately. The goal is to learn a mapping function such that for a given set of inputs, the predictor determines an accurate output. As shown in **Eq. 2-3**, the input values in  $X$  are mapped into an output  $Y$ .

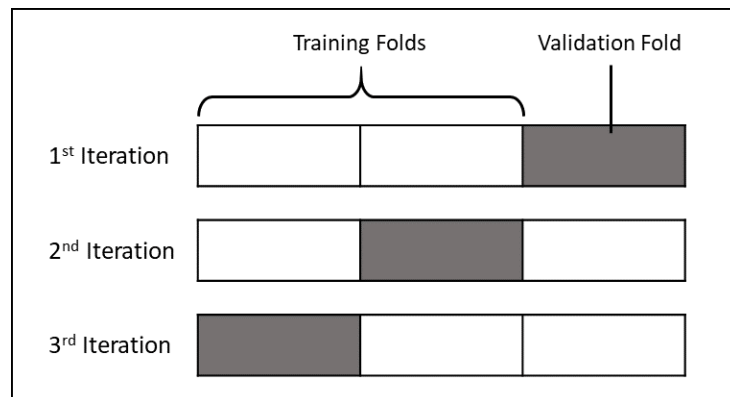
$$f(X) = Y \qquad \text{Eq. 2-3}$$

The supervised learning techniques pass through the training and testing phases. In the training phase, the class labels are available during the learning process. The adjustments are made as necessary to ensure that the built model is a good predictor of the class. The testing phase uses the constructed model to classify the test instances. Since the actual class labels are known, the performance of the model can be evaluated by comparing the observed outcomes against the expected outcomes. Some notable

supervised learning methods include Naïve Bayes, Decision Tree, k-Nearest Neighbor (k-NN), and Neural Network.

## 2.6 Model Validation

In machine learning, the model validation is referred to as the process where the trained model is evaluated using the test data. The cross-validation is one of the model validation techniques that examines how the results obtained by a predictive model generalizes to the new independent dataset [19]. Cross-validation evaluates the predictive model's performance on limited data through random resampling. Its purpose is to perform some statistical analysis of a model to determine its actual effectiveness in terms of accuracy and other quality measures when applied to the previously unseen data. The holdout method is one of the variants of cross-validation technique where the data is split in some ratio for training and testing purposes. The more substantial portion is used for training, and the smaller portion is used for testing. In k-fold cross-validation, the dataset is split into  $k$  equal subsets known as folds. The  $k - 1$  folds are used for training, and the remaining held-out fold is used for testing, as shown in **Figure 2-2**.



**Figure 2-2:** The iterations for k-fold cross-validation with  $k = 3$ .

This process is repeated  $k$  times, ensuring that each fold is used as the validation fold once. The performance outcomes obtained from each of the iterations are averaged to get the model's overall performance result, which is given by

$$Performance = \frac{1}{k} \sum_{i=1}^k Performance_i. \quad \text{Eq. 2-4}$$

## 2.7 Performance Evaluation

Several metrics are available to evaluate a predictor's performance. The performance result ideally consists of the counts of true-positives (TP), false-positives (FP), true-negatives (TN), and false-negatives (FN). These are represented in a confusion matrix form with expected and observed outputs for binary classifications. The performance of a model can be evaluated by analyzing measures like accuracy, precision, recall, and specificity. The precision and recall can be combined into a single performance metric called F1 score. These metrics can be multiplied by 100 for scaling.

### 2.7.1.1 Accuracy

The accuracy of a model depends on how many instances are correctly predicted when compared to the total number of predictions made. It is computed as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \quad \text{Eq. 2-5}$$

Using accuracy as a performance evaluator is appropriate only when all the classes in a data sample are evenly represented. If they are not, then classifying all the instances into the most representative class would still give a good accuracy result, causing a false sense of high accuracy.

### 2.7.1.2 Precision

Precision is an indicator of the model's ability to identify the positive instances correctly, which is given by

$$Precision = \frac{TP}{TP + FP}. \quad \text{Eq. 2-6}$$

An IDS with a low precision would imply that a significant number of regular traffic is being classified as an attack. It is essential to reduce such misclassifications, to avoid unnecessary flagging of the regular traffic.

### 2.7.1.3 Recall

Recall, also known as sensitivity, is a measure of what proportion of the instances that are positive are classified as positive. It is given by

$$Recall \text{ or } Sensitivity = \frac{TP}{TP + FN}. \quad \text{Eq. 2-7}$$

Since both TP and FN are actual positive instances, recall helps determine the model's ability to identify the true-positive instances as positives.

### 2.7.1.4 Specificity

Specificity computes the proportion of actual negative instances classified as negatives. It is the opposite of recall and is given by

$$Specificity = \frac{TN}{TN + FP}. \quad \text{Eq. 2-8}$$

Specificity and FPR are related, such that  $FPR = 1 - Specificity$ .

### 2.7.1.5 F1 Score

F1 score computes the model's accuracy based on its precision and recall as

$$F1 \text{ score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \quad \text{Eq. 2-9}$$



This measure helps determine the balance between precision and recall. Since it is not affected by the imbalanced class distribution, it is a better measure of accuracy when the data sample represents one class significantly more than the others.

## 2.8 Utilized Datasets

The NSL-KDD and CICIDS2017 datasets have been used for the experiments throughout this dissertation. These are popular IDS-related datasets that come in a structured form. They are openly available and are widely used in research [20] [21]. The compressed file size of the NSL-KDD dataset is 6.45 megabytes, and that of the CICIDS2017 dataset is 229.49 megabytes.

### 2.8.1 NSL-KDD

The NSL-KDD dataset is derived from the KDD Cup 1999 dataset to address some of its inherent issues [20]. This dataset comes in the form of text files containing comma-separated values (CSVs). A detailed analysis of this dataset is conducted in [22]. It is available in different parts containing training and testing sets. **Table 2-2** shows the data types of the attributes in this dataset.

**Table 2-2:** The count of attributes based on their data type in the NSL-KDD dataset.

Data Type	Subtype	Number of Attributes
Numerical	Discrete (Integer)	17
	Continuous	15
Categorical	Nominal	3
	Nominal (Binary)	6
<b>Total Attributes</b>		<b>41</b>

The NSL-KDD dataset has a total of 43 columns. The values in the 42<sup>nd</sup> column are the labels indicating whether the specific row represents a normal instance or some specific kind of attack. Other columns, excluding the 43<sup>rd</sup> column, contain the connection-related, content-related, time-related, and host-related traffic data [22]. The 43<sup>rd</sup> column indicates the level of classification difficulty for a particular instance. All the descriptors available in the NSL-KDD dataset are listed in **Figure 2-3**.

NSL-KDD Dataset Columns			
1. duration	12. logged_in	23. count	34. dst_host_same_srv_rate
2. protocol	13. num_compromised	24. srv_count	35. dst_host_diff_srv_rate
3. service	14. root_shell	25. serror_rate	36. dst_host_same_src_port_rate
4. flag	15. su_attempted	26. srv_serror_rate	37. dst_host_srv_diff_host_rate
5. src_bytes	16. num_root	27. rerror_rate	38. dst_host_serror_rate
6. dst_bytes	17. num_file_creations	28. srv_rerror_rate	39. dst_host_srv_serror_rate
7. land	18. num_shells	29. same_srv_rate	40. dst_host_rerror_rate
8. wrong_fragment	19. num_access_files	30. diff_srv_rate	41. dst_host_srv_rerror_rate
9. urgent	20. num_outbound_cmds	31. srv_diff_host_rate	<b>42. label</b>
10. hot	21. is_hot_login	32. dst_host_count	43. difficulty
11. num_failed_logins	22. is_guest_login	33. dst_host_srv_count	

**Figure 2-3:** The list of descriptors available in the NSL-KDD dataset. Out of the total 43 descriptors, the 42<sup>nd</sup> one is the class label.

The instances of both classes are quite evenly distributed in this dataset, with 48.12% of them being attacks and 51.88% of them being normal, as seen in **Table 2-3**. Such balanced datasets are considered appropriate for building classification models because each class is evenly represented, reducing the chance of bias.

**Table 2-3:** An overview of the instances in the NSL-KDD dataset. This dataset is available in separate training and testing parts.

Filename	Attack	Normal	Total Rows
KDDTrain+.txt	58,630	67,343	125,973
KDDTest+.txt	12,833	9,711	22,544
<b>Total Instances</b>	<b>71,463</b>	<b>77,054</b>	<b>148,517</b>

For this dissertation, the available training and testing files are merged into a single file. The merged file is used for both training and testing by leveraging the random sampling and cross-validation techniques.

### 2.8.2 CICIDS2017

The CICIDS2017 dataset contains the traffic data collected for five days. This dataset is considered to have network traffic data resembling the real-world attacks [23]. Even though the actual data packets obtained by capturing network packets are available, the information from those packets has been extracted into eight CSV files. Each of those files pertains to a specific day and the types of attacks undertaken that day. The rows in these files represent the information extracted or computed from the captured packets. The data types and the number of attributes using them in this dataset are shown in **Table 2-4**.

**Table 2-4:** The observed data types in the CICIDS2017 dataset.

Data Type	Subtype	Number of Attributes
Numerical	Discrete (Integer)	40
	Continuous	23
Categorical	Nominal	1
	Nominal (Binary)	8
	Nominal (Unary)	6
<b>Total Attributes</b>		<b>78</b>

There are 79 columns in this dataset. The last column is the class label that specifies whether an instance belongs to the normal traffic or some type of attack.

**Figure 2-4** lists all the available descriptors in this dataset.

CICIDS2017 Dataset Columns			
1. Destination Port	21. Fwd IAT Total	41. Packet Length Mean	61. Bwd Avg Packets/Bulk
2. Flow Duration	22. Fwd IAT Mean	42. Packet Length Std	62. Bwd Avg Bulk Rate
3. Total Fwd Packets	23. Fwd IAT Std	43. Packet Length Variance	63. Subflow Fwd Packets
4. Total Backward Packets	24. Fwd IAT Max	44. FIN Flag Count	64. Subflow Fwd Bytes
5. Total Length of Fwd Packets	25. Fwd IAT Min	45. SYN Flag Count	65. Subflow Bwd Packets
6. Total Length of Bwd Packets	26. Bwd IAT Total	46. RST Flag Count	66. Subflow Bwd Bytes
7. Fwd Packet Length Max	27. Bwd IAT Mean	47. PSH Flag Count	67. Init_Win_bytes_forward
8. Fwd Packet Length Min	28. Bwd IAT Std	48. ACK Flag Count	68. Init_Win_bytes_backward
9. Fwd Packet Length Mean	29. Bwd IAT Max	49. URG Flag Count	69. act_data_pkt_fwd
10. Fwd Packet Length Std	30. Bwd IAT Min	50. CWE Flag Count	70. min_seg_size_forward
11. Bwd Packet Length Max	31. Fwd PSH Flags	51. ECE Flag Count	71. Active Mean
12. Bwd Packet Length Min	32. Bwd PSH Flags	52. Down/Up Ratio	72. Active Std
13. Bwd Packet Length Mean	33. Fwd URG Flags	53. Average Packet Size	73. Active Max
14. Bwd Packet Length Std	34. Bwd URG Flags	54. Avg Fwd Segment Size	74. Active Min
15. Flow Bytes/s	35. Fwd Header Length	55. Avg Bwd Segment Size	75. Idle Mean
16. Flow Packets/s	36. Bwd Header Length	56. Fwd Header Length	76. Idle Std
17. Flow IAT Mean	37. Fwd Packets/s	57. Fwd Avg Bytes/Bulk	77. Idle Max
18. Flow IAT Std	38. Bwd Packets/s	58. Fwd Avg Packets/Bulk	78. Idle Min
19. Flow IAT Max	39. Min Packet Length	59. Fwd Avg Bulk Rate	79. Label
20. Flow IAT Min	40. Max Packet Length	60. Bwd Avg Bytes/Bulk	

**Figure 2-4:** The list of descriptors available in the CICIDS2017 dataset. There are 79 attributes in this dataset, and the last one is the class label.

An analysis of the CICIDS2017 dataset reveals that it contains significantly more normal instances than attack instances. Such a difference can cause bias in the learning process by heavily favoring the normal instances. Only 19.7% of the instances represent the attacks. It has been noted that this resembles the practical network traffic, where the number of attacks is usually significantly lower than the regular traffic exchanges.

For this research, all eight available files in the CICIDS2017 dataset are merged into a single file, and a specific number of rows are selected through random sampling as needed. When sampling, the even class balance is enforced such that the equal number of normal-related and attack-related samples are selected. Specifically, a dataset containing a million rows with fifty-percent instances representing attacks and another fifty-percent representing regular traffics is formed by randomly selecting the samples from the entire dataset. The eight files available in this dataset and the number of instances of each class in each file are shown in **Table 2-5**.

**Table 2-5:** An overview of the instances in the CICIDS2017 dataset. The dataset contains eight different CSV files with the data spanning over five consecutive days.

Filename	Attack	Normal	Total Rows
Monday-WorkingHours.pcap_ISCX.csv	0	529,918	529,918
Tuesday-WorkingHours.pcap_ISCX.csv	13,835	432,074	445,909
Wednesday-workingHours.pcap_ISCX.csv	252,672	440,031	692,703
Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv	2,180	168,186	170,366
Thursday-WorkingHours-Afternoon-Infiltration.pcap_ISCX.csv	36	288,566	288,602
Friday-WorkingHours-Morning.pcap_ISCX.csv	1,966	189,067	191,033
Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv	128,027	97,718	225,745
Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv	158,930	127,537	286,467
<b>Total Instances</b>	<b>557,646</b>	<b>2,273,097</b>	<b>2,830,743</b>

## 2.9 Configuration and Tools

The experiments are undertaken on a single host computer. For experiments requiring a distributed network environment, a multi-node environment is simulated in a single central computer. **Table 2-6** shows the utilized system's hardware configuration.

**Table 2-6:** An overview of the system configuration utilized for the experiments.

<b>Processor</b>	Intel® Xeon(R) CPU E5-1620 v3 @ 3.50GHz
<b>Memory</b>	32 GB 2133 MHz RIMM DDR4
<b>Storage</b>	HP EX950 M.2 1TB PCIe 3.1 x4 NVMe 3D TLC NAND SSD

To prepare, setup, and run the experiments, various tools, including RapidMiner Studio version 9.5, Microsoft Excel, and some Python libraries, along with self-written codes, are utilized. The outputs of the self-written programs have been modularly validated against the outputs produced by other reputable tools to check their reliability before using them.

## **2.10 Conclusions**

This chapter explained some preliminary concepts needed to understand the presented ideas. It also introduced the datasets, NSL-KDD and CICIDS2017, which are used to verify the applicability of the proposed techniques in the intrusion detection domain. These are popular and openly available datasets that have been widely used for IDS-related research. The tools and methods that have been utilized were also described. In general, some relevant technical insights were provided.

## **CHAPTER 3**

### **DISTRIBUTED FEATURE EXTRACTION FOR IDS CLASSIFIER CONSTRUCTION**

#### **3.1 Background**

When building a prediction model, the quality of the features used dictates its performance. Many practical datasets tend to have numerous features. Several of these features can carry redundant or useless information for prediction. The models built using such features can cause overfitting or underfitting. Additionally, when many features are used to build a model, the complexity of the problem becomes high, causing the need for expensive computational resources. Each attribute in a dataset is considered its dimension; hence, the number of attributes is equal to the number of dimensions. A higher dimension causes a more complex problem, resulting in the curse of dimensionality. The feature extraction process deals with taking the existing data descriptors and extracting new features from them while ensuring that the newly extracted features retain the maximum information from the data [24]. The features holding the least variance can be excluded from the model building process. Such exclusion results in dimensionality reduction.

There are numerous feature extraction techniques. PCA is one of the prevalent dimensionality reduction and feature extraction techniques. It describes data variation as a set of uncorrelated and independent variables known as principal components (PCs).

The PCs are generated by projecting high dimensional data into a low dimensional feature space while preserving its intrinsic characteristics [25] [26]. Such extracted PCs are orthogonal from one another. The PCA is undertaken without any regard to the class labels, so it is an unsupervised feature extraction process.

The PCA is conducted by first forming a covariance matrix of the given data, then performing eigen-decomposition to compute the eigenpair, which contains the eigenvalues and their respective eigenvectors. The eigenvalues and eigenvectors summarize the data. The first few largest eigenvalues with high explained variance are selected, and their corresponding eigenvectors are used to determine the new features.

For a sample dataset  $\mathbf{X} = \{R_1, R_2, \dots, R_N\} \in \mathbb{R}^{n \times N}$  with  $n$  dimensions and  $N$  number of rows, the covariance,  $\sigma(x, y)$ , between two random variables,  $x$  and  $y$ , is

$$\sigma(x, y) = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y}), \quad \text{Eq. 3-1}$$

where  $x_i$  and  $y_i$  are the observed values for  $x$  and  $y$  attributes, and  $\bar{x}$  and  $\bar{y}$  are the means of all the values in those attributes, respectively. Based on the computed covariances between all pairs of attributes in  $\mathbf{X}$ , the covariance matrix,  $\mathbf{K} \in \mathbb{R}^{n \times n}$ , can be determined. A covariance matrix is symmetric and positive definite [27], and it can be decomposed into three matrices such that it becomes equivalent to their products. In **Eq. 3-2**,  $\mathbf{V}$  is the matrix with eigenvectors,  $\mathbf{\Lambda}$  is a diagonal matrix containing the corresponding eigenvalues on its diagonal elements in decreasing order, and  $\mathbf{V}^T$  is a transposed  $\mathbf{V}$ .

$$\mathbf{K} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \quad \text{Eq. 3-2}$$



Let  $\lambda$  and  $v$  represent eigenvalue and eigenvector, respectively. If  $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$  is a set of eigenvalues such that  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$ , then the eigenpairs containing eigenvalues with their corresponding set of eigenvectors, can be represented as

$$Eig_K = \{(\lambda_1, \{v_{\lambda_1,1}, v_{\lambda_1,2}, \dots\}), \dots, (\lambda_n, \{v_{\lambda_n,1}, v_{\lambda_n,2}, \dots\})\}. \quad \text{Eq. 3-3}$$

Depending on the predefined explained variance threshold, the first few eigenvalues are selected. Then, a projection matrix,  $\mathbf{P}$ , is constructed using the eigenvectors derived by the selected eigenvalues. This projection matrix is used to project the original data into a lower dimension linearly, as given by

$$\mathbf{Y} = \mathbf{P}^T \cdot \mathbf{X}, \quad \text{Eq. 3-4}$$

where  $\mathbf{P}^T$  is the transpose of matrix  $\mathbf{P}$ . The resulting matrix,  $\mathbf{Y}$ , contains the PCs, with the most important component being the first one [28].

The PCA in a centralized environment is a well-studied area; however, there has been a limited study on its applicability in a distributed IDS to extract features for a prediction model. With the rise in the implementation of a distributed computing architectures, it would be nonsensical to continue using a strictly-central IDS requiring extravagant computational, storage, and bandwidth resources on a single IDS host. Besides that, since all the information stored in the descriptors would have to pass through the central processor to build a predictor using a central IDS, it would not be suitable for privacy-conscious nodes that do not desire to share their raw data with others. This chapter discusses a distributed feature extraction technique for IDS, where multiple nodes collaboratively extract the features that is representative of the global dataset with only their portion of the data. The distributed nodes achieve this with some assistance from a central coordinator.

### 3.2 Related Works

There have been studies on the type of distributed PCA where each node has access to only a subset of data. According to the review in [29], this type of PCA usually has local and global stages. At the local stage, each node with access to only a subset of data performs local PCA and forwards some information about the result to the central coordinator. At the global stage, the central coordinator performs a global PCA by aggregating the information received from each node.

In [30], the authors propose and analyze a distributed PCA algorithm where each node computes the top  $K$  eigenvectors of the covariance matrix for its portion of data. These top  $K$  eigenvectors are sent to the central node. The central node aggregates the information collected from the nodes and performs PCA based on the aggregated information. Through their experiments and analysis, the authors successfully show that, with enough intermediate nodes, the distributed PCA, despite having access to only limited data, performs as well as the centralized PCA. The authors have validated the results they presented by running experiments in a simulated environment.

Similarly, [31] proposes a Minimum Volume Elliptical PCA algorithm that is claimed to be robust due to its ability to identify PCs of the data, even when there are anomalies present in a training dataset. Such ability prevents any skewing of PCs caused by anomalous data [31]. The authors demonstrate that their proposed algorithm performs better in a centralized environment. They, however, reformulate the technique using a distributed convex optimization problem, where the problem is split across many nodes. Each node, then, computes based only on its portion of data and exchanges the resulting

small matrices with its neighboring nodes. This approach caused the performance of the distributed method to be comparable with that of the centralized method.

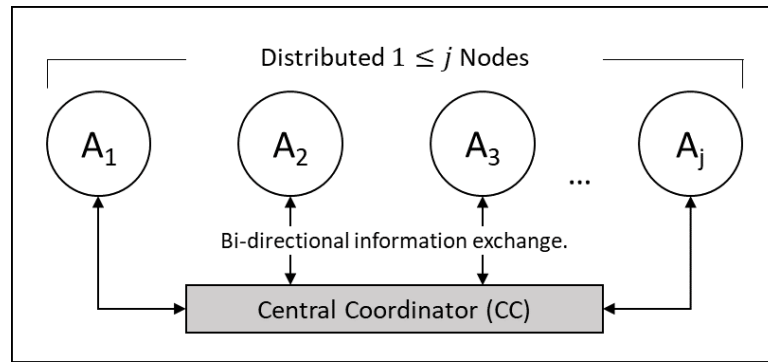
Tarzanagh et al. [32] propose an online scheme to estimate principal eigenspaces for streaming data. They break the incoming batch of data into subsets and allocate those to different computational nodes. The nodes determine the low-rank approximation for the subset assigned to them. They, then, perform local aggregation to estimate the principal eigenspaces, which pass through the fusion center for global estimation. The experiments on real data showed that the proposed algorithm is capable of computing the principal eigenspaces quickly while maintaining the level of approximation accuracy.

There are other distributed methods discussed in the literature, but no relevant research was found whose contribution is specifically on distributed feature extraction using the PCA for an IDS. The motive of this chapter is to propose an approach for distributed feature extraction using PCA to help build a classification model for IDSs.

### 3.3 Methodology

#### 3.3.1 Network Topology

A distributed environment with a fixed number of nodes is simulated where each node only has access to a subset of data. A node performs calculations based on its part of the data and sends the calculated values to the central coordinator for aggregation. Let  $j$  be the number of nodes and  $A = \{A_j : 1 \leq j\}$  be the set of nodes. All nodes have a two-way link with the central coordinator. The network architecture simulated for the experiments is shown in **Figure 3-1**.



**Figure 3-1:** A simulated network topology where each of the  $j$  nodes is connected to the central coordinator for bi-directional information exchanges.

### 3.3.2 Data Distribution

Once the appropriate datasets are discovered and selected, the datasets are split randomly into  $j$  subsets, each with a varying number of rows, and every node is assigned a subset. To simulate a distributed architecture while utilizing a pre-existing dataset, such splitting and assignment of data are conducted. The assumption, however, is that each node is the owner of the data assigned to it. The nodes are unaware of the values in the data present on other nodes. For a dataset  $\mathbf{X}$ , its subsets of data assigned to each node can be represented as

$$\mathbf{X} = \{\mathbf{X}_{A_1}, \mathbf{X}_{A_2}, \dots, \mathbf{X}_{A_j}\}. \quad \text{Eq. 3-5}$$

### 3.3.3 Features Analysis

The PCA works with numerical data; therefore, only the attributes that have continuous or other numerical values are selected.

### 3.3.4 Data Transformation

Each node, individually, processes its part of data through a data preprocessing stage to prepare them for further operations. After executing the rudimentary preprocessing actions like cleaning data, handling missing and inappropriate values, and

filtering out the unneeded features, the data is transformed to ensure its readiness for feature extraction. The essential nontrivial transformations performed are data normalization and class relabeling.

#### 3.3.4.1 Data Normalization

Since PCA works with numerical data, the selected data must be analyzed to identify the attributes that are appropriate for it. The numerical data are normalized in each node to ensure all the values are in the same range. Since both datasets contain various attributes with numerical data, it is crucial to normalize such data to bring them to a standard scale, without affecting their difference in range. The z-score normalization technique is chosen for normalization. The data is spread across multiple nodes; therefore, a straight-forward normalization using **Eq. 2-1** is not possible. The distributed computation of arithmetic mean and approximation of standard deviation, involving local and global computations, are necessary.

##### 3.3.4.1.1 Local Computations

Let  $A_i$  be one of the nodes. For its portion of the data, the local mean of the values in attribute,  $f$ , can be computed as

$$\bar{x}_{f_{A_i}} = \frac{1}{N_{A_i}} \sum_{k=1}^{N_{A_i}} x_{k_{f_{A_i}}}, \quad \text{Eq. 3-6}$$

where  $N_{A_i}$  is the number of rows in  $A_i$ 's dataset and  $x_{k_{f_{A_i}}}$  is the  $k^{th}$  value in  $f$ . The local mean is computed for each of the attributes by every node.

Similarly, the local standard deviation of a sample can be computed as

$$s_{f_{A_i}} = \sqrt{\frac{1}{N_{A_i} - 1} \sum_{k=1}^{N_{A_i}} (x_{k_{f_{A_i}}} - \bar{x}_{f_{A_i}})^2}, \quad \text{Eq. 3-7}$$

where  $\bar{x}_{f_{A_i}}$  is  $A_i$ 's local mean computed for the attribute,  $f$ , derived from **Eq. 3-6**.

The mean and standard deviation are computed locally by each node for the attributes requiring normalization. These are sent to the central coordinator, along with the total number of rows available in each node's part of data; therefore, the node  $A_i$  would send  $N_{A_i}$ ,  $\bar{x}_{f_{A_i}}$ , and  $s_{f_{A_i}}$  to the central coordinator.

### 3.3.4.1.2 Global Computations

After receiving the total number of rows, local means, and local standard deviations for each feature from all the nodes, the central coordinator computes the weighted global average for an attribute with

$$\bar{x}_f = \frac{\sum_{i=1}^j (N_{A_i} \times \bar{x}_{f_{A_i}})}{\sum_{i=1}^j N_{A_i}}, \quad \text{Eq. 3-8}$$

where  $\sum_{i=1}^j N_{A_i} = N$  is the cumulative total number of rows.

The global standard deviation for a specific attribute can be estimated using

$$s_f \approx \sqrt{\frac{\sum_{i=1}^j (N_{A_i} - 1) (s_{f_{A_i}})^2}{(\sum_{i=1}^j N_{A_i}) - j}}, \quad \text{Eq. 3-9}$$

where  $j$  is the number of nodes,  $(s_{f_{A_i}})^2$  is the variance of  $f$  attribute's values in the node  $A_i$ 's dataset.

The computed global averages and standard deviations for each attribute are shared with all the participating nodes by the central coordinator.

### 3.3.4.1.3 Normalization

Each node normalizes the values in its attributes using the z-score normalization technique, with the global mean and standard deviation. This procedure transforms the values spread across multiple nodes into the same scale.

### 3.3.4.2 *Class Relabeling*

The selected datasets are labeled; hence, they have a target variable containing a class label specifying whether a row is an instance of a normal or attack traffic. The attacks are labeled with a specific type of attack in both datasets. The interest is in distinguishing only between the normal and abnormal traffic, so the different labels representing various attacks are grouped into the same class, *Attack*, and all the classes representing the regular traffic are recorded as *Normal*. The class-relabeled datasets are used throughout the dissertation. **Table 3-1** lists the original and the corresponding assigned class labels in each dataset.

**Table 3-1:** The class relabeling in the NSL-KDD and CICIDS2017 datasets.

Dataset	Original Label(s)	Assigned
NSL-KDD	Normal	<i>Normal</i>
	back · buffer_overflow · ftp_write · guess_passwd · imap · ipsweep · land · loadmodule · multihop · neptune · nmap · perl · phf · pod · portsweep · rootkit · satan · smurf · spy · teardrop · warezclient · warezmaster	<i>Attack</i>
CICIDS2017	BENIGN	<i>Normal</i>
	Bot · DDoS · DoS GoldenEye · DoS Hulk · DoS Hulk · DoS Slowhttptest · DoS slowloris · FTP Patator · Heartbleed · Infiltration · PortScan · SSH Patator · Web Attack – Brute Force · Web Attack – Sql Injection · Web Attack – XSS	<i>Attack</i>

### 3.3.5 Distributed Feature Extraction

The distributed feature extraction with multiple nodes involves local and global computations. The centralized extraction is identical to the distributed extraction with a single node.

#### 3.3.5.1 Local Eigen-Decomposition

Each node, after normalizing its portion of data, computes a covariance matrix. Suppose  $\mathbf{K}_1$  is the covariance matrix computed from  $\mathbf{X}_{A_1}$ . The eigenpairs observed after eigen-decomposition of  $\mathbf{K}_1$  forms the set  $Eig_{K_1}$ . Each of the nodes repeats this process. The computed eigenpairs for the node  $A_1$  can be represented as

$$Eig_{K_1} = \left\{ \left( \lambda_{e_{K_1}}, \{v_{1_{e_{K_1}}}, v_{2_{e_{K_1}}}, \dots\} \right) : 1 \leq e \leq n \right\}, \quad \text{Eq. 3-10}$$

where  $n$  is the number of attributes. Each node forwards its eigenpairs to the central coordinator for global aggregation.

#### 3.3.5.2 Global Aggregation

At the global level, the central coordinator compiles the eigenpairs received from each of the nodes. The eigenpairs received from  $j$  number of nodes can be represented as a set,  $Eig = \{Eig_{K_i} : 1 \leq i \leq j\}$ , whose elements are the sets containing the eigenpairs computed by each node. The collected eigenpairs are aggregated by calculating the arithmetic means of the corresponding eigenvalues and eigenvectors. Such aggregation results in a single set of eigenpairs representing the global averages of eigenvalues and their corresponding eigenvectors.

Suppose  $Eig_{Agg}$  is a set containing the aggregated eigenpairs, then the elements in this set are derived by summing each corresponding value of eigenvalues or eigenvectors and dividing the resulting sums by the total number of nodes. The



following sequence represents the globally approximated eigenvalues and their corresponding eigenvectors.

$$Eig_{Agg} = \left\{ \left( \frac{\sum_{i=1}^j \lambda_{e_{K_i}}}{j}, \left\{ \frac{\sum_{i=1}^j v_{1_{e_{K_i}}}}{j}, \frac{\sum_{i=1}^j v_{2_{e_{K_i}}}}{j}, \dots \right\} \right) : 1 \leq e \leq n \right\} \quad \text{Eq. 3-11}$$

The central coordinator shares the aggregated eigenvalues and eigenvectors with all the participating nodes for further processing.

### 3.3.5.3 Local Extraction

When a node,  $A_j$ , receives  $Eig_{Agg}$  from the central coordinator, it forms a projection matrix,  $\mathbf{P}$ , by using the eigenvectors corresponding to the eigenvalues that exceed the defined explained variance threshold. For instance, if the first two eigenvalues exceed the threshold, then the eigenvectors corresponding to them are used to form the projection matrix. The dot product of the transpose of the projection matrix,  $\mathbf{P}^T$ , is taken with the original data to get the PCs. For  $A_j$ , the data containing the PCs it extracts is given by

$$\mathbf{Y}_{A_j} = \mathbf{P}^T \cdot \mathbf{X}_{A_j}, \quad \text{Eq. 3-12}$$

which contains the new features for an IDS classifier.

### 3.3.6 Classification

The classification is performed using the predictive models built using various supervised learning techniques by utilizing the features extracted with the discussed distributed method. The constructed models are trained and tested, and the observations made are reported and analyzed. The k-Nearest Neighbors (k-NN) and Neural Network-based classifiers are constructed because of their known ability to handle continuous values effectively. Similarly, the classification models are also constructed

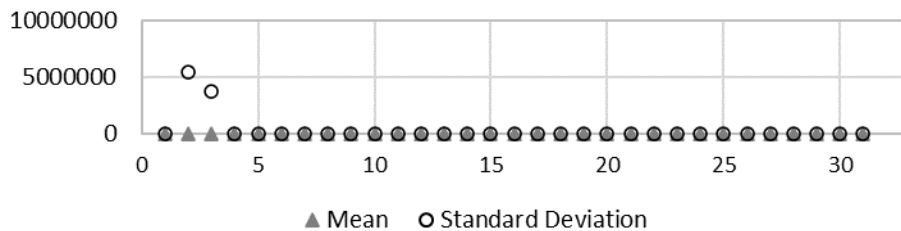
using different variants of Naïve Bayes classifier to examine the performance when the classification is done using continuous values and when done after discretizing the continuous values.

### 3.4 Experimental Procedure & Observations

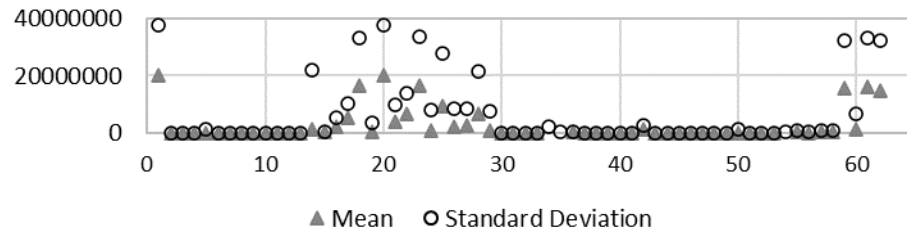
A series of experiments are conducted to verify that the proposed method performs as expected. A multi-node distributed networking environment containing the desired number of nodes and a central coordinator is simulated. The tests are performed on both NSL-KDD and CICIDS2017 datasets.

#### 3.4.1 Data Splitting, Distribution, and Normalization

The data is split randomly to match the number of nodes. Each partition of data, which is assigned to a unique node, has a varied number of rows. The data is normalized using the distributed method discussed in section 3.3.4.1. After normalization, all the numerical attributes are relatively in the same range. The observed global mean and standard deviation of each attribute used for z-score normalization in the NSL-KDD and CICIDS2017 datasets are plotted in **Figure 3-2** and **Figure 3-3**, respectively, to give an idea of the distribution of the values.



**Figure 3-2:** The pre-normalized standard deviation and arithmetic mean for the numerical attributes in the NSL-KDD dataset show that only a few features have an extremely high variance.



**Figure 3-3:** The pre-normalized standard deviation and average for the numerical attributes in the CICIDS2017 dataset show that several features have a high variance.

### 3.4.2 Eigen-Decomposition

Each node computes a covariance matrix for its data portion by using the normalized data. They also perform eigen-decomposition to find the eigenvalues and their respective eigenvectors, which are sent to the central coordinator for aggregation. The central coordinator averages all the corresponding eigenvalues and their eigenvectors to determine the globally aggregated eigenpairs. The eigen-decomposition is performed in both centralized, which involves a single node, and distributed, which involves multiple nodes, manners for comparison. The cumulative explained variance (EV) threshold is set to 95%, so the newly extracted features will retain at least 95% information from the original data. The eigenvalues whose cumulative explained variance exceeds the specified threshold in the NSL-KDD and CICIDS2017 datasets are listed in **Table 3-2** and **Table 3-3**, respectively.

**Table 3-2:** A comparison of the eigenvalues computed by a various number of nodes whose cumulative EV exceeds the threshold of 95% in the NSL-KDD dataset.

<i>PC</i>	Explained Variance in NSL-KDD (%)					
	1 Node	3 Nodes	5 Nodes	10 Nodes	25 Nodes	50 Nodes
1.	20.53	20.53	20.53	20.54	20.91	21.23
2.	15.65	15.65	15.65	15.65	15.92	16.16
3.	7.03	7.88	7.88	8.22	8.28	8.30
4.	5.97	5.98	5.98	5.98	6.09	6.03
5.	4.95	4.95	4.95	4.98	5.17	5.20
6.	4.24	4.24	4.25	4.42	4.55	4.53
7.	3.68	3.70	3.74	3.93	3.96	3.99
8.	3.51	3.59	3.52	3.59	3.58	3.61
9.	3.33	3.35	3.35	3.40	3.37	3.43
10.	3.26	3.25	3.26	3.26	3.31	3.35
11.	3.23	3.23	3.23	3.23	3.28	3.32
12.	3.21	3.20	3.20	3.20	3.25	3.28
13.	3.13	3.15	3.16	3.12	3.13	3.09
14.	3.05	3.01	3.03	2.99	2.96	2.79
15.	2.94	2.88	2.90	2.55	2.42	2.36
16.	2.57	2.38	2.33	2.21	1.97	1.94
17.	2.29	1.85	1.80	1.76	1.62	1.65
18.	1.63	1.54	1.56	1.53	1.49	1.45
19.	1.50	1.39	1.46	1.38	-	-
<b><i>EV Sum</i></b>	<b>95.68</b>	<b>95.76</b>	<b>95.78</b>	<b>95.92</b>	<b>95.25</b>	<b>95.71</b>

**Table 3-3:** A comparison of the eigenvalues computed by a various number of nodes whose cumulative EV exceeds the threshold of 95% in the CICIDS2017 dataset.

<i>PC</i>	Explained Variance in CICIDS2017 (%)					
	1 Node	3 Nodes	5 Nodes	10 Nodes	25 Nodes	50 Nodes
1.	26.13	26.14	26.17	26.27	26.48	26.61
2.	12.78	13.05	12.93	13.93	13.76	14.21
3.	9.22	9.22	9.22	9.23	9.25	9.27
4.	7.18	7.18	7.21	7.28	7.82	7.96
5.	4.91	4.95	4.97	4.99	4.99	4.93
6.	4.22	4.43	4.46	4.37	4.35	4.28
7.	3.79	3.93	4.04	3.96	3.93	3.89
8.	3.50	3.62	3.70	3.54	3.48	3.40
9.	3.25	3.25	3.26	3.23	3.23	3.18
10.	3.09	3.10	3.11	2.91	2.75	2.56
11.	2.28	2.42	2.51	2.42	2.24	2.17
12.	2.11	2.04	2.07	2.04	2.01	1.99
13.	1.96	1.94	1.95	1.94	1.91	1.90
14.	1.91	1.82	1.86	1.85	1.82	1.81
15.	1.77	1.71	1.71	1.71	1.70	1.69
16.	1.59	1.59	1.59	1.58	1.57	1.56
17.	1.57	1.54	1.55	1.53	1.49	1.47
18.	1.52	1.46	1.41	1.43	1.36	1.33
19.	1.38	1.28	1.31	1.24	1.24	1.21
20.	1.27	1.26	-	-	-	-
<b><i>EV Sum</i></b>	<b>95.43</b>	<b>95.92</b>	<b>95.02</b>	<b>95.42</b>	<b>95.41</b>	<b>95.43</b>

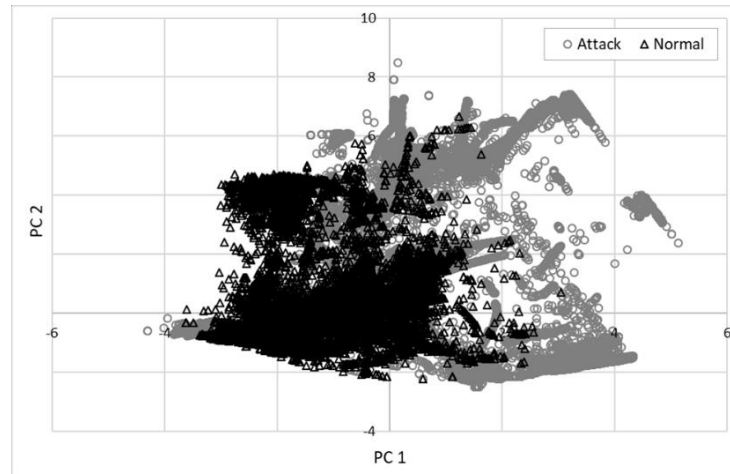
### 3.4.3 Local Feature Extraction

The qualifying eigenvalues' eigenvectors, based on the specified threshold, are used to construct a projection matrix. Each node then takes a dot product of the transpose of the projection matrix with its original data to project the data into a lower dimension, which results in dimensionality reduction. The NSL-KDD dataset had 32, and the CICIDS2017 dataset had 63 original numeric dimensions. These are reduced to 19 or 18 and 20 or 19 dimensions, depending on the number of nodes used, respectively. **Table 3-4** shows the new dimensions for each dataset after data extraction.

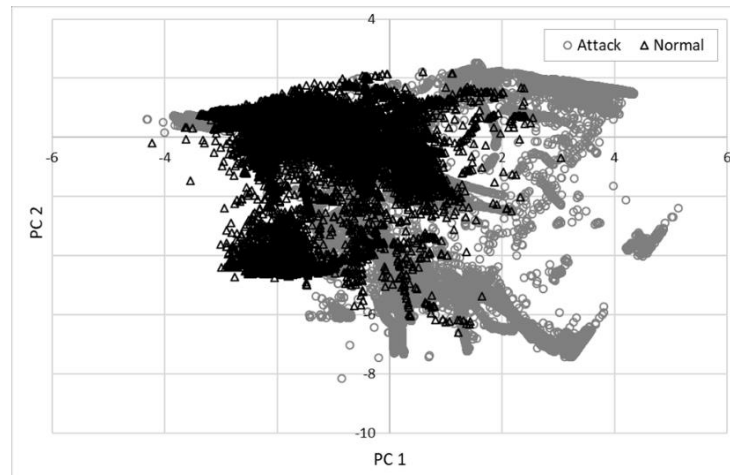
**Table 3-4:** The number of new dimensions observed after feature extraction from the NSL-KDD and CICIDS2017 datasets.

Dataset	New Dimension					
	1 Node	3 Nodes	5 Nodes	10 Nodes	25 Nodes	50 Nodes
NSL-KDD	19	19	19	19	18	18
CICIDS2017	20	20	19	19	19	19

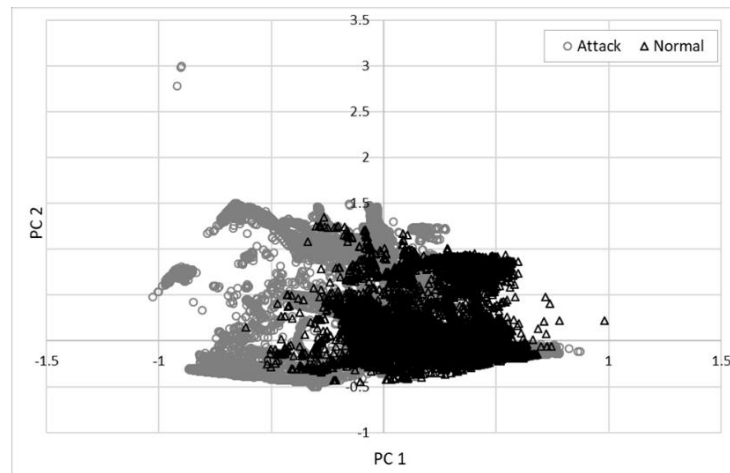
The new data so generated after projection is described by the PCs, which are the extracted features. The first PC holds the most information about the data, and the last PC holds the least amount of information. Based on the comparisons between the first two PCs, it is observed in the NSL-KDD dataset that even though the orientation of the extracted data has changed, the general explained variance has remained relatively constant. In the CICIDS2017 dataset, the principal components appear to have shifted more drastically, as the number of nodes changed. The comparisons of the first versus second PCs determined with various nodes in each dataset are displayed in the following charts.



1 Node Extraction

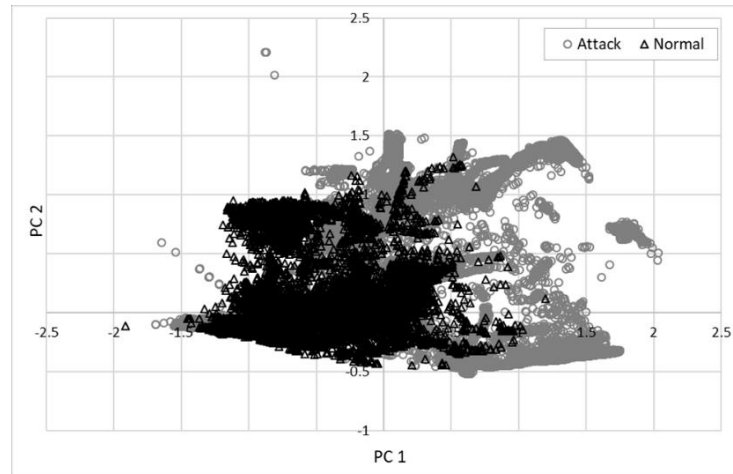


3 Nodes Extraction

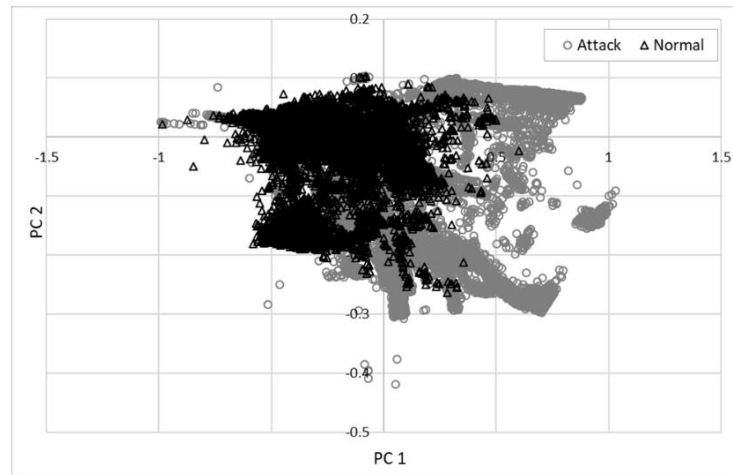


5 Nodes Extraction

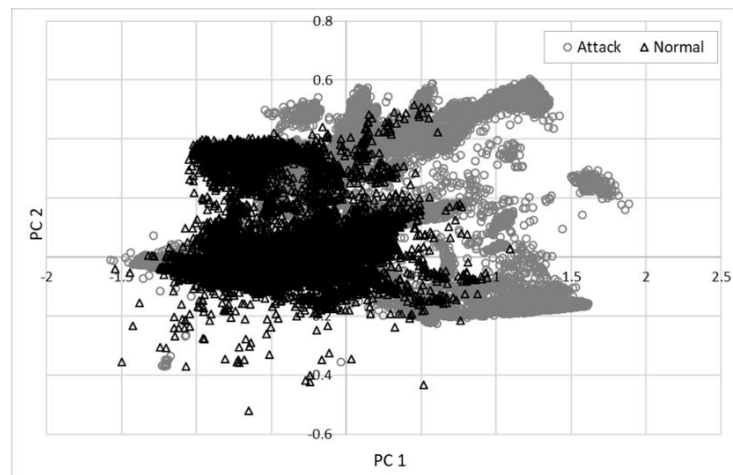
**Figure 3-4:** The first versus second PCs extracted with 1, 3, and 5 nodes in the NSL-KDD dataset. The variances explained by PC 1 and PC 2, respectively, in 1-node, 3-node, and 5-node extractions are the same.



10 Nodes Extraction



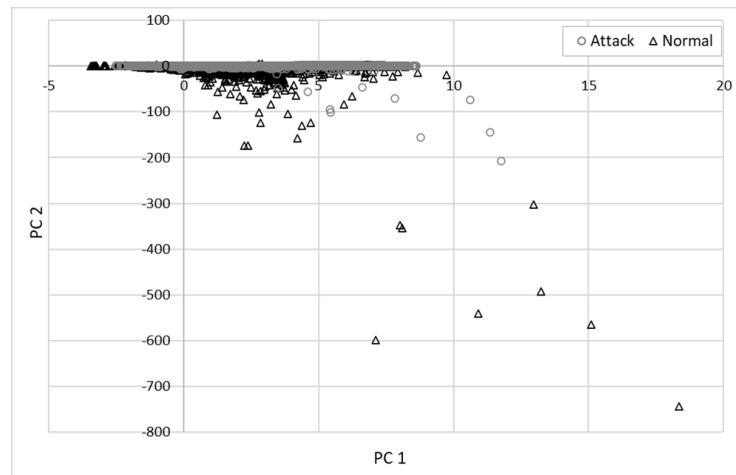
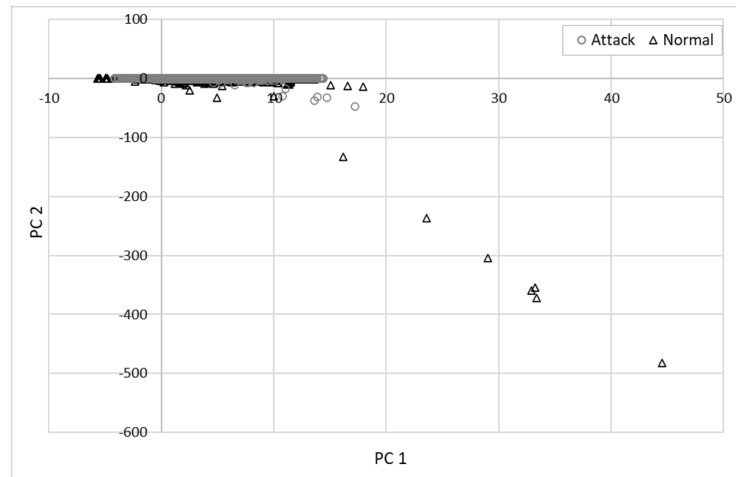
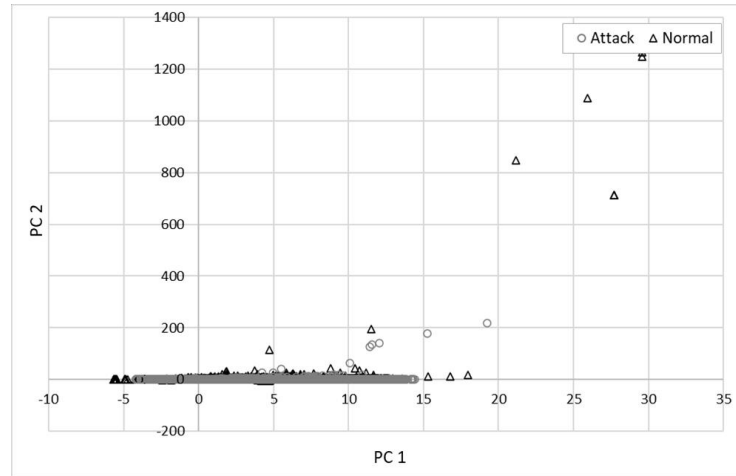
25 Nodes Extraction



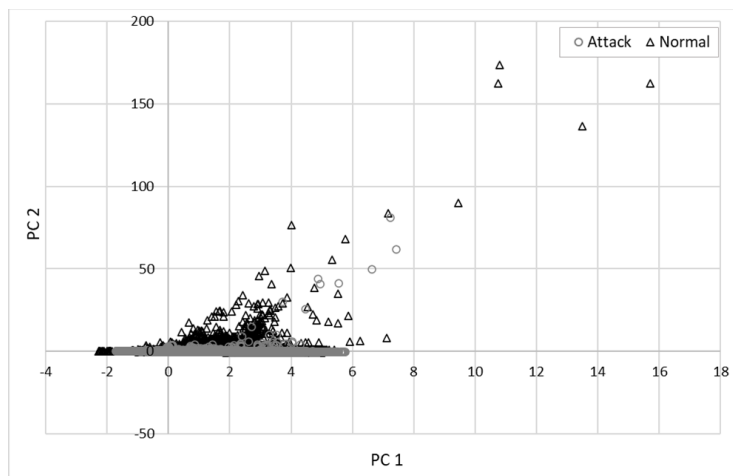
50 Nodes Extraction

**Figure 3-5:** The plots of the first versus second PCs extracted with 10, 25, and 50 nodes in the NSL-KDD dataset. The variances explained by PC 1 and PC 2, respectively, in these extractions are consistent.

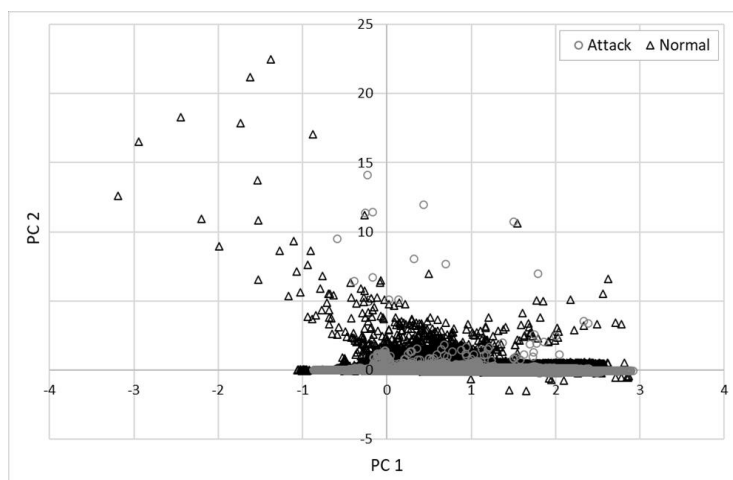




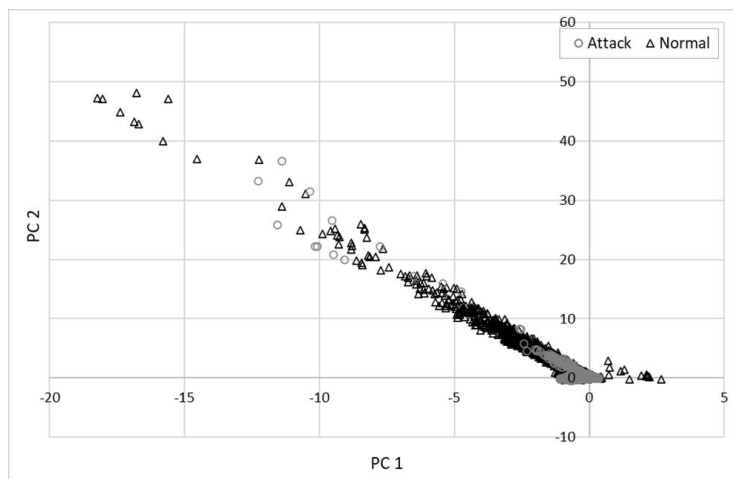
**Figure 3-6:** The first versus second PCs extracted with 1, 3, and 5 nodes in the CICIDS2017 dataset. PC 1 and PC 2 in these extractions look somewhat correlated.



10 Nodes Extraction



25 Nodes Extraction



50 Nodes Extraction

**Figure 3-7:** The first versus second PCs extracted with 10, 25, and 50 nodes in the CICIDS2017 dataset. The plots morph more rapidly with the increase in the number of nodes in this dataset.

#### 3.4.4 Classification Model-Building

To verify the effectiveness of the extracted features, various classifiers are built and tested before and after extracting the features. All the values are numeric, so they are first normalized. When using the original features, the chi-square statistic-based feature selection technique is utilized to select the top 19 features. The chi-square requires discretized data. The bin size of 1,000 is used for discretization. The classifiers tested are based on Naïve Bayes, Neural Network, and k-NN. Naïve Bayes is known to perform well with discretized data, so the same discretized data used for chi-square-based analysis is used for it. Neural Network and k-NN handle continuous data. The primary purpose of the experiments is to examine how the number of nodes used affects the quality of the extracted features. The inter-classifier performance comparison is not the main motive. The parameters set for Neural Network are as follows — training cycle: 100, learning rate: 0.03, momentum: 0.4, and hidden layers: 2. Similarly, for k-NN, the Euclidean distance measure with  $k = 5$  is used. The built models are validated using the k-fold cross-validation technique with 5 folds.

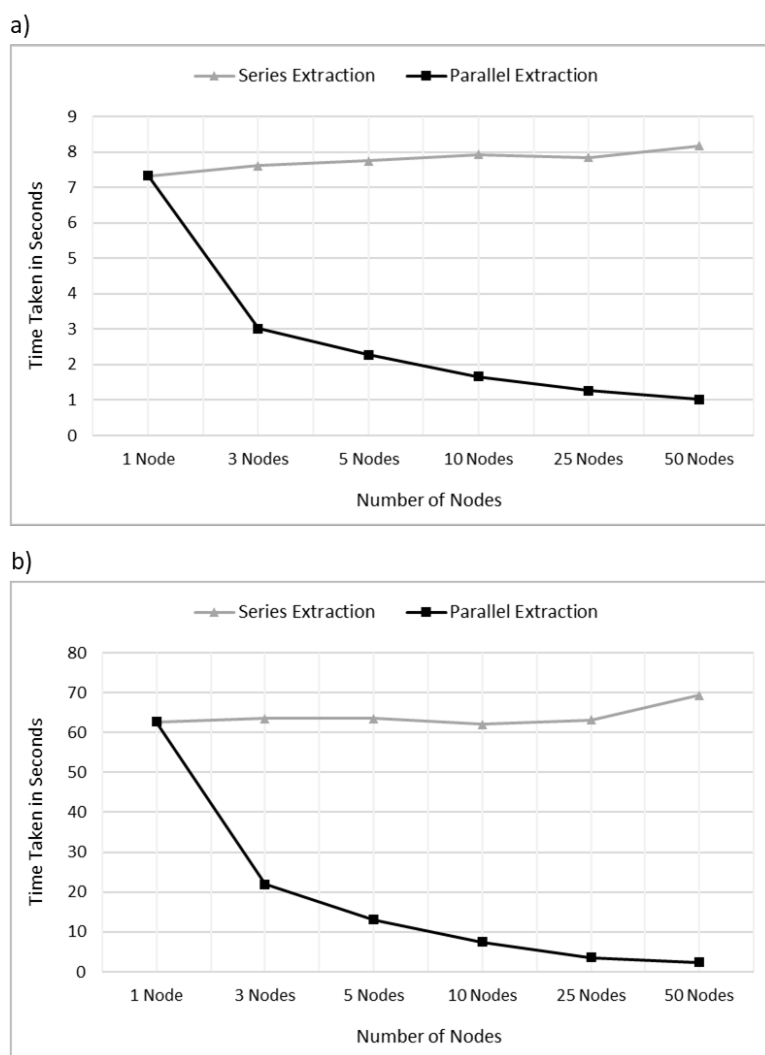
### **3.5 Results and Discussion**

This section presents and analyzes the time needed for feature extraction with a different number of nodes and the performance of the classifiers built using the features extracted in centralized and distributed manners.

#### 3.5.1 Time Analysis of Feature Extraction

The time consumed by the series extraction, where each node waits for another node to finish its task before proceeding, and by the parallel extraction, where the nodes work simultaneously, are recorded. One of the benefits of using a distributed feature

extraction technique is the reduction in time required for feature extraction. The time needed for extraction significantly reduced when done in parallel. The time taken to extract features stayed quite constant when done in series; however, when done in parallel, the time taken decreased as the number of nodes increased. It appears that for massive datasets, the distributed feature extraction done in parallel takes a significantly shorter time. **Figure 3-8** depicts the reduction of time taken when the features are extracted in parallel with multiple nodes.



**Figure 3-8:** The comparison between the time taken to extract the features from the a) NSL-KDD and b) CICIDS2017 datasets with a various number of nodes. 50-node parallel extraction is much faster than centralized extraction.

In the NSL-KDD dataset, the 50-node parallel feature extraction took 1.02 seconds and was 7.21 times faster than the central extraction that took 7.33 seconds. In a significantly larger CICIDS2017 dataset, the 50-node extraction took only 2.34 seconds, which was 26.74 times faster than 62.69 seconds that the central extraction took.

### 3.5.2 Classification with Original Features

The level of accuracy observed, when using the original features, ranged between 88.75% and 97.94% in the NSL-KDD dataset and between 92.47% and 98.71% in the CICIDS2017 dataset. The k-NN-based classifiers performed the best, and the Naïve Bayes-based ones performed the worst, in general, as seen in **Table 3-5** and **Table 3-6**.

**Table 3-5:** The comparison of performances between different classifiers built with the original features in the NSL-KDD dataset. k-NN performs the best with an accuracy of 97.94%.

Classifier	Recall	Precision	Specificity	Accuracy
Naïve Bayes	79.45	96.58	97.39	88.75
Neural Network	96.87	96.23	96.52	96.69
k-NN	97.97	97.75	97.92	97.94

**Table 3-6:** The comparison of performances between different classifiers built with the original features in the CICIDS2017 dataset. k-NN performs the best with an accuracy of 98.71%.

Classifier	Recall	Precision	Specificity	Accuracy
Naive Bayes	98.20	93.69	93.38	95.79
Neural Network	87.72	96.93	97.22	92.47
k-NN	98.89	98.54	98.54	98.71

It must be noted that even though k-NN appears to perform the best in terms of overall accuracy, the time taken (~5 hours) to build and validate each classifier based on it was significantly longer than what other algorithms took. The Naïve Bayes-based classifiers took the shortest time, which was only a fraction of what the Neural Network-based classifiers took.

### 3.5.3 Classification with Extracted Features

All the extracted features are used to build the classification models. The number of features varies based on the number of nodes and the dataset used. The performance of every classifier of the same type stayed reasonably consistent even when using the features extracted with a different number of nodes. Just like with the original features, the classifiers based on k-NN performed better on both datasets.

In the NSL-KDD dataset, the highest accuracy of 98.49% was achieved by the k-NN-based classifier when using the centrally extracted features. The Naïve Bayes-based classifier was the worst performer with the lowest accuracy of 91.92% when using the features extracted with 5 nodes. **Table 3-7** reports the performances of the classifiers constructed and validated for the NSL-KDD dataset using the features extracted in both centralized and distributed manners.

**Table 3-7:** The comparison of performances between different classifiers built with the features extracted using a various number of nodes for the NSL-KDD dataset. The overall performance of the k-NN-based classifier is the best.

Classifier	Nodes	Recall	Precision	Specificity	Accuracy
Naïve Bayes	1	90.98	93.01	93.66	<b>92.37</b>
	3	89.79	93.50	94.21	92.08
	5	91.28	91.87	92.51	91.92
	10	91.05	92.33	92.99	92.05
	25	<b>93.84</b>	89.55	90.70	92.14
	50	88.52	<b>94.75</b>	<b>95.46</b>	92.12
Neural Network	1	94.69	96.82	97.11	95.95
	3	92.96	<b>96.86</b>	<b>97.20</b>	95.16
	5	96.79	92.92	93.67	95.11
	10	97.03	95.40	94.96	<b>96.03</b>
	25	94.93	95.84	96.18	95.58
	50	<b>97.11</b>	95.03	94.53	95.86
k-NN	1	<b>98.58</b>	98.28	<b>98.40</b>	<b>98.49</b>
	3	98.35	98.18	98.31	98.33
	5	98.23	98.08	98.22	98.22
	10	98.22	<b>98.35</b>	98.23	98.23
	25	98.12	98.07	98.21	98.16
	50	98.24	97.99	97.83	98.04

In the CICIDS2017 dataset, the centrally extracted features gave an accuracy of 99.71% with the k-NN classifier. The accuracy appears to fluctuate more significantly with the increase in the number of nodes in this dataset — with the lowest observed accuracy for the k-NN-based classifier being 99% when using the features extracted with 50 nodes. Each type of classifier’s corresponding accuracies, however, stayed somewhat

within the same range. **Table 3-8** shows the performances of the classifiers constructed and validated for the CICIDS2017 dataset using the features extracted in both centralized and distributed manners.

**Table 3-8:** The comparison of performances between different classifiers built with the features extracted using a various number of nodes for the CICIDS2017 dataset. The observed performance is comparable to the classifiers that use the centrally extracted features.

Classifier	Nodes	Recall	Precision	Specificity	Accuracy
Naïve Bayes	1	93.56	89.59	89.13	<b>91.34</b>
	3	<b>94.23</b>	86.15	84.85	89.54
	5	92.68	86.03	84.95	88.81
	10	93.47	83.76	81.88	87.67
	25	92.65	86.17	85.13	88.89
	50	86.83	<b>90.60</b>	<b>91.00</b>	88.91
Neural Network	1	92.02	92.89	92.95	92.48
	3	93.80	92.14	91.99	92.90
	5	90.12	93.16	93.39	91.76
	10	<b>94.43</b>	90.65	90.26	92.34
	25	90.70	<b>96.07</b>	<b>96.29</b>	<b>93.50</b>
	50	87.46	95.52	95.90	91.68
k-NN	1	<b>99.77</b>	<b>99.65</b>	<b>99.65</b>	<b>99.71</b>
	3	99.66	99.49	99.48	99.57
	5	99.36	99.63	99.63	99.50
	10	99.50	99.22	99.22	99.36
	25	98.99	99.37	99.37	99.18
	50	98.76	99.24	99.25	99.00



### 3.5.4 FPR Analysis

The FPRs of each of the classifiers constructed using the extracted features are compared and analyzed. The best achievements in terms of FPR by each classifier are seen in **Table 3-9**.

**Table 3-9:** The best FPR achieved by each of the tested classifiers on the respective datasets. The lowest FPRs were achieved by the k-NN.

Dataset	Classifier	Nodes	Test Instances	False Positives	FPR
NSL-KDD	Naïve Bayes	50	148,517	3,502	4.52
	Neural Network	3	148,517	2,154	2.89
	k-NN	1	148,517	1,231	1.60
CICIDS2017	Naïve Bayes	50	1,000,000	45,021	9.00
	Neural Network	25	1,000,000	18,531	3.71
	k-NN	1	1,000,000	7,308	1.46

It must be acknowledged that even for a low FPR, the number of normal instances falsely predicted to be an attack can still be overwhelmingly high. The FPR for the k-NN-based classifier on CICIDS2017 dataset is only 1.46%, but the number of instances falsely identified as an attack is 7,308. If those instances are to be reviewed manually to verify the correctness of the classification, it could consume a significant amount of resources.

## 3.6 Conclusions

In this chapter, we discussed a distributed feature extraction method to build a classifier for an IDS. It was based on PCA's underlying principles. The nodes computed the eigenpairs from their subset of the data locally. These computed eigenpairs were sent to the central coordinator for aggregation. With the globally approximated eigenpairs,

each node extracted the features from its portion of the dataset. By extracting the features using multiple nodes, the time required for extraction was reduced significantly. For the larger dataset, CICIDS2017, the extraction using 50 nodes took only 2.34 seconds, which was 26.74 times faster than when done centrally. Extracting features in this manner also reduced the amount of data needed to be transferred to the central coordinator, requiring lower bandwidth and storage.

We constructed several classifiers using the extracted features to verify their usefulness. These classifiers were validated using the k-fold cross-validation technique. All the classifiers of the same type performed fairly evenly, regardless of the number of nodes used to extract the features. The k-NN-based classifiers performed better consistently over other classifiers. The best performances attained by the k-NN-based classifiers were the accuracy of 98.49% in the NSL-KDD dataset and 99.71% in the CICIDS2017 dataset. Both best performances observed were for the features extracted using only one node, which is equivalent to the centralized extraction. Despite this, the performances of the classifiers built using the features extracted with multiple nodes were comparable to the best performing classifiers that used the centrally extracted features. The worst performing k-NN-based classifier, which used the features extracted by 50 nodes in the NSL-KDD dataset, still achieved an accuracy of 98.04%. The same for the CICIDS2017 dataset was 99%. Even though the k-NN based classifiers performed the best in terms of accuracy, the time required to construct them was significantly longer than what was required for the Naïve Bayes-based classifiers.

With these observations, we conclude that there can be some degradation in performance when using the features extracted in a distributed manner; however, the

centrally extracted features tend to perform only slightly better over the features extracted distributedly. The benefits like the reduced time needed to extract the features, the applicability in a distributed network environment, and the relieved stress on an IDS could make the distributed feature extraction worthwhile.

## **CHAPTER 4**

### **DISTRIBUTED CONSTRUCTION OF A PREDICTION MODEL**

#### **4.1 Background**

The IDS classifiers that are constructed and implemented in a centralized manner continue to suffer from long training duration, slow detection, and poor robustness. A load on an IDS could be distributed across several nodes to relieve the stress on a single IDS. Doing so could improve the learning and detection speeds. Similarly, since the network traffic-related data are rapidly generated and collected from various sources, transferring them regularly to the central system for detecting intrusions can be detrimental. If the data available in distributed nodes are utilized to construct an IDS classifier collaboratively without having to transfer those data to the central coordinator, then that would reduce the required total bandwidth while also better preserving the privacy of data. If those nodes could individually perform traffic monitoring and scanning using the classifiers constructed collaboratively, then the robustness would improve.

This chapter discusses the distributed construction of a classifier based on the Bayes' theorem for an IDS. The collaboratively constructed model's effectiveness is examined in terms of training duration, detection speed, and classification performances. The Bayes' theorem helps determine the probability of an event occurring given that

certain events have occurred. It calculates the posterior probability based on prior probabilities [33]. For two events,  $A$  and  $B$ , the Bayes' theorem is the conditional probability of  $A$  occurring if  $B$  has occurred, which is given by

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}, \quad \text{Eq. 4-1}$$

where  $P(B | A)$  is the conditional probability of  $B$  occurring if  $A$  has occurred, and  $P(A)$  and  $P(B)$  are marginal probabilities of  $A$  and  $B$  occurring, respectively. The Bayes' theorem is used in many classification applications [34].

If  $F = \{f_1, f_2, \dots, f_k\}$  is the set of  $k$  features in a dataset and  $C = \{c_1, c_2, \dots, c_m\}$  is the set containing  $m$  distinct class labels, then for a new unknown instance,  $x$ , the classification based on Bayes' theorem is done using

$$c(x) = \arg \max_{c \in C} P(c) \cdot P(v_1, v_2, \dots, v_k | c), \quad \text{Eq. 4-2}$$

where  $c(x)$  is the predicted class for  $x$  and  $\langle v_1, v_2, \dots, v_k \rangle$  is a feature vector containing the values from the respective features [35].

Naïve Bayes is a widely-known statistical classification technique based on Bayes' theorem that naïvely assumes all the features to be independent [36]. It is known to perform well, even when this assumption of independence is violated to some extent. The strengths of Naïve Bayes include low storage requirements, high scalability, and the ability to train and make predictions quickly [37]. There are different variants of Naïve Bayes. The Gaussian Naïve Bayes works with continuous features like the ones extracted in CHAPTER 3. The categorical Naïve Bayes, however, demonstrated superior performance over the Gaussian Naïve Bayes during experiments; therefore, the discretized values from the extracted features have been used to construct a categorical Naïve Bayes classifier.

Once the classifier is built, as the new training data becomes available, the incremental training of Naïve Bayes depends only on updating the frequency counts, which makes it highly scalable. Most algorithms used to construct a classifier require extensive retraining to fit both the new and old data when the new information becomes available. If the model is not retrained to fit the newly available information, then the classifier can gradually lose its effectiveness and become obsolete. For the models that require retraining, figuring out the best time to retrain the model is challenging [38]. The fact that Naïve Bayes works by merely counting the frequencies and does not require expensive retraining, as long as the frequency counts are kept up-to-date when the new instances are identified, makes it a solid choice for an IDS classifier.

## 4.2 Related Works

The interest in the construction and implementation of a collaborative IDS has grown gradually over the years. The different types of collaborative IDS approaches have been surveyed in [39]. Many recent studies on collaborative IDS seem to focus on privacy preservation, robustness improvement, and overhead reduction. Some relevant works in the literature are briefly discussed here.

Toulouse et al. [33] propose a wholly distributed network IDS that works by detecting anomalies. Their proposed method is based on the Naïve Bayes classifier, where the probabilities computed by one node are shared with other nodes through an iterative average consensus protocol. The authors show that their consensus-based model has a lower communication overhead in comparison to other distributed methods.

In [40], the authors study the distributed machine learning that is suitable when the data is distributed across several parties, and those parties do not wish to share the

raw data they possess with others. Through their study, the authors propose utilizing the asynchronous stochastic gradient descent (SGD) algorithm to learn from the distributed features collaboratively. Their proposed technique is capable of learning even when the original features or the local model parameters are not shared with others.

Similarly, [41] proposes a modified Naïve Bayes algorithm for intrusion detection classification that is based on an artificial bee colony algorithm. The authors compare their version of the algorithm with other competing algorithms and successfully demonstrate that their algorithm performs better than the competitors. Through some experiments, they show that their method gets over 91% accuracy in the NSL-KDD dataset. Fung et al. [42] present a collaborative framework for intrusion detection networks that uses a Bayesian approach for feedback aggregation.

A thorough literature review reveals that despite some achievements, further advancement is needed to ensure that the IDSs can keep up with the shifting dynamics of the network ecosystem that has growingly adapted to the distributed architecture.

### 4.3 Methodology

#### 4.3.1 Data Preparation & Transformation

Suppose  $A = \{A_1, A_2, \dots, A_j\}$  is a set containing all the nodes and  $X = \{X_1, X_2, \dots, X_j\}$  is the set of data on each corresponding node. Each part of the data has the same number of features. If they have  $k$  features each, then for the part of data,  $X_j$ , the set of its features is represented as  $F_{X_j} = \{f_{1_{X_j}}, f_{2_{X_j}}, \dots, f_{k_{X_j}}\}$ . Since the number of rows varies in each part, let  $N = \{n_{A_1}, n_{A_2}, \dots, n_{A_j}\}$  be the set containing the number of rows for each data part distributed across  $j$  nodes.

The datasets containing the features extracted in CHAPTER 3 and their corresponding class labels are the source of data for this chapter. It is assumed that each participating node has its own set of data containing the extracted features and class labels. For a network with  $j$  nodes, there are  $j$  parts of data with each consisting a varying number of rows. The testing set is created by randomly sampling and separating the sampled instances from the data extracted by the nodes. The training is done using the remaining data. The ratio of training and testing sets is about 4:1.

Some data transformations are necessary to prepare the data for further processing. Because the data is decentralized, the alterations must be done in a distributed way. The two main transformations include data standardization and binning.

#### 4.3.1.1 Standardization

Even though the principal components in CHAPTER 3 are extracted using the standardized data, the post-extraction data are no longer in a standard form; therefore, the data is standardized by using the same technique described in section 3.3.4.1.

#### 4.3.1.2 Discretization by Binning

The fixed-width binning is performed after standardization to discretize the continuous values in each dataset. Because the data is distributed, the binning must be done collaboratively. Each node evaluates the values in its features and sends the minimum and maximum values in its features to the central coordinator.

Suppose  $MinMax_{A_j} = \left\{ \left( f_{i_{A_j}}, \left\{ v_{f_{i_{A_j}, \min}}, v_{f_{i_{A_j}, \max}} \right\} \right) : 1 \leq i \leq k \right\}$  is the set containing the features,  $f_{i_{A_j}}$ , in node,  $A_j$ , and their corresponding observed maximum and minimum values. Each respective node shares this with the central coordinator. The central coordinator determines the global minimum and maximum values for each feature based



on the information received from all the nodes. It, then, with some specified number of bins, computes the width of bins for each feature using **Eq. 3-2**. Once the widths are computed, they are shared with all the nodes along with the observed global minimum value for each feature. The nodes, with the received widths and respective universal minimum values, perform binning. The binning so undertaken is consistent and puts the values belonging to the same range in the same bin across all nodes.

#### 4.3.2 Features Analysis and Selection

The available features are analyzed to ensure their usability to construct a classifier. Selecting the most suitable features and removing the unnecessary features impact the performance of the classifier. Because Naïve Bayes-based classifier assumes independence among attributes, it is crucial to ensure that only the considerably uncorrelated features that hold the most information about the class are selected. The principal components being used as features are orthogonal to one another, so they are considered independent, but the feature analysis is still conducted to identify the most suitable features to build a classifier. Since the datasets are distributed across several nodes, each node performs a chi-square statistic-based analysis to test the independence of the existing features only on its part of data.

The chi-square test of independence is one of the statistical methods to examine the dependency between two variables, which is given by

$$x_d^2 = \sum_{i=1}^r \sum_{j=1}^u \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}, \quad \text{Eq. 4-3}$$

where  $d$  is the degrees of freedom, such that  $d = (r - 1) \cdot (u - 1)$ ,  $r$  is total rows,  $u$  is total columns, and  $O_{i,j}$  and  $E_{i,j}$  are the observed and expected values of two categorical attributes, respectively. This method ranks each feature based on its dependency on the

target variable. It identifies the features that have a low reliance on other features but a high dependence on the class.

All the features and their ranking weight computed on each node are shared with the central coordinator. The central coordinator averages the weights received from the nodes and sends the features along with their corresponding averaged weight back to the nodes. The central coordinator also instructs each node to use the top  $n$  features for classifier construction.

#### 4.3.3 Naïve Bayes Classifier

The Naïve Bayes classifier is one of the Bayesian Network Classifiers that makes a bold assumption of the features being independent [35]. If  $\langle v_1, v_2, \dots, v_k \rangle$  is the feature vector containing the values for each of the respective  $k$  features; then, the classification done using this method is expressed as

$$c(x) = \arg \max_{c \in C} P(c) \cdot \prod_{i=1}^k P(v_i | c), \quad \text{Eq. 4-4}$$

where  $\prod_{i=1}^k P(v_i | c)$  is the product of all class-specific conditional prior probabilities of the values in each feature. When the data is distributed, each node must send some relevant information on its data to the central coordinator to be able to build the classification model. This process is discussed in the following sections.

#### 4.3.4 Distributed Model-Building

The global frequency of every value in each feature associated with a class that is representative of the entirety of data must be determined in a distributed manner. Additionally, to compute the class-specific prior probabilities, the class frequencies must also be computed collaboratively.

#### 4.3.4.1 Data Separation by Class

It is necessary to segregate the data by class labels to ease counting the class-specific frequency of each value in every feature. The class-segregated data in a node  $j$  can be expressed as  $X_j = \{X_{j_{c_1}}, X_{j_{c_2}}, \dots, X_{j_{c_m}}\}$ . The  $m$  different classes in a dataset can be represented as  $C = \{c_1, c_2, \dots, c_m\}$ . Once the data is separated based on the association with the class labels, the frequencies are counted.

#### 4.3.4.2 Local Frequency Counting

Every node counts the number of rows in its dataset representing each class. Similarly, every node also computes the class-specific frequency of each value in a feature. Let  $f$  represent the frequency and  $\mathcal{F}$  represent the set of frequencies. For  $m$  distinct classes in  $C$ , a set containing the frequencies for each class in node  $A_j$  can be expressed as

$$\mathcal{F}_{c_{A_j}} = \{f_{c_{i_{A_j}}} : 1 \leq i \leq m\}. \quad \text{Eq. 4-5}$$

Similarly, if there are  $p$  unique values in a feature  $f_k$  associated with the class  $c_m$  in node  $A_j$ , then the set containing class-specific frequencies for each value can be represented as

$$\mathcal{F}_{v_{f_k c_m A_j}} = \{f_{v_{i_{f_k c_m A_j}}} : 1 \leq i \leq p\}. \quad \text{Eq. 4-6}$$

Each node determines the class-specific value frequencies for all unique values in each of the features. The calculated class frequencies and class-specific value frequencies are sent to the central coordinator.

#### 4.3.4.3 Global Frequency Counting

The central coordinator uses the collected local frequencies to compute the global frequencies that are representative of the cumulative data.

#### 4.3.4.3.1 Class Frequencies

All the corresponding class frequencies from each node are summed to get the total frequency that is representative of the entire global data. Based on the class frequencies received from each node, a set containing global class frequencies can be determined as

$$\mathcal{F}_c = \left\{ \sum_{n=1}^j \mathcal{F}_{c_i A_n} : 1 \leq i \leq m \right\}, \quad \text{Eq. 4-7}$$

where  $m$  is the number of classes and  $j$  is the number of nodes. Based on the elements in  $\mathcal{F}_c$ , the total number of rows,  $N$ , in all the data parts combined can be computed as

$$N = \sum_{i=1}^m \mathcal{F}_{c_i}. \quad \text{Eq. 4-8}$$

The central coordinator shares the set containing the computed global frequency of each class,  $\mathcal{F}_c$ , and the total number of rows,  $N$ , with each participating node.

#### 4.3.4.3.2 Class-Specific Value Frequencies

The central coordinator compiles all the local class-specific frequencies for each value in every feature. For a feature  $f_k$  associated with class  $c_m$ , the global class-specific value frequencies for  $p$  unique values can be determined as

$$\mathcal{F}_{v_{f_k c_m}} = \left\{ \sum_{n=1}^j \mathcal{F}_{v_{f_k c_m} A_n} : 1 \leq i \leq p \right\}. \quad \text{Eq. 4-9}$$

Such computation is repeated for every feature and class. The sets containing global class-specific value frequencies for each feature and class are also shared with the nodes.

#### 4.3.5 Local Prior-Probabilities Computation

After each node has access to the information containing the class frequencies and class-specific value frequencies, it can individually compute the necessary probabilities

to perform the Naïve Bayes classification. If  $P_c$  is the set of class probabilities, then its elements are computed as

$$P_c = \left\{ \frac{\mathcal{F}_{c_i}}{N} : 1 \leq i \leq m \right\}. \quad \text{Eq. 4-10}$$

Similarly, if  $P_{v_{f_k c_m}}$  is the set containing prior probabilities of  $p$  unique values appearing in a class,  $c_m$ , for a feature,  $f_k$ , then its elements are computed as the ratio between the value's and class' frequencies, given by

$$P_{v_{f_k c_m}} = \left\{ \frac{\mathcal{F}_{v_{i f_k c_m}}}{\mathcal{F}_{c_m}} : 1 \leq i \leq p \right\}, \quad \text{Eq. 4-11}$$

where  $\mathcal{F}_{v_{i f_k c_m}}$  is the global frequency of the  $i^{\text{th}}$  value in a feature  $f_k$  for the class  $c_m$  and  $\mathcal{F}_{c_m}$  is the global frequency of the class  $c_m$  from the set  $\mathcal{F}_c$ . Such prior probabilities are computed for the values in all the features associated with each class.

Both class and value probabilities are necessary to perform classifications. The prediction model construction using Naïve Bayes depends only on these prior probabilities, so it is a quick learning algorithm.

#### 4.3.6 Classification

Any participating node can classify a new unknown instance according to the **Eq. 4-4** by using the available global prior probabilities. The likelihood of each class being the right one for a newly observed instance is computed. An instance is then classified into the class that has the highest probability of being the correct one.

#### 4.3.7 Validation

The predictors constructed using a distributed approach is validated on each of the nodes. The testing set is created by randomly sampling the instances and separating them from the data on each node. The remaining data is used for training, which involves

frequency counting and probability calculations. This process follows the holdout cross-validation method, with 80% of data used for training, and the remaining 20% held-out data used for testing. The results observed using the classifiers constructed with a different number of nodes are compared against each other to analyze the outcomes.

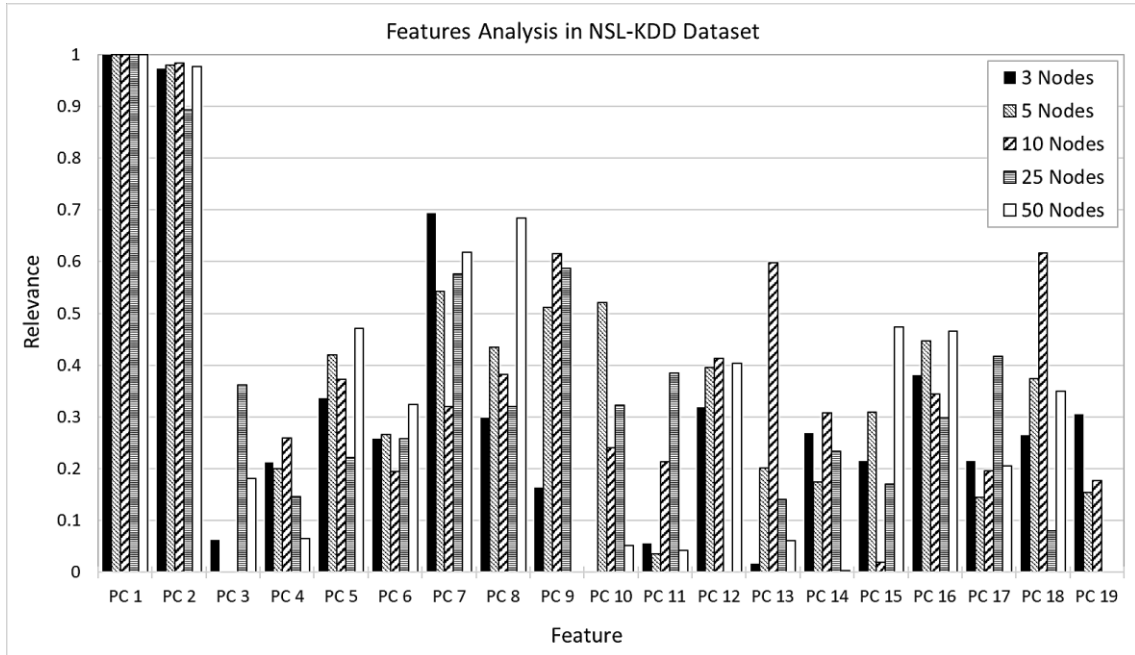
## 4.4 Experimental Procedure & Observations

### 4.4.1 Data Transformation & Splitting

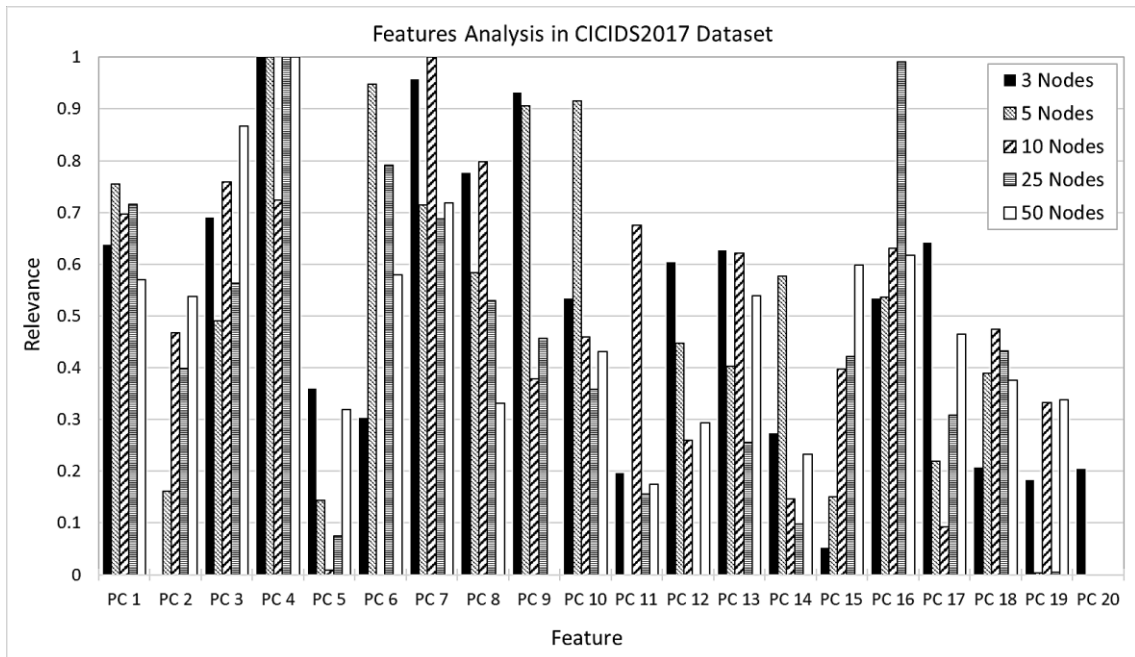
The data in each node are standardized using a collaborative method explained in 3.3.4.1. After standardization, these data are discretized to make them appropriate for the chi-square statistic-based analysis and categorical Naïve Bayes. The discretization is also done collaboratively, as described in 4.3.1.2, using the equal-width binning method with 1,000 bins for both NSL-KDD and CICIDS2017 datasets. The 20% of instances in the nodes are randomly sampled and separated to form a testing set. The rest of the data are used as the training set. The training part of the dataset is used for feature selection, frequency counting, and other training-related procedures. The testing part of the data is used solely for testing and validation.

### 4.4.2 Features Analysis and Selection

The features extracted in CHAPTER 3 are analyzed, after standardization and discretization, using the chi-square test of independence technique to identify the best features to build a classifier. The analyzed features, along with the normalized relevancy weights assigned to them, are sent to the central coordinator. The central coordinator averages the relevancy weights. The top 15 features with the most significant aggregated weights are selected for model-construction. **Figure 4-1** and **Figure 4-2** show the relevance of the features extracted in distributed manners.



**Figure 4-1:** The relevance of each feature determined by the chi-square test of independence in the NSL-KDD dataset using a varying number of nodes. PC 1 and PC 2 are consistently identified as the two most relevant features.



**Figure 4-2:** The relevance of each feature determined by the chi-square test of independence in the CICIDS2017 dataset using a varying number of nodes. PC 4 is most-frequently identified as the most relevant feature.

#### 4.4.3 Data Separation by Class

Each node segregates its part of the data by class. Such segregation is optional, but it is easier to count the class and value frequencies when the instances belonging to the same class are grouped. All pieces of the data have two classes — *Attack* and *Normal*. Through analysis of the segregated data, it is evident that even though the data were randomly sampled and split across several nodes, the class balance is still mostly maintained. Such a balanced dataset, where all the classes are evenly represented, is suitable for classifier construction, as it helps mitigate potential biases.

#### 4.4.4 Frequency Counting

Even when the class frequencies are counted in a distributed manner with a varying number of nodes, the resulting frequencies for a respective class is always the same. Similarly, the cumulative total number of rows for the datasets is also always equal. The class probabilities, as a result, for each class label stays the same for any number of nodes; however, this does not apply to class-specific values' frequencies because the data extracted in CHAPTER 3 are different for a different number of nodes. Such difference causes the bins to form differently during the discretization process, which results in a discrepancy of frequency for class-specific values in a feature.

The data containing frequency-related values exchanged between each node and the central coordinator are sent in a JavaScript Object Notation (JSON) format where the key-value pair is formatted in a dictionary form [43]. Depending on the information being exchanged, the key contains the class label or the unique value from the dataset, and the value contains the corresponding frequency.



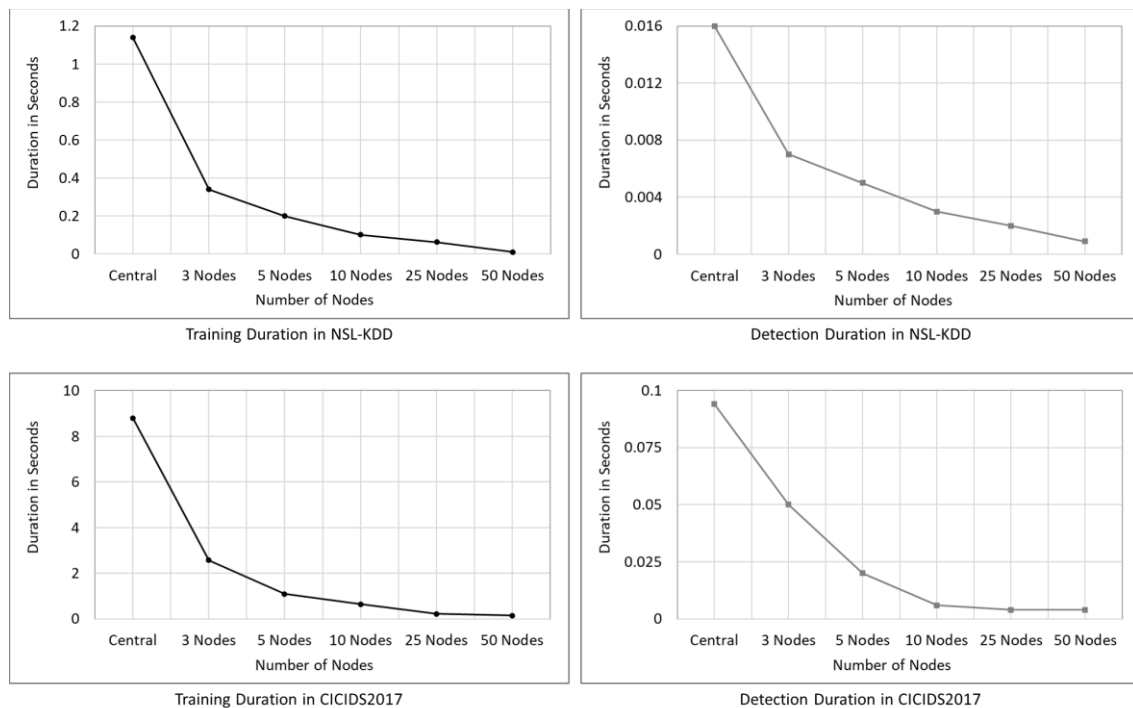
#### 4.4.5 Model Construction

The training phase of model construction constitutes using the counted frequencies to compute the required probabilities. In Naïve Bayes, the model construction only involves the computations of prior probabilities. The posterior probability of an instance belonging to the class can be calculated based on the prior probabilities. The testing phase validates the constructed model with the test set of the data. The results observed during the testing and validation of the constructed models are reported in the following section.

### 4.5 Results and Discussion

#### 4.5.1 Training and Detection Durations Analysis

The training duration is the time taken to construct a classifier, and the detection duration is the time the constructed classifier takes to classify all the instances in the testing set. The training and detection durations decreased as the number of nodes increased. The centralized training took 1.141 seconds and detection took 0.016 seconds for the NSL-KDD dataset. When using 50 nodes for the same dataset, the training and detection durations plummeted to 0.01 and less than 0.001 seconds, respectively. Similarly, for the CICIDS2017 dataset, the centralized training took 8.781 seconds, and detection took 0.094 seconds. With 50 nodes, it only took 0.151 seconds to train and 0.004 seconds to detect. The rates of decrease appear to follow the exponential decay trend, as observed in **Figure 4-3**.



**Figure 4-3:** The comparison between training and detection speeds when using various number of nodes for model construction and intrusion detection. The duration for both training and detection reduced as the number of nodes increased.

It must be noted that these durations do not account for network latencies. The observed shortening of training and detection duration implies that it is possible to distribute an IDS classifier construction and detection jobs across several nodes to boost the speed.

#### 4.5.2 Classification Performance

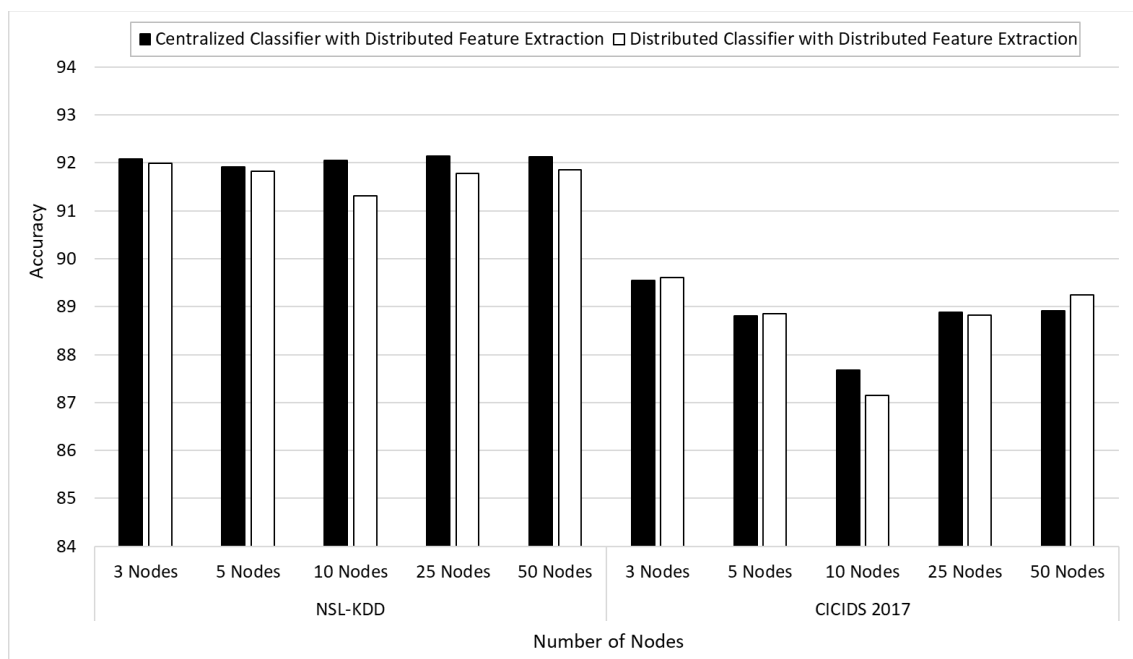
Despite using the varying number of nodes to build the classifiers, the observed performance remained consistent. In the NSL-KDD dataset, all predictors attained over 91% accuracy for any number of tested nodes. In the CICIDS2017 dataset, the performance fluctuated more rapidly, with the lowest accuracy observed being just over 87%, and the highest accuracy observed being close to 90%. The performances of the distributed classifiers are reported in **Table 4-1**.

**Table 4-1:** The performance comparisons between the classifiers constructed in a distributed way using a varying number of nodes.

Dataset	Nodes	Recall	Precision	Specificity	Accuracy
NSL-KDD	3	90.67	<b>94.32</b>	<b>93.56</b>	<b>91.99</b>
	5	91.83	92.41	91.83	91.83
	10	91.80	89.93	90.87	91.31
	25	90.37	94.29	93.46	91.78
	50	<b>94.31</b>	88.36	89.85	91.85
CICIDS2017	3	<b>93.72</b>	86.65	85.45	<b>89.60</b>
	5	89.78	<b>88.11</b>	<b>87.92</b>	88.85
	10	93.18	83.17	81.11	87.15
	25	92.39	86.23	85.28	88.83
	50	90.99	87.93	87.48	89.24

#### 4.5.2.1 *Centralized vs. Distributed Predictor Performance*

The Naïve Bayes-based predictors constructed in a distributed manner are compared against the ones that were constructed centrally in CHAPTER 3 (see **Table 3-7** and **Table 3-8**). Even though the predictors constructed in a distributed manner using the procedure discussed in this chapter appear slightly inferior in terms of accuracy, their observed performance is still impressive. The centrally constructed predictors, which use the features extracted in a distributed manner, have performed only slightly better in most cases than their counterparts that use the features extracted and the classifier constructed distributedly. The models constructed to examine the quality of the features in CHAPTER 3, however, did not involve any feature selection. **Figure 4-4** shows comparisons between centralized and distributed classification models.



**Figure 4-4:** The performance comparison based on accuracy between the predictors constructed in centralized and distributed manners. When 3, 5, and 50 nodes were used for the CICIDS2017 dataset, the distributed classifier performed better than the centralized classifier.

#### 4.5.3 FPR Analysis

The FPR, when using the NSL-KDD dataset, stayed below 10% for any number of nodes; however, in the CICIDS2017 dataset, the FPR was regularly over 12%, with the worst FPR being 18.89% when using ten nodes. In comparison to the predictors constructed centrally in CHAPTER 3 (see **Table 3-9**), the classifiers constructed in a distributed manner have higher FPRs for both datasets, as seen in **Table 4-2**.

**Table 4-2:** The best FPR achieved in each dataset by the distributed Naïve Bayes.

Dataset	Nodes	Test Instances	False Positives	FPR
NSL-KDD	3	29,703	878	6.44
CICIDS2017	5	200,000	12,097	12.08

## 4.6 Conclusions

With the growing volume of data that an IDS must process to detect attacks, the centralized IDSs are becoming outdated for modern-day distributed network infrastructures that facilitate high-volume data exchanges. The attacks are evolving rapidly, so an IDS must adapt continuously to retain its effectiveness. We identified categorical Naïve Bayes as a scalable method that is fast and appropriate for an IDS, as it only requires frequency counting and prior probability computations for model construction. This chapter outlined a procedure to perform Naïve Bayes in a distributed setting, where numerous nodes, with the help of the central coordinator, collaboratively construct a classifier and independently detect attacks.

By constructing and validating the classifiers using multiple nodes, we demonstrated that the durations for constructing classifiers and detecting attacks could be reduced by employing multiple nodes. The rate of decrease in duration closely followed the exponential decay trend when more nodes were added into the network. Similarly, the classifiers retained a similar level of performance-accuracy even when numerous nodes were used for classification model construction, instead of just one. The distributed classifiers constructed with the NSL-KDD dataset consistently attained an accuracy of over 91%; whereas, the ones constructed with the CICIDS2017 dataset attained the accuracy between 87.15% and 89.60%. Such observations show that when the data is spread across several nodes, an effective distributed classifier can be constructed and utilized.

It is apparent that the classifiers constructed and deployed in a distributed manner can handle a larger volume of data in a shorter time. In addition, since each node can

independently detect the attacks, once the model is constructed, such an approach mitigates the issues related to the single point of failure by making the IDS implementation more robust.

## CHAPTER 5

### SIMILARITY MEASURE-BASED LEARNING AND MULTI-MODEL BINARY CLASSIFICATION

#### 5.1 Background

In CHAPTER 3, we used the numerical attributes in the available datasets for feature extraction, and in CHAPTER 4, we utilized those extracted features for collaborative classifier construction. Because the PCA, which works with numerical data, was used for feature extraction, all the existing categorical data present in the datasets had been ignored; therefore, any significance held by them were disregarded. We now introduce a similarity measure-based classification algorithm that utilizes categorical data.

Even though the distance measures are often perceived as applicable only to the points in a 3-dimensional space, most distance measures can compute the distance between multi-dimensional data points that extend beyond the 3-dimensional physical space [44]. Based on the properties a distance measure satisfies, it can be categorized into metric distance measure or semi-metric distance measure. For points  $A$ ,  $B$ , and  $C$ , a metric distance measure meets the following properties.

- a) The distance between  $A$  and  $B$  is greater than or equal to 0.
- b) The distance between  $A$  and  $B$  is 0, if and only if  $A = B$ .
- c) The distance between  $A$  and  $B$  is equal to the distance between  $B$  and  $A$ .

- d) The distance between  $A$  and  $B$  is less than or equals to the sum of distances between those points and some other point; i.e.,

$$\text{distance}(A, B) \leq \text{distance}(A, C) + \text{distance}(C, B).$$

A semi-metric distance measure, on the other hand, satisfies only the first three of these properties [45]. The dissimilarity and similarity between points are typically related, such that the degree of similarity between two points,  $A$  and  $B$ , can be expressed as

$$\text{Similarity}(A, B) = 1 - \text{Dissimilarity}(A, B), \quad \text{Eq. 5-1}$$

where  $\text{Dissimilarity}(A, B)$  is the degree of dissimilarity computed by a distance measure, and  $\text{Dissimilarity}(A, B) = \text{Distance}(A, B)$ . Some similarity measures do not require the computation of the degree of dissimilarity and work with categorical data. Cosine similarity, which has widespread applications, is one of them. It finds the cosine angle between the vectors — a smaller angle implies more similarity [46].

For two non-zero vectors,  $A$  and  $B$ , the cosine similarity is computed as their dot product and magnitudes, which is given by

$$\text{Similarity}(A, B) = \text{Cos}(\theta) = \frac{A \cdot B}{\|A\| \times \|B\|}. \quad \text{Eq. 5-2}$$

As seen in **Eq. 5-2**, the similarity between  $A$  and  $B$  is based on the ratio between their dot product and the product of their L2-norms. The resulting value ranges from  $-1$  to  $1$ . When the value is  $-1$ , then the two vectors are the opposite; when the value is  $0$ , then they are orthogonal; and, when the value is  $1$ , the cosine angle is the least, and the vectors are precisely the same.

The presented similarity measure-based classification technique determines the frequency-based centroid of the data by averaging the frequencies of all unique values in each feature. It uses the cosine similarity-based method to find the degree of similarity



between the class-specific value weights vector of a newly observed instance and the determined centroid of the data. The computed degree of similarity is then used to perform a supervised classification.

As an extension, the computed similarities are utilized along with the probabilities yielded by a Naïve Bayes-based classification model constructed using a technique discussed in CHAPTER 4 to form the inputs for the third classifier. The purpose of such multi-model approach is to improve the overall accuracy of the classification.

## 5.2 Related Works

There have been many studies on distance measures in terms of their applicability to the IDS. Since the behavioral-based IDS may use a classification or clustering technique to build a model, the distance measures that are applicable to these are of interest. A survey of distance and similarity measures used in the network anomaly detection problem domain is conducted in [44]. An overview of the distance-based classifications is given in [47].

Ahmed et al. [48] propose a dissimilarity metric based on Minimum Spanning Tree (MST). This metric is used to isolate the abnormal clusters and individual data points by using a two-step process where the MSTs are first built at the global level and then at the local level. Out of the compared methods, the authors show that their proposed method performs better in most cases.

A new metric distance measure for categorical values is proposed in [49] for unsupervised learning. This metric considers the frequency probability of each attribute in the entire dataset to compute the distance between two categorical data. Additionally,

to ensure that the distance metric treats each attribute differently, based on their importance, a dynamic attribute weight is assigned to them.

Kruegel et al. [50] present an anomaly-based IDS that utilizes a multi-model process to detect anomalous traffic to defend web servers and web-based applications. Their method employs multiple models that analyze the queries used to pass the parameters to invoke the server-side programs. Each model assigns a probability value based on their observation. The detection relies on those values. The authors claim that when the models outputted Bayesian technique-based probability values, they produced favorable results.

This chapter discusses a procedure for conducting a similarity measure-based supervised classification, which is different than what is found in the recent literature because it deals with structured categorical data using a directed technique. Additionally, it is ensembled with the probabilistic technique discussed in CHAPTER 4 to produce relevant outputs for a tertiary classifier that is used to achieve a higher accuracy through a multi-model classification approach.

## 5.3 Methodology

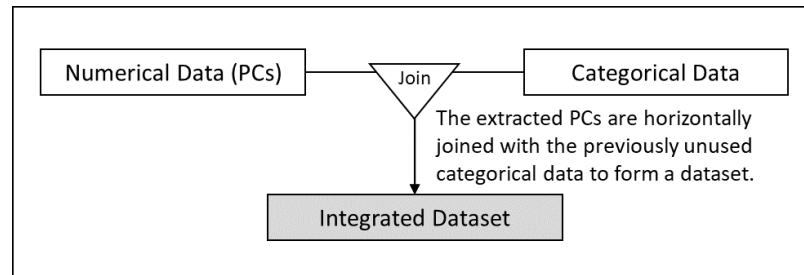
### 5.3.1 Data Selection and Integration

The categorical data available in the original datasets had been ignored in the previous chapters, and only the numerical and discretized-numerical data were used. The previously unused attributes containing the categorical values from both NSL-KDD and CICIDS2017 datasets are shown in **Table 5-1**.

**Table 5-1:** The categorical attributes in the NSL-KDD and CICIDS2017 datasets utilized for the similarity measure-based classification. These attributes were ignored in the previous chapters.

Dataset	Categorical Attributes
NSL-KDD	protocol · service · flag · land · logged_in · root_shell · su_attempted · is_hot_login · is_guest_login
CICIDS2017	Destination Port · FIN Flag Count · SYN Flag Count · RST Flag Count · PSH Flag Count · ACK Flag Count · URG Flag Count · CWE Flag Count · ECE Flag Count

The utilized data for the proposed classification method is inclusive of the original categorical attributes. The categorical attributes are integrated with the principal components, which were centrally extracted in CHAPTER 3, to form a dataset containing both categorical and numerical data. **Figure 5-1** depicts the data integration process.



**Figure 5-1:** Data integration performed to combine the previously unused categorical data and the numerical PCs extracted in CHAPTER 3.

The dataset formed through integration also includes the class labels. Such integrations are undertaken for both NSL-KDD and CICIDS2017 datasets.

### 5.3.2 Data Transformation

The numerical data are normalized and discretized to ensure their suitability for the discussed supervised classification method.

### 5.3.2.1 Normalization

It is essential to normalize the numerical data present in the datasets because of the reasons explained in section 2.2.1. The normalization is done using the z-score normalization technique, as described in section 3.3.4.1. The distributed technique of normalization can be used even in a centralized environment by assuming the presence of only one node.

### 5.3.2.2 Discretization by Binning

It is necessary to discretize the numerical values into categorical values. The equal-width binning method is used for this with the bin size of 1,000 for each dataset. The discretization follows the process undertaken in CHAPTER 4. See section 4.3.1.2.

### 5.3.3 Features Analysis, Ranking, and Selection

All available attributes are analyzed to identify the ones that would be most useful to build a classification model. The analysis is done by testing the independence between two features. The idea is to identify the features that are independent of one another but are dependent on the class label. The features are analyzed using the chi-square statistic-based method, which has been described in section 4.3.2. The features are ranked based on their determined importance, after analysis. The top  $n$  features identified as important are selected to build the classification models.

### 5.3.4 Similarity Measure-based Classification (SMC)

This subsection formally introduces SMC. First, the frequency of each unique value in the features are counted. The determined value-specific frequencies for each feature in the entire dataset are averaged to identify the data centroid. After determining the centroid, the instances are separated by class labels. The selected features are

analyzed to find the frequency of the values on those features for the respective class. Each unique value is assigned a class-specific frequential weight. For a newly-observed instance, the similarity between the class-specific weights for its values and the data centroid is calculated. Such measurement is repeated for each class. An observed instance is then classified into the class whose weights vector for the values in the instance shares the highest similarity with the centroid.

It is possible to perform SMC in both centralized and decentralized environments. Suppose a dataset,  $X$ , has a finite number of categorical attributes and a target variable,  $C$ , containing class labels. Let  $F = \{f_i : 1 \leq i \leq n\}$  be the set of selected  $n$  features and  $C = \{c_i : 1 \leq i \leq p\}$  be the set of  $p$  unique class labels. The data is separated based on the class label, such that:  $X = \{X_{c_1}, X_{c_2}, \dots, X_{c_p}\}$ . The paired list of all the selected  $n$  features and the corresponding unique values they hold that are associated with a specific class,  $c$ , can be represented as

$$X_c = \left\{ \left( f_{i_c}, \left\{ v_{j_{f_{i_c}}} : 1 \leq j \leq k_{f_{i_c}} \right\} \right) : 1 \leq i \leq n \right\}, \quad \text{Eq. 5-3}$$

where  $n$  is the number of features and  $k_{f_{i_c}}$  is the number of unique values in feature,  $f_{i_c}$ , for the class,  $c$ .

#### 5.3.4.1 Frequency Analysis

The frequency analysis of the values in the entire dataset and in each specific class are performed to determine the required frequencies. Let  $f$  represent the frequency in the following sections.

#### 5.3.4.1.1 Value-Frequency in Dataset

The number of occurrences is counted for each unique value in a feature. Such counting is done without any regard to the class labels. If a dataset has  $N$  number of rows, then for a unique value,  $v_f$ , in a feature,  $f$ , the frequency can be determined using

$$\#_{v_f} = \sum_{i=1}^N [v_f = v_{i_f}], \quad \text{Eq. 5-4}$$

where the frequency,  $\#_{v_f}$ , is computed by comparing every unique value,  $v_f$ , against all the other values,  $v_{i_f}$ , in a feature,  $f$ . The frequency is incremented by 1 whenever a match is found, as denoted by the Iverson bracket in **Eq. 5-4**. The frequencies so computed for every possible value in each feature are stored. The purpose of determining these frequencies is to compute the frequency-based centroid of the training set.

#### 5.3.4.1.2 Value-Frequency in Class

The portion of data associated with a class is analyzed to determine each value's class-specific frequency in a feature. The frequency analysis is done by counting each occurrence of a particular value in a feature for a class,  $c$ . This frequency counting process can be expressed similar to **Eq. 5-4** as

$$\#_{v_{f_c}} = \sum_{i=1}^{N_c} [v_{f_c} = v_{i_{f_c}}], \quad \text{Eq. 5-5}$$

where the frequency,  $\#_{v_{f_c}}$ , is determined by comparing each unique value  $v_{f_c}$  against other values,  $v_{i_{f_c}}$ , in a feature,  $f$ , for a class,  $c$ . For each match, the frequency is incremented. Such frequency analysis involves all  $N_c$  instances associated with a class. The resulting frequency distribution for every possible value in all selected features for a

specific class is stored. The values are assigned a weight based on their class-specific frequency.

#### 5.3.4.2 Frequency-based Data Centroid

The centroid is identified by averaging all the frequencies for the values in a feature. These frequencies are computed using the process explained in section 5.3.4.1.1. For a dataset with  $n$  number of selected features, the data centroid,  $m$ , based on the value frequencies can be determined using

$$\text{Centroid } (m) = \left\{ \frac{1}{k_{f_i}} \sum_{j=1}^{k_{f_i}} f_{v_j f_i} : 1 \leq i \leq n \right\}, \quad \text{Eq. 5-6}$$

where  $k_{f_i}$  is the number of unique values in feature,  $f_i$ , and  $f_{v_j f_i}$  is the frequency for the  $j^{\text{th}}$  value in feature,  $f_i$ . The centroid is formed by the averages of the frequencies for each value in the selected features. The computed centroid is used to determine the similarity between an observed new instance and the class.

#### 5.3.4.3 Frequential-Weight Determination

The weight for each value based on its frequency is computed relative to the number of rows in a dataset containing only class  $c$  and the total number of rows in the entire dataset. The expression to compute the weight,  $w_{v_{f_c}}$ , for each value is given by

$$w_{v_{f_c}} = \frac{f_{v_{f_c}} \times N_c}{N}, \quad \text{Eq. 5-7}$$

where  $f_{v_{f_c}}$  is the frequency of a value,  $v$ , in a feature,  $f$ , for a class,  $c$ ,  $N_c$  is the number of instances representing the class  $c$  and  $N$  is the total number of instances in the dataset.

After the class-specific weight for each value is determined and stored, the weight for any existing value can be extracted using the mapping function,  $W$ , which gives the weight,  $w$ , associated with a value,  $v$ .  $W$  can be defined as

$$W(x) = \begin{cases} w, & \text{for } x = v \\ W(v), & \text{otherwise} \end{cases} \quad \text{Eq. 5-8}$$

If an associative array with a value-weight pair for all the possible values is maintained, then for a value  $v_{fc}$  associated with the class  $c$  in a feature  $f$ , its weight  $w_{v_{fc}}$  can be determined by using the mapping function represented by

$$W : v_{fc} \rightarrow w_{v_{fc}}. \quad \text{Eq. 5-9}$$

#### 5.3.4.4 Similarity Measurement

The degree of similarity is measured between the class-specific weights vector representing a newly observed instance,  $e$ , and the centroid of the data. First, the values for each feature from an observed instance are extracted. Then, by using the function,  $W$ , the class-specific frequency-based weights for the values are determined, and these weights are used to form a vector. For  $n$  features, suppose  $w_{e_c} = \langle w_{1e_c}, w_{2e_c}, \dots, w_{ne_c} \rangle$  is a vector defined by the weights for the values in an observed instance for a class,  $c$ , and  $m = \langle m_1, m_2, \dots, m_n \rangle$  is the vector defined by the values describing a centroid. There can be situations where some values in a feature are present only for certain classes. In those situations, the weight of 0 is assigned to such values for a class. The components in  $w_{e_c}$  and  $m$  are in the same order, with each component in  $w_{e_c}$  representing the class-specific weight for a value in a feature and the corresponding component in  $m$  representing the feature-specific average of the global value-frequencies. When there are  $n$  features, the cosine similarity between  $w_{e_c}$  and  $m$  can be measured using **Eq. 5-2** as



$$\text{Similarity}(w_{e_c}, m) = \frac{\sum_{i=1}^n (w_{i_{e_c}} \times m_i)}{\sqrt{\sum_{i=1}^n w_{i_{e_c}}^2} \sqrt{\sum_{i=1}^n m_i^2}}. \quad \text{Eq. 5-10}$$

Since each component of  $w_{e_c}$  has a value that is 0 or higher and  $m$  has all positive components, the resulting degree of similarity ranges between 0 and 1, with 0 implying that the vectors are entirely dissimilar and 1 implying that they are the same.

#### 5.3.4.5 Classification

The classification is done by comparing the degree of similarity between the class-specific frequential-weights for an observed instance and the centroid of data. If there are  $p$  classes, then let  $w_e = \{w_{e_{c_i}} : 1 \leq i \leq p\}$  be the set containing the compiled sets of value-weights in an instance for each of the classes. Then, the computed similarities between the elements in  $w_e$  and the centroid,  $m$ , can be represented as

$$S_{w_e} = \{S(w_{e_{c_i}}, m) : 1 \leq i \leq p\}. \quad \text{Eq. 5-11}$$

The highest degree of similarity is the most significant value in  $S_{w_e}$ , such that

$$S_{max} = \max [S_{w_e}]. \quad \text{Eq. 5-12}$$

An observed instance is then classified into the class that it shares the most similarity with, as represented by the following mapping function,  $\mathcal{C}$ .

$$\mathcal{C} : S_{max} \rightarrow c \quad \text{Eq. 5-13}$$

#### 5.3.5 Distributed SMC

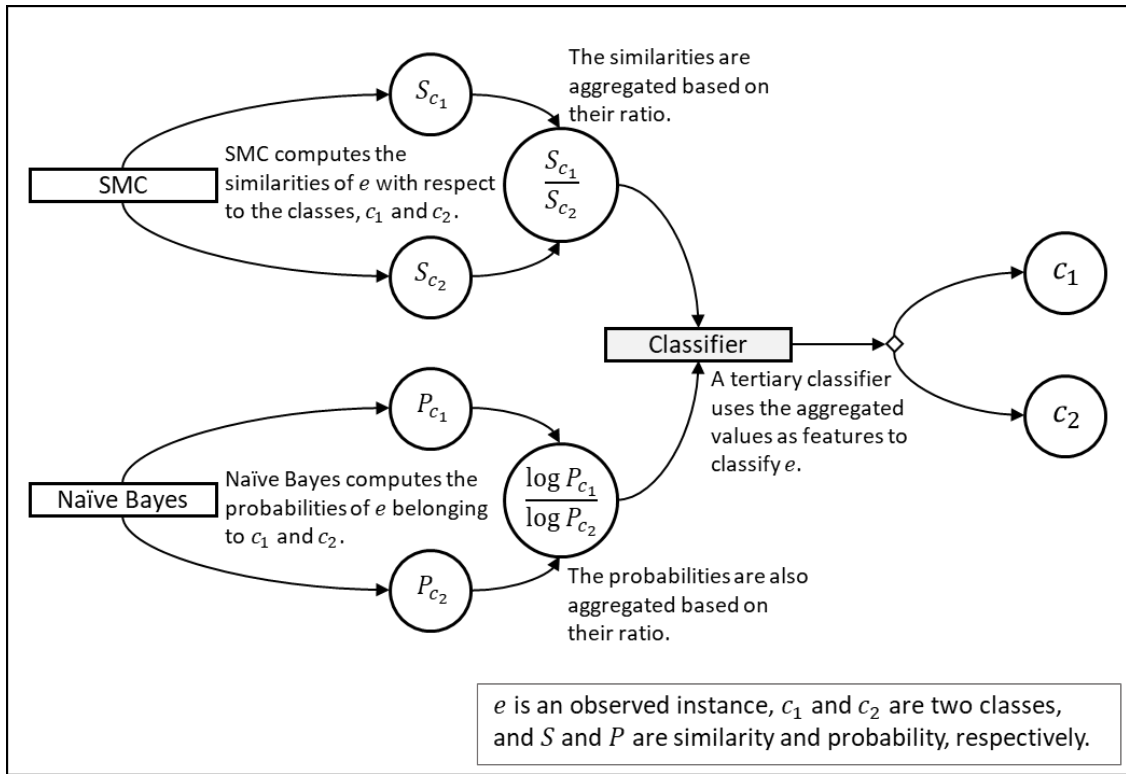
The SMC has been performed centrally for experiments; however, it is possible to perform SMC in a distributed environment. This subsection outlines how this can be done. When performing SMC in a distributed setting, the frequency of each value must be counted in a distributed manner. The number of instances associated with every class

also must be counted collaboratively. These processes closely follow the procedures explained in sections 4.3.4.1, 4.3.4.2, and 4.3.4.3. Each node, first, segregates the data by class and counts the number of instances it has for every class. Then, it computes the value's local frequency in a feature for a class. The frequencies of the values in the entire dataset can be calculated similarly but without any regard for the class. The nodes then share the local frequencies with the central coordinator. The central coordinator collects all the local frequencies and determines the global frequencies. The determined global frequencies are shared with each node. Each node now knows the total number of rows in a dataset, the number of rows associated with each class, the class-specific value frequencies in every feature, and the total frequency of each value in a dataset.

With the known information, the nodes can independently compute the data centroid and value-weights using the processes explained in sections 5.3.4.2 and 5.3.4.3, respectively. A new instance observed by any node is classified using the processes described in sections 5.3.4.4 and 5.3.4.5.

### 5.3.6 Multi-Model Binary Classification

SMC, which depends on similarity measurement, can be used alongside other classifiers to perform multi-model classifications. Naïve Bayes, as discussed in CHAPTER 4, depends on Bayes' theorem. Since both of these techniques heavily rely on frequency counts, it is logical to combine the information outputted by them to form a multi-model classifier to improve the overall accuracy of classification. The classification to be conducted is binary, so let  $c_1$  and  $c_2$  be the two class labels. **Figure 5-2** shows a high-level overview of the multi-model classification process.



**Figure 5-2:** A high-level illustration of a multi-model classifier for binary classification. SMC, Naïve Bayes, and a tertiary classification model collaborate to make decisions.

The multi-model classification makes use of the same training set used for the SMC and Naïve Bayes classifiers. After constructing SMC and Naïve Bayes-based models, each instance,  $e$ , in the training set is passed through those models to get the similarities  $S(w_{e_{c_1}}, m)$  and  $S(w_{e_{c_2}}, m)$  and posterior probabilities  $P(e | c_1)$  and  $P(e | c_2)$ , respectively. These similarities and probabilities are used to compute two new values — similarity ratio and probability ratio. These ratios become new features. The similarity ratio for an arbitrary instance is given by

$$\text{Similarity Ratio } (\Phi_{S_e}) = 1 - \frac{S(w_{e_{c_1}}, m)}{S(w_{e_{c_2}}, m)}, \quad \text{Eq. 5-14}$$

where a similarity computed for one class is divided by another similarity to aggregate those into a single value indicating the ratio between the two similarities.

Similarly, the probability ratio for the same instance is given by

$$\text{Probability Ratio } (\Phi_{P_e}) = 1 - \frac{\log P(e | c_1)}{\log P(e | c_2)}, \quad \text{Eq. 5-15}$$

where the log-probability of an instance belonging to one class is divided by the log-probability of an instance belonging to another class. The log-function is used to represent the values in a logarithmic scale, rather than in the usual  $[0, 1]$  probability scale.

With the computed similarity and probability ratios, the new training dataset is formed that includes the similarity ratio, probability ratio, and class label for each of the instances from the original training set. This new training set is used to construct a classifier of choice. After the classifier is constructed using a newly formed training set, the model is validated.

For each test instance, the similarities and probabilities must be computed before it can be used. The computed similarities and probabilities are utilized to get the similarity ratio and probability ratio using **Eq. 5-14** and **Eq. 5-15**. These ratios are passed as inputs to the constructed tertiary classifier to get the desired output.

Depending on the nature of implementation, two variants of the multi-model approach have been proposed. Both variants use a similar technique. The difference is in whether the tertiary classifier is only partially involved or fully involved.

#### 5.3.6.1 Partially-Dependent Multi-Model (PDMM)

SMC and Naïve Bayes are mutually used to perform an initial classification in this approach. If SMC and Naïve Bayes-based models concur and classify an instance into the same class, then that is regarded as the final decision; otherwise, the respective

similarities and probabilities they compute are aggregated to form similarity and probability ratios separately. These ratios become inputs for the tertiary classifier, which then performs the final classification.

#### 5.3.6.2 Fully-Dependent Multi-Model (FDMM)

SMC and Naïve Bayes do not make any classification in this approach. Instead, they output the determined similarities and probabilities, to enable the computations of the similarity and probability ratios. These ratios are supplied to the third model as inputs for classification. The tertiary classifier does all the classifications; hence, the classification model entirely depends on it for the final decision-making. If the used third classifier has a high complexity, then this process can become expensive.

### **5.4 Experimental Procedure & Observations**

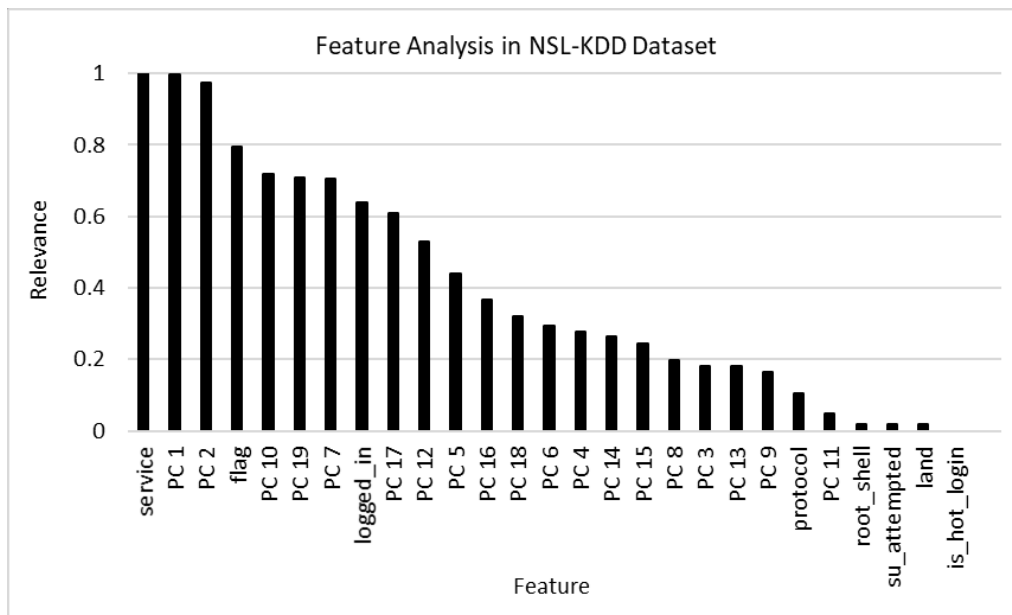
#### 5.4.1 Preparation

The unnecessary and redundant columns are eliminated from the datasets. The rows containing invalid and incorrect data are either corrected or removed. The centrally extracted PCs in CHAPTER 3 are horizontally joined with the corresponding rows of the previously unused categorical columns to form new integrated datasets. The integrated datasets are then split in a 4:1 ratio to form the training and testing sets. The feature selection is undertaken on training sets to identify the relevant features for classification.

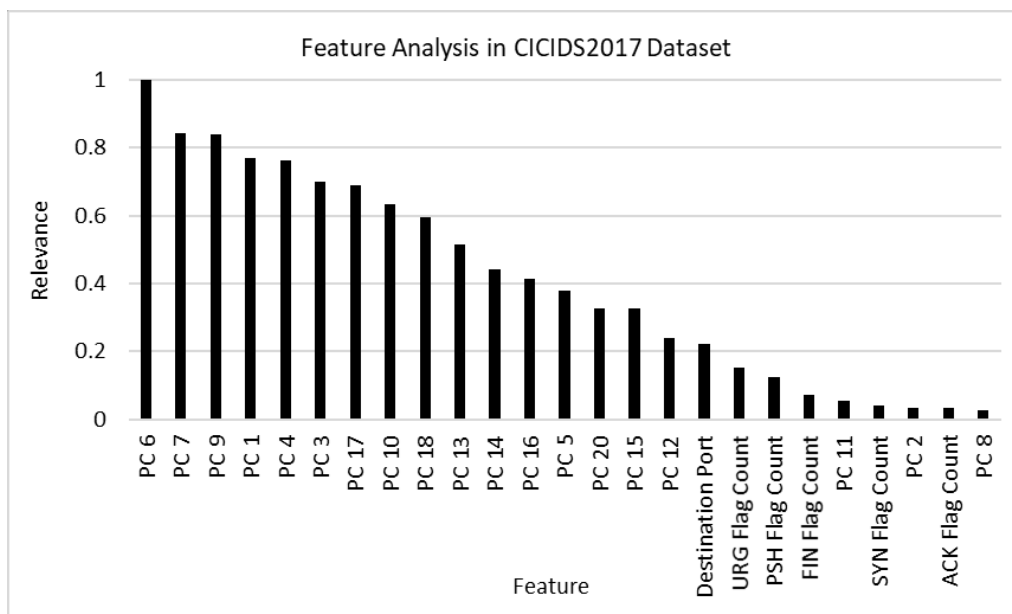
#### 5.4.2 Features Selection

The datasets containing categorical and discretized numerical data are analyzed using the chi-square statistic technique to identify the best features to build a classifier. The SMC-based classifier has been constructed and tested using a different number of top

features. **Figure 5-3** and **Figure 5-4** show the relevance of each analyzed feature determined by the chi-square statistic-based test of independence.



**Figure 5-3:** The relevancy of features in the NSL-KDD-based integrated dataset. *service* is identified as the most relevant feature, followed by *PC 1* and *PC 2*.



**Figure 5-4:** The relevancy of features in the CICIDS2017-based integrated dataset. None of the original categorical features were among the 16 most relevant features.

In order to remain consistent with the procedure followed in CHAPTER 4, the top 15 features are used for the Naïve Bayes-based classifier.

### 5.4.3 SMC-based Model Construction

The SMC-based model construction involves a four-step process — counting the number of rows representing each class, determining the frequency of each unique value in a feature for a class, identifying the centroid of the data by averaging the feature-specific frequencies of the values, and calculating and assigning class-specific frequency-based weights to the values. These steps are undertaken in the training dataset. Once the data centroid and class-specific frequential weights for each unique value are known, a new observed instance is classified by measuring the degrees of similarity between the class-specific vectors containing the weights of its values and the centroid.

#### 5.4.3.1 Class Frequencies

In the training sets, the class frequencies are determined by counting the number of rows representing a specific class. **Table 5-2** shows the observed class frequencies in each dataset.

**Table 5-2:** The number of rows representing each class in the training sets.

<b>Dataset</b>	<b>Class</b>	<b>Frequency</b>
NSL-KDD	<i>Attack</i>	57,227
	<i>Normal</i>	61,587
CICIDS2017	<i>Attack</i>	400,153
	<i>Normal</i>	399,847

### 5.4.3.2 Frequential Value-Weight Determination

In the training set of the NSL-KDD dataset, the number of total rows representing *Attack* is 57227. The feature `service` in this part has 65 distinct values. The value `private` has the highest frequency, 19894; hence, its weight can be computed using **Eq. 5-7** as  $\frac{19894 \times 57227}{118814} = 9581.98$ . The value `eco_i` has the second-highest frequency, 3466, so its weight is  $\frac{3466 \times 57227}{118814} = 1669.41$ . In the dataset's portion representing the *Normal* class, the total number of rows is 61587. The feature `service` has 28 distinct values, with `http` being the most frequent one with the frequency of 35726, so its weight is  $\frac{35726 \times 61587}{118814} = 18518.50$ . The value `domain_u` is the second most frequent one with the frequency of 7869; therefore, its weight is  $\frac{7869 \times 61587}{118814} = 4078.88$ . The weights for all the unique values in features associated with each class labels are calculated similarly.

The computed weights are assigned to their respective values, and the value-weight pairs are stored in a dictionary form so they can be quickly retrieved whenever needed. The value-weight pairs are available for each unique class. The three largest computed value-weights in the NSL-KDD dataset for each feature in the *Attack* and *Normal* classes are tabulated in **Table 5-3** and **Table 5-4**, respectively.



**Table 5-3:** The computed three most significant weights for the values in the top 16 features for the *Attack* class in the NSL-KDD dataset.

<u>service</u>		<u>PC 1</u>		<u>PC 2</u>		<u>flag</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
private	9581.98	bin_891	269.73	bin_83	888.17	S0	14080.61
eco_i	1669.41	bin_895	250.94	bin_88	866.01	SF	6484.96
ecr_i	1370.30	bin_896	249.98	bin_82	847.71	REJ	4759.69
<u>PC 10</u>		<u>PC 19</u>		<u>PC 7</u>		<u>logged in</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_862	9273.25	bin_862	9273.25	bin_653	1159.34	0	25779.94
bin_863	5048.68	bin_863	5048.68	bin_652	1097.68	1	1783.56
bin_869	2426.08	bin_654	1087.57	bin_642	1701.68	-	-
<u>PC 17</u>		<u>PC 12</u>		<u>PC 5</u>		<u>PC 16</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_561	4561.24	bin_760	10959.51	bin_568	300.55	bin_689	17823.05
bin_560	4435.05	bin_759	8763.66	bin_567	283.69	bin_688	7415.03
bin_562	3696.20	bin_758	1512.39	bin_538	278.39	bin_690	1471.93
<u>PC 18</u>		<u>PC 6</u>		<u>PC 4</u>		<u>PC 14</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_449	695.51	bin_419	3989.52	bin_175	1199.31	bin_407	12827.36
bin_451	650.71	bin_417	3724.61	bin_181	1196.91	bin_408	8491.52
bin_448	639.63	bin_418	2934.22	bin_180	1157.89	bin_405	2533.01

**Table 5-4:** The computed three most significant weights for the values in the top 16 features for the *Normal* class in the NSL-KDD dataset.

<u>service</u>		<u>PC 1</u>		<u>PC 2</u>		<u>flag</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
http	18518.50	bin_265	1089.05	bin_157	3360.97	SF	30258.04
domain_u	4078.88	bin_264	899.85	bin_151	2554.42	REJ	1108.23
smtp	3169.70	bin_266	818.47	bin_152	1925.66	S1	160.69
<u>PC 10</u>		<u>PC 19</u>		<u>PC 7</u>		<u>logged in</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_859	15081.85	bin_859	15081.85	bin_584	1395.91	1	22919.28
bin_860	6052.75	bin_860	6052.75	bin_585	908.15	0	9004.22
bin_861	1839.10	bin_589	820.54	bin_642	1405.24	-	-
<u>PC 17</u>		<u>PC 12</u>		<u>PC 5</u>		<u>PC 16</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_563	2065.62	bin_758	18769.38	bin_601	1037.21	bin_688	24091.77
bin_564	1811.63	bin_759	7174.45	bin_596	944.95	bin_689	6414.04
bin_555	850.09	bin_760	2945.77	bin_599	931.47	bin_690	477.40
<u>PC 18</u>		<u>PC 6</u>		<u>PC 4</u>		<u>PC 14</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_433	1309.35	bin_419	4219.35	bin_208	1230.56	bin_408	22757.55
bin_434	1040.32	bin_423	3568.31	bin_207	950.65	bin_407	6661.29
bin_435	885.34	bin_420	3355.27	bin_209	940.28	bin_406	955.32

The weights are computed and recorded similarly for the CICIDS2017 dataset. The three largest computed weights for the values in the *Attack* and *Normal* classes of this dataset are tabulated in **Table 5-5** and **Table 5-6**, respectively.

**Table 5-5:** The computed three most significant weights for the values in the top 16 features for the *Attack* class in the CICIDS2017 dataset.

<u>PC 6</u>		<u>PC 7</u>		<u>PC 9</u>		<u>PC 1</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_863	25765.35	bin_527	18995.26	bin_504	29382.73	bin_92	25608.79
bin_864	24441.85	bin_526	18128.93	bin_508	22875.75	bin_93	24849.50
bin_866	16632.86	bin_527	18995.26	bin_505	12036.60	bin_95	10264.42
<u>PC 4</u>		<u>PC 3</u>		<u>PC 17</u>		<u>PC 10</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_545	73013.42	bin_303	26866.27	bin_492	42002.06	bin_465	28295.32
bin_544	65164.92	bin_300	26189.01	bin_488	25477.74	bin_468	24518.37
bin_549	17135.05	bin_306	9213.02	bin_491	22241.50	bin_466	20310.77
<u>PC 18</u>		<u>PC 13</u>		<u>PC 14</u>		<u>PC 16</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_498	46450.76	bin_709	57264.40	bin_524	37236.74	bin_667	25357.20
bin_494	31116.90	bin_711	40850.12	bin_525	21402.18	bin_673	25335.19
bin_501	28998.09	bin_712	25734.34	bin_523	18926.24	bin_676	20836.97
<u>PC 5</u>		<u>PC 20</u>		<u>PC 15</u>		<u>PC 12</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_866	101664.37	bin_280	27977.70	bin_507	31084.39	bin_104	54685.41
bin_869	19749.05	bin_279	22325.04	bin_511	28460.38	bin_105	52286.49
bin_865	17738.28	bin_283	20417.81	bin_508	24663.43	bin_106	35927.24

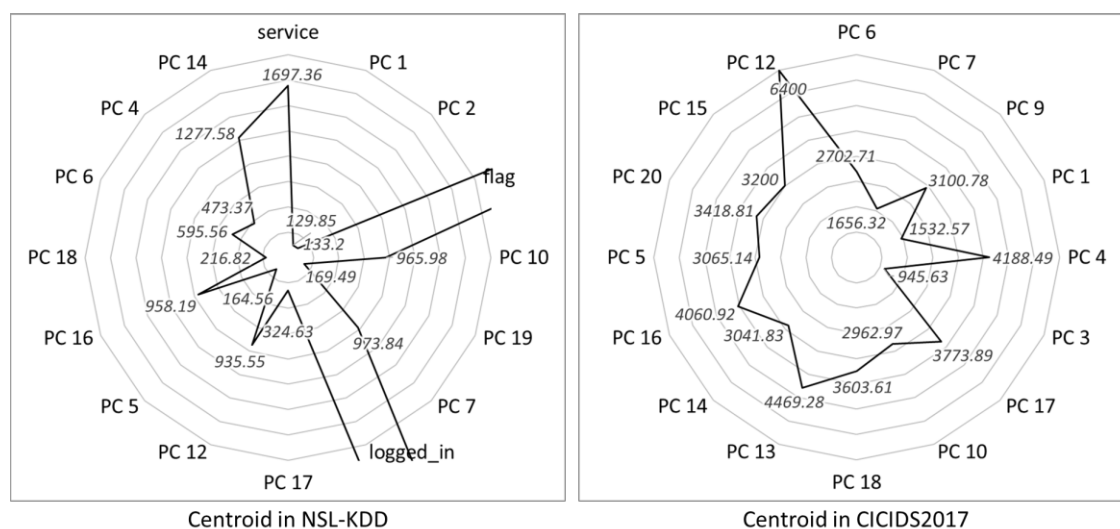
**Table 5-6:** The computed three most significant weights for the values in the top 16 features for the *Normal* class in the CICIDS2017 dataset.

<u>PC 6</u>		<u>PC 7</u>		<u>PC 9</u>		<u>PC 1</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_865	12092.87	bin_508	7106.78	bin_496	16007.37	bin_91	41680.05
bin_853	11854.96	bin_509	6648.96	bin_497	15471.08	bin_92	34692.72
bin_866	11808.98	bin_529	6518.01	bin_504	14732.36	bin_93	13487.84
<u>PC 4</u>		<u>PC 3</u>		<u>PC 17</u>		<u>PC 10</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_544	96214.68	bin_305	30700.75	bin_491	70188.64	bin_456	12033.40
bin_543	34428.33	bin_304	27822.35	bin_492	35276.00	bin_467	11525.09
bin_545	18996.73	bin_306	27523.47	bin_490	18103.57	bin_454	11514.09
<u>PC 18</u>		<u>PC 13</u>		<u>PC 14</u>		<u>PC 16</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_499	46541.69	bin_711	49509.56	bin_526	34423.33	bin_675	20196.27
bin_498	32680.49	bin_712	43320.42	bin_527	26638.81	bin_676	19745.94
bin_500	24650.57	bin_713	22907.73	bin_523	21278.86	bin_673	17663.74
<u>PC 5</u>		<u>PC 20</u>		<u>PC 15</u>		<u>PC 12</u>	
<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>	<b>Value</b>	<b>Weight</b>
bin_869	40749.91	bin_279	16419.22	bin_506	27011.16	bin_105	78080.62
bin_866	37413.68	bin_282	16171.31	bin_505	24821.50	bin_104	60444.87
bin_870	27307.05	bin_280	14546.43	bin_504	24700.05	bin_103	13448.85

#### 5.4.3.3 *Data Centroids*

The data centroid based on the value-frequencies in its features is determined for each dataset using **Eq. 5-6**. The determined centroid is stored for the future degree of

similarity calculations. The centroids observed for each dataset are plotted in **Figure 5-5**. The averages of the frequencies computed for the values in `flag` and `logged_in` features of the NSL-KDD dataset's training set transcend the boundaries of the plotted radar; therefore, they are not visible.



**Figure 5-5:** The average value-frequency-based centroids in the NSL-KDD and CICIDS2017 datasets. Each point represents the average value-frequency in the respective feature.

#### 5.4.4 Similarity Measurements

The similarity of a newly observed instance is measured from each of the classes. For the illustrated similarity measurement, only the top four features are used. **Table 5-7** shows the values extracted from the top four features of a sample instance that is randomly selected from the test set of the NSL-KDD dataset.

**Table 5-7:** A sample instance from the testing set of the NSL-KDD dataset with the observed values for features `service`, `PC 1`, `PC 2`, and `flag`.

<u>service</u>	<u>PC 1</u>	<u>PC 2</u>	<u>flag</u>
http	bin_203	bin_150	SF

The observed values in the selected four features are used to extract the weights for those values representing each of the classes. For the *Attack* class, the weights for the values {service: “http”, PC 1: “bin\_203”, PC 2: “bin\_150”, flag: “SF”} are represented by the vector,  $w_{Attack} = \langle 1338.51, 7.71, 2.41, 6484.96 \rangle$ . Similarly, for the *Normal* class, the weights for the same values are represented by the vector,  $w_{Normal} = \langle 18518.50, 256.06, 1055.87, 30258.05 \rangle$ .

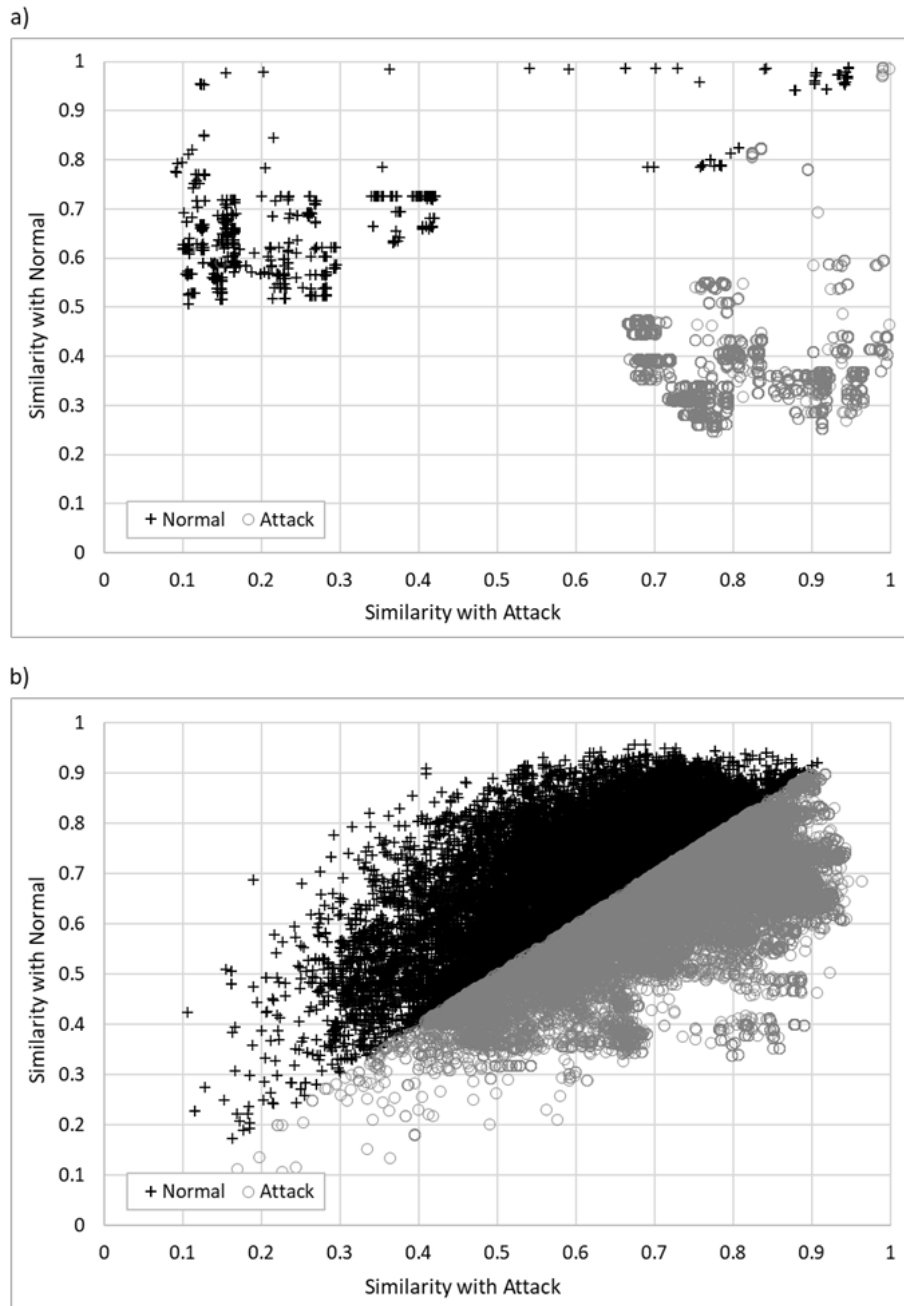
The centroid based on the value-frequencies in the NSL-KDD dataset is represented by  $m = \langle 1697.36, 129.85, 133.20, 9901.25 \rangle$ . Based on these, the similarities  $S(w_{Attack}, m)$  and  $S(w_{Normal}, m)$  can be computed using **Eq. 5-10**. The following are the computed similarities:

$$S(w_{Attack}, m) = 0.9992 \text{ and } S(w_{Normal}, m) = 0.9288.$$

The similarity measurements using other instances and datasets can be done by following the same process.

#### 5.4.5 SMC-based Classification

Since  $S(w_{Attack}, m) > S(w_{Normal}, m)$  in an example above, the instance being investigated is classified into the *Attack* class. The classification using SMC-based classifier distinguishes the point formed by the similarity measures based on its closeness to an axis. The points closer to the y-axis, as seen in **Figure 5-6** for each dataset, are classified as *Normal*; otherwise, they are classified as *Attack*.



**Figure 5-6:** The plots showing the classifications in the a) NSL-KDD and b) CICIDS2017 datasets when using 12 features. The points closer to the x-axis are classified as *Attack*, and the ones closer to the y-axis are classified as *Normal*.

The SMC-based classifiers are constructed using a varying number of features to identify the number that gives the best result. The identified number of features is used for multi-model classification.

#### 5.4.6 Multi-Model Classification

The multi-model classification depends on the values computed by the SMC and Naïve Bayes-based models; therefore, those models are first constructed. The similarity and probability ratios obtained from SMC and Naïve Bayes, respectively, are used as features to construct the third classifier. Both PDMM and FDMM variants of the multi-model approaches have been implemented and tested. k-NN is chosen to form the tertiary classifier due to its known ability to perform well with a limited number of features. It also performed the best when used in CHAPTER 3 to examine the quality of the extracted features. It has now been reused to check the applicability of the PDMM and FDMM methods. For k-NN,  $k = 3$  is used.

In order to build and test both single-model and multi-model classifiers, the combinations of the classifiers listed in **Table 5-8** are constructed and validated.

**Table 5-8:** The types of classifiers constructed to evaluate the performances of the single-model and multi-model classifiers.

Type	Classifier	Features Used
Single-Model	SMC	8, 10, 12, 14, and 16 most relevant
	Naïve Bayes	15 most relevant
	k-NN	Two most relevant
Multi-Model	FDMM	Two (similarity ratio and probability ratio)
	PDMM	Two (similarity ratio and probability ratio)

## 5.5 Results and Discussion

### 5.5.1 Classification Performances

The observed performances of the classification models constructed using both single-model and multi-model approaches are presented and evaluated.



### 5.5.1.1 *Single-Model Performances*

The SMC-based classifiers have been constructed and tested using a different number of features to determine the ideal number. The classifiers constructed using 16 and 10 features gave the best accuracy results in the NSL-KDD and CICIDS2017 datasets, respectively. The highest accuracy observed in the NSL-KDD dataset was 81.13%, and in the CICIDS2017 dataset, it was 75.60%. **Table 5-9** demonstrates the performance achieved by the SMC-based classifiers.

**Table 5-9:** The performance comparisons of the SMC-based classifiers constructed using a varying number of features.

Dataset	Features	Recall	Precision	Specificity	Accuracy
NSL-KDD	8	89.20	73.93	71.05	79.75
	10	88.92	73.89	<b>71.08</b>	79.63
	12	91.80	74.50	<b>71.08</b>	81.01
	14	91.95	74.52	71.07	81.08
	16	<b>92.06</b>	<b>74.55</b>	71.07	<b>81.13</b>
CICIDS2017	8	75.43	73.24	72.52	73.97
	10	<b>78.89</b>	<b>73.97</b>	72.32	<b>75.60</b>
	12	74.49	72.34	71.61	73.05
	14	63.23	73.65	<b>77.45</b>	70.35
	16	66.48	72.66	75.06	70.77

The Naïve Bayes and k-NN classifiers demonstrated superior performance over the SMC-based classifiers. In the NSL-KDD dataset, k-NN performed the best with the accuracy of 94.51%, and in the CICIDS2017 dataset, Naïve Bayes exceeded the performance of k-NN. It must be noted that Naïve Bayes used 15 features, while k-NN used only two. **Table 5-10** records the performances of Naïve Bayes and k-NN.

**Table 5-10:** The performance comparisons of the single-model Naïve Bayes and k-NN classifiers.

Dataset	Classifier	Recall	Precision	Specificity	Accuracy
NSL-KDD	Naïve Bayes	90.07	96.10	96.63	93.49
	k-NN	93.46	95.00	95.47	94.51
CICIDS2017	Naïve Bayes	93.77	86.75	85.72	89.74
	k-NN	93.60	85.94	84.73	89.16

### 5.5.1.2 *Multi-Model Performances*

The FDMM-based classifiers that entirely relied on k-NN for the final classification decision performed the best by achieving the accuracy of 96.89% and 96.77%, for the NSL-KDD and CICIDS2017 datasets, respectively. In contrast, the PDMM, which involved k-NN only when SMC and Naïve Bayes failed to reach a mutual agreement, achieved an accuracy of 94.77% and 92.39% for each dataset, respectively. These accuracies were still higher than those achieved by the single-model classifiers. Even though the FDMM performed better in terms of accuracy, the classifiers based on it took much longer to classify all the test instances. **Table 5-11** displays the performances observed when using the multi-model approach for classification.

**Table 5-11:** The observed performances when using multi-model approaches based on SMC, Naïve Bayes, and k-NN.

Dataset	Classifier	Recall	Precision	Specificity	Accuracy
NSL-KDD	PDMM	93.28	95.70	96.15	94.77
	FDMM	96.87	96.63	96.90	96.89
CICIDS2017	PDMM	96.29	89.31	88.51	92.39
	FDMM	97.56	96.03	95.98	96.77

### 5.5.1.3 *FPR Analysis*

The observations reported in **Table 5-12** show that number of false-positives can be decreased by using a multi-model approach. The FDMM approach achieved the best FPRs, which were 3.10% and 4.02% for the NSL-KDD and CICIDS2017 datasets, respectively. The FPR decreased drastically in the CICIDS2017 dataset.

**Table 5-12:** The best FPR achieved by each classifier in all tested datasets.

Dataset	Classifier	Test Instances	False Positives	FPR
NSL-KDD	SMC	29,703	4,473	28.92
	Naive Bayes	29,703	521	3.37
	k-NN	29,703	700	4.53
	PDMM	29,703	596	3.85
	FDMM	29,703	<b>480</b>	<b>3.10</b>
CICIDS2017	SMC	200,000	22,581	22.55
	Naïve Bayes	200,000	14,297	14.28
	k-NN	200,000	15,288	15.27
	PDMM	200,000	11,511	11.49
	FDMM	200,000	<b>4,031</b>	<b>4.02</b>

## 5.6 Conclusions

In this chapter, we introduced the SMC technique, which uses the frequential weight of the values associated with a specific class. Just like the Naïve Bayes, this method works by merely utilizing the values derived from the counted frequencies, so it was quick. We tested and validated SMC by constructing the classifiers using the NSL-KDD and CICIDS2017 datasets. The best accuracies observed using this technique on these datasets were 81.13% and 75.60%, respectively. These performances were

subpar in comparison to the performances demonstrated by the single-model Naïve Bayes and k-NN classifiers. SMC, despite being the weakest performer, contributed to the multi-model classification approach to improve the overall classification accuracy.

We introduced two variants of the multi-model classification technique for binary classification. In PDMM, the tertiary classifier participated in the classification process only when SMC and Naïve Bayes failed to make a mutual decision. In the other variant, FDMM, all the classifications were done by a tertiary classifier by using the information produced by SMC and Naïve Bayes. It was clear through experiments that the multi-model approach can improve the accuracy of the classification. The FDMM-based approach gave an accuracy of 96.89% in the NSL-KDD dataset and 96.77% in the CICIDS2017 dataset, in contrast to the best accuracies of 94.51% and 89.74% given by the single-model approaches for those datasets, respectively. Even though FDMM gave the best result, it took a long time to process all the instances for classification; on the other hand, the PDMM-based model took a much shorter time because only a limited number of instances had to pass through the third classifier.

Furthermore, we also analyzed the FPR-based performances. When the FDMM approach was used, the number of false positives reduced drastically to 3.10% in the NSL-KDD and 4.02% in the CICIDS2017 datasets. Since the IDSs continue to suffer from high FPRs in general, such reduction in FPR when using a multi-model approach is a promising achievement. Based on these observations, we conclude that the classification accuracy can be improved while diminishing the FPR by using the multi-model classification approach.

## CHAPTER 6

### CONCLUSIONS AND FUTURE WORK

#### 6.1 Conclusions

The primary objectives of this research hovered around identifying the potential improvements in IDSs. The traditional IDSs suffer from high-dimensionality data, single point of failure, slow operations, inability to adapt to new attacks, and low accuracy. The ideas we expressed throughout this dissertation attempted to tackle these issues. Initially, we discussed distributed feature extraction and classifier construction techniques. Additionally, we proposed a new similarity measure-based supervised classification method for categorical data and introduced a multi-model approach for binary classification.

##### 6.1.1 Distributed Feature Extraction

In CHAPTER 3, we conducted PCA-based distributed feature extraction using the initial set of descriptors. The dimensionality of data was reduced significantly after feature extraction, which consequently constrained the feature space. Since multiple nodes simultaneously collaborated with the assistance of the central coordinator to extract the features, the total time taken was also drastically shortened. The features so extracted were used to construct various classifiers to verify their effectiveness. The observations made implied that the classifiers constructed using the features extracted in a distributed

manner performed well. Their accuracy-based performance was competitive with that of the classifiers constructed with the original and centrally extracted features. Given this method's advantages like shortened extraction time, improved privacy, and limited data exchange requirements, it is appropriate for IDS construction. The discussed feature extraction technique can be useful in the distributed networks supporting high-volume data exchanges.

#### 6.1.2 Distributed Classifier Construction

In CHAPTER 4, we utilized the features extracted in CHAPTER 3 to construct a classifier in a distributed manner using the Naïve Bayes-based technique. Its fast speed and high scalability have established it as an ideal choice for systems like IDSs that require quick model training and attack detection. Since Naive Bayes works by computing the prior probabilities, which depend on frequency counting, we presented the process to perform frequency counting in a distributed manner with the help of the central coordinator. The global frequencies of all the observed values and classes in the training set were shared with each participating node by the central coordinator. Each node, which uses the global frequencies to compute the prior probabilities, can perform classifications. Since the training in the Naïve Bayes only involves frequency counting and determining prior probabilities, the time taken to undertake these was significantly shorter when these tasks were distributed across multiple nodes. We observed that the classifiers constructed in a distributed manner give a similar level of accuracies as the ones constructed centrally. Since each node could classify the new instances independently, a distributed method also improved the robustness.

### 6.1.3 SMC and Multi-Model Approach for Binary Classification

In CHAPTER 5, we presented a similarity measure-based learning method and a multi-model approach for classification. Like with Naïve Bayes, the training of SMC relied on the frequency counts. The counted frequencies were used to determine the class-specific weights for each of the values. Those weights were used to compute the similarity with the data centroid. An instance was classified into the class whose weights vector had the highest similarity with the centroid. The performance of this classification technique was not impressive; however, the similarities computed by it were used alongside the probabilities computed by the Naïve Bayes classifier to form another classifier. In such an approach, the SMC's outputs contributed to improving accuracy.

In a discussed multi-model approach, we used the similarity ratio and probability ratio determined using the similarities and probabilities computed by SMC and Naïve Bayes as features to train and validate the k-NN-based classifiers. The PDMM variant of the multi-model approach involved k-NN only when SMC and Naïve Bayes failed to classify an instance into the same class. In contrast, FDMM always used k-NN for the final classification. The multi-model approaches, as expected, improved the overall accuracy of the classification. The FDMM variant of the multi-model approach significantly decreased the FPRs. Such improvement in performance showed that it is possible to use the outputs of multiple lightweight classification models and use those outputs as an input for another classifier to perform a more accurate classification.

### 6.1.4 Final Discussion

In this dissertation, we successfully implemented a distributed feature extraction technique for dimensionality reduction in a simulated distributed environment where each

node only had access to a subset of data. By constructing and validating the classifiers with the extracted features, we demonstrated that these features work as effectively as the features extracted centrally. We also constructed and implemented a distributed classifier based on a probabilistic model, which utilized the extracted features. This distributed classification model performed comparatively against the centralized model, while significantly diminishing the model-training and attack-detection durations. Similarly, we also proposed a similarity measure-based classification technique and used it to build an IDS classifier. Finally, we undertook a multi-model classification approach that relied on the information outputted by the probabilistic and similarity measure-based classifiers to construct a tertiary classifier. This multi-model approach was successful in improving the accuracy of classification. The promising results we observed throughout the dissertation when using the presented techniques and concepts make them noteworthy for future endeavors.

## **6.2 Future Work**

There are countless possible directions to explore. The concepts discussed are presumed to be applicable in a real-world scenario to construct an IDS classifier. Since all the experiments were conducted in a simulated environment on a single host machine, it would be sensible to undertake these in an actual distributed network and observe the effects. Since only two pre-existing datasets were used to construct the prediction models for experiments, experimenting with more datasets could give a better understanding of how the presented techniques would adapt to and perform on other datasets. The classifiers could also be constructed by using customized data pertaining to a specific



type of network; then, it could be tested live by deploying the built classifier into an IDS for that network.

There are also numerous avenues for improvement within the dissertation. For instance, in all distributed procedures, the nodes were assumed to be homogenous. In circumstances when all the nodes do not have equal resources, type of data, or the size of datasets, then the applicability and the observations can differ. Similarly, for the SMC, the cosine similarity measure was used. A different similarity measure could give a different outcome. It would be within the purview to try other similarity measures. In the proposed multi-model approach, the third classifier has been constructed using k-NN. It, however, could also be constructed using different algorithms. Furthermore, in the PDMM variant of the multi-model approach, additional adjustments could be made to decide which instances get sent to the tertiary classifier, instead of solely basing it on whether SMC and Naïve Bayes made a mutual classification.

We expect the relevant future works to consider this work and build upon it to enhance the state of IDSs.

## BIBLIOGRAPHY

- [1] K. Kent, S. Chevalier, T. Grance and H. Dang, "Guide to Integrating Forensic Techniques into Incident Response," *National Institute of Standards & Technology*, 2006.
- [2] J. Mikhail, J. Fossaceca and R. Iammartino, "A Semi-Boosted Nested Model With Sensitivity-Based Weighted Binarization for Multi-Domain Network Intrusion Detection," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 3, pp. 1-27, April 2019.
- [3] B. I. Santoso, M. R. S. Idrus and I. P. Gunawan, "Designing Network Intrusion and Detection System using Signature-Based Method for Protecting OpenStack Private Cloud," in *2016 6th International Annual Engineering Seminar (InAES)*, Jakarta, 2016.
- [4] M. A. A. S. Monther Aldwairi, "Characterizing Realistic Signature-based Intrusion Detection Benchmarks," in *ICIT 2018: Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*, Hong Kong, 2018.
- [5] T. D. Leyla Bilge, "Investigating Zero-Day Attacks," *login.*, vol. 38, no. 4, pp. 6-12, August 2013.
- [6] S. R. Snapp, S. E. Smaha, D. M. Teal and T. Grance, "The DIDS (Distributed Intrusion Detection System) Prototype," in *Summer '92 USENIX*, San Antonio, TX, 1992.
- [7] C. Clark, W. Lee, D. Schimmel, D. Contis, M. Koné and A. Thomas, "A hardware platform for network intrusion detection and prevention," in *Network Processor Design: Issues and Practices*, vol. 3, Morgan Kaufmann, 2005, pp. 99-118.
- [8] A. Khraisat, I. Gondal, P. Vamplew and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 20, July 2019.

- [9] C. Fung, "Collaborative intrusion detection networks and insider attacks," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 2, no. 1, pp. 63-74, 2012.
- [10] N. Lakshminarayan, "Know Your Data Before You Undertake Research," *The Journal of Indian Prosthodontic Society*, vol. 13, no. 3, pp. 384-386, July 2013.
- [11] S.-A. N. Alexandropoulo, S. B. Kotsiantis and M. N. Vrahatis, "Data preprocessing in predictive data mining," *The Knowledge Engineering Review*, vol. 34, pp. 1-33, 2019.
- [12] R. K. S. James Dougherty, "Supervised and Unsupervised Discretization of Continuous Features," in *Machine Learning: Proceedings of the Twelfth International Conference*, Tahoe City, CA, 1995.
- [13] A. R. Webb and K. D. Copsey, "Feature Selection and Extraction," in *Statistical Pattern Recognition, Third Edition*, John Wiley & Sons, Ltd., 2011, pp. 305-354.
- [14] R. C. Arthur Munson, "On Feature Selection, Bias-Variance, and Bagging," in *ECMLPKDD'09: Proceedings of the 2009th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, Berlin, Heidelberg, 2009.
- [15] R. Raphael, V. P. and B. Omman, "X-ANOVA ranked features for Android malware analysis," in *2014 Annual IEEE India Conference (INDICON)*, Pune, 2014.
- [16] S. Lei, "A Feature Selection Method Based on Information Gain and Genetic Algorithm," in *2012 International Conference on Computer Science and Electronics Engineering*, Hangzhou, 2012.
- [17] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A*, vol. 374, no. 2065, April 2016.
- [18] N. M. Varma and A. Choudhary, "Evaluation Of Distance Measures In Content Based Image Retrieval," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, 2019.
- [19] O. Kilinc and I. Uysal, "Source-Aware Partitioning for Robust Cross-Validation," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Miami, FL, 2015.

- [20] "NSL-KDD dataset," [Online]. Available: [www.unb.ca/cic/datasets/nsl.html](http://www.unb.ca/cic/datasets/nsl.html). [Accessed 01 March 2019].
- [21] "Intrusion Detection Evaluation Dataset (CICIDS2017)," Canadian Institute of Cybersecurity, [Online]. Available: [www.unb.ca/cic/datasets/ids-2017.html](http://www.unb.ca/cic/datasets/ids-2017.html). [Accessed 03 August 2019].
- [22] L. Dhanabal and S. Shantharajah, "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, pp. 446-452, June 2015.
- [23] R. Panigrahi and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems," *International Journal of Engineering & Technology*, vol. 7, no. 3.24, pp. 479-482, 2018.
- [24] M. A. Ambusaidi, X. He, P. Nanda and Z. Tan, "Building an Intrusion Detection System Using a Filter-Based Feature Selection Algorithm," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2986-2998, 2016.
- [25] M. O. Ulfarsson and V. Solo, "Selecting the Number of Principal Components with SURE," *IEEE Signal Processing Letters*, vol. 22, no. 2, pp. 239-243, February 2015.
- [26] Z. Zhang, F. Li, M. Zhao, L. Zhang and S. Yan, "Joint Low-Rank and Sparse Principal Feature Coding for Enhanced Robust Representation and Visual Classification," *IEEE Transactions on Image Processing*, vol. 25, no. 6, pp. 2429-2443, June 2016.
- [27] R. G. Staudte and S. J. Sheather, "Linear Algebra Results," in *Robust Estimation and Testing*, John Wiley & Sons, Inc., 1990, pp. 279-286.
- [28] J. Lever, M. Krzywinski and N. Altman, "Principal component analysis," *Nature Methods*, vol. 14, pp. 641-642, 29 June 2017.
- [29] S. X. Wu, H.-T. Wai, L. Li and A. Scaglione, "A Review of Distributed Algorithms for Principal Component Analysis," *Proceedings of the IEEE*, vol. 106, no. 8, pp. 1321-1340, August 2018.
- [30] J. Fan, D. Wang, K. Wang and Z. Zhu, "Distributed estimation of principal eigenspaces," *The Annals of Statistics*, vol. 47, no. 6, pp. 3009-3031, 2019.

- [31] C. O'Reilly, A. Gluhak and M. A. Imran, "Distributed Anomaly Detection Using Minimum Volume Elliptical Principal Component Analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2320-2333, 01 September 2016.
- [32] D. A. Tarzanagh, M. K. S. Faradonbeh and G. Michailidis, "Online Distributed Estimation of Principal Eigenspaces," in *2019 IEEE Data Science Workshop (DSW)*, Minneapolis, MN, 2019.
- [33] M. Toulouse, B. Q. Minh and P. Curtis, "A Consensus Based Network Intrusion Detection System," in *2015 5th International Conference on IT Convergence and Security (ICITCS)*, Kuala Lumpur, 2015.
- [34] G. V. Bard, "Uses and misuses of Bayes' rule and Bayesian classifiers in cybersecurity," in *AIP Conference Proceedings 1910*, 2017.
- [35] L. Jiang, L. Zhang, C. Li and J. Wu, "A Correlation-Based Feature Weighting Filter for Naive Bayes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 2, pp. 201-213, February 2019.
- [36] R. Vijayasathy, S. V. Raghavan and B. Ravindran, "A system approach to network modeling for DDoS detection using a Naive Bayesian classifier," in *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, Bangalore, 2011.
- [37] A. Ashari, I. Paryudi and A. M. Tjoa, "Performance Comparison between Naïve Bayes, Decision Tree and k-Nearest Neighbor in Searching Alternative Design in an Energy Simulation Tool," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 11, pp. 33-39, 2013.
- [38] F. B. Sebastian Schelter, T. Januschowski, D. Salinas, S. Seufert and G. Szarvas, "On Challenges in Machine Learning Model Management," *IEEE Data Engineering Bulletin*, vol. 41, pp. 5-15, 2018.
- [39] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser and M. Fischer, "Taxonomy and Survey of Collaborative Intrusion Detection," *ACM Computing Surveys*, vol. 47, no. 4, 2015.
- [40] Y. Hu, D. Niu, J. Yang and S. Zhou, "FDML: A Collaborative Machine Learning Framework for Distributed Features," in *KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Achorage, 2019.

- [41] J. Yang, Z. Ye, L. Yan, W. Gu and R. Wang, "Modified Naive Bayes Algorithm for Network Intrusion Detection based on Artificial Bee Colony Algorithm," in *2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, Lviv, 2018.
- [42] C. J. Fung, Q. Zhu, R. Boutaba and T. Başar, "Bayesian decision aggregation in collaborative intrusion detection networks," in *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, Osaka, 2010.
- [43] M. Droettboom, "Understanding JSON Schema," 22 October 2019. [Online]. Available: [json-schema.org/understanding-json-schema/UnderstandingJSONSchema.pdf](https://json-schema.org/understanding-json-schema/UnderstandingJSONSchema.pdf). [Accessed 04 03 2020].
- [44] D. J. Weller-Fahy, B. J. Borghetti and A. A. Sodemann, "A Survey of Distance and Similarity Measures Used Within Network Intrusion Anomaly Detection," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 70-91, 11 July 2014.
- [45] M. P. Kumar and D. Koller, "MAP Estimation of Semi-Metric MRFs via Hierarchical Graph Cuts," in *UAI '09: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, Montreal, 2009.
- [46] J. Han, M. Kamber and J. Pei, "Getting to Know Your Data," in *Data Mining: Concepts and Techniques, The Morgan Kaufmann Series in Data Management Systems*, 2012, pp. 39-82.
- [47] M. Biehl, B. Hammer and T. Villmann, "Distance Measures for Prototype Based Classification," in *Brain-Inspired Computing. BrainComp 2013.*, 2014.
- [48] I. Ahmed, A. Dagnino and Y. Ding, "Unsupervised Anomaly Detection Based on Minimum Spanning Tree Approximated Distance Measures and Its Application to Hydropower Turbines," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 654-667, April 2019.
- [49] H. Jia, Y.-m. Cheung and J. Liu, "A New Distance Metric for Unsupervised Learning of Categorical Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 5, pp. 1065-1079, May 2016.
- [50] C. Kruegel, G. Vigna and W. Robertson, "A multi-model approach to the detection of web-based attacks," *Computer Networks*, vol. 48, no. 5, pp. 717-738, 2005.