

2018

# Probabilistic Clustering Ensemble Evaluation for Intrusion Detection

Steven M. McElwee

Nova Southeastern University, [mcelwee@ieee.org](mailto:mcelwee@ieee.org)

This document is a product of extensive research conducted at the Nova Southeastern University [College of Engineering and Computing](#). For more information on research and degree programs at the NSU College of Engineering and Computing, please click [here](#).

Follow this and additional works at: [https://nsuworks.nova.edu/gscis\\_etd](https://nsuworks.nova.edu/gscis_etd)

 Part of the [Computer Sciences Commons](#)

## Share Feedback About This Item

---

### NSUWorks Citation

Steven M. McElwee. 2018. *Probabilistic Clustering Ensemble Evaluation for Intrusion Detection*. Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (1044)  
[https://nsuworks.nova.edu/gscis\\_etd/1044](https://nsuworks.nova.edu/gscis_etd/1044).

This Dissertation is brought to you by the College of Engineering and Computing at NSUWorks. It has been accepted for inclusion in CEC Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact [nsuworks@nova.edu](mailto:nsuworks@nova.edu).

# Probabilistic Clustering Ensemble Evaluation for Intrusion Detection

by


Steven M. McElwee

A dissertation submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy  
in  
Information Assurance

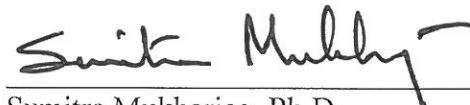
College of Engineering and Computing  
Nova Southeastern University

2018

We hereby certify that this dissertation, submitted by Steven McElwee, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

  
\_\_\_\_\_  
James D. Cannady, Ph.D.  
Chairperson of Dissertation Committee

6/27/2018  
Date

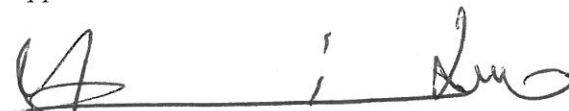
  
\_\_\_\_\_  
Sumitra Mukherjee, Ph.D.  
Dissertation Committee Member

6/27/2018  
Date

  
\_\_\_\_\_  
Paul S. Cerkez, Ph.D.  
Dissertation Committee Member

6/27/2018  
Date

Approved:

  
\_\_\_\_\_  
Yong X. Tao, Ph.D., P.E., FASME  
Dean, College of Engineering and Computing

6/27/2018  
Date

College of Engineering and Computing  
Nova Southeastern University

2018

An Abstract of a Dissertation Submitted to Nova Southeastern University  
in Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy

## Probabilistic Clustering Ensemble Evaluation for Intrusion Detection

by  
Steven M. McElwee  
2018

Intrusion detection is the practice of examining information from computers and networks to identify cyberattacks. It is an important topic in practice, since the frequency and consequences of cyberattacks continues to increase and affect organizations. It is important for research, since many problems exist for intrusion detection systems. Intrusion detection systems monitor large volumes of data and frequently generate false positives. This results in additional effort for security analysts to review and interpret alerts. After long hours spent reviewing alerts, security analysts become fatigued and make bad decisions. There is currently no approach to intrusion detection that reduces the workload of human analysts by providing a probabilistic prediction that a computer is experiencing a cyberattack.

This research addressed this problem by estimating the probability that a computer system was being attacked, rather than alerting on individual events. This research combined concepts from cyber situation awareness by applying clustering ensembles, probability analysis, and active learning. The unique contribution of this research is that it provides a higher level of meaning for intrusion alerts than traditional approaches.

Three experiments were conducted in the course of this research to demonstrate the feasibility of these concepts. The first experiment evaluated cluster generation approaches that provided multiple perspectives of network events using unsupervised machine learning. The second experiment developed and evaluated a method for detecting anomalies from the clustering results. This experiment also determined the probability that a computer system was being attacked. Finally, the third experiment integrated active learning into the anomaly detection results and evaluated its effectiveness in improving the accuracy.

This research demonstrated that clustering ensembles with probabilistic analysis were effective for identifying normal events. Abnormal events remained uncertain and were assigned a belief. By aggregating the belief to find the probability that a computer system was under attack, the resulting probability was highly accurate for the source IP addresses and reasonably accurate for the destination IP addresses. Active learning, which simulated feedback from a human analyst, eliminated the residual error for the destination IP addresses with a low number of events that required labeling.

## **Acknowledgements**

To the giants of research on whose shoulders I stood to see a little farther, it has been a privilege to build upon the foundation you have laid.

To Dr. James Cannady, my sincerest gratitude for allowing me to build upon your work in intrusion detection and for your willingness to serve as my dissertation committee chair.

To my dissertation committee, who provided constructive criticism throughout the process, thank you for your patience and for your feedback. You have helped to improve my research skills.

To Allison, my loving and supportive wife, thank you for sacrificing our time together for late study nights and for trips to Florida. I could not have accomplished this without your encouragement and understanding.

To my children, who themselves have grown and accomplished so much since I started this pursuit, thank you for encouraging me along the way. Remember to continue to learn, regardless of your age.

# Table of Contents

**Abstract** ii

**List of Tables** vi

**List of Figures** vii

## Chapters

### **1. Introduction 1**

Background 1

Problem Statement 2

Dissertation Goal 2

Discussion 3

Relevance and Significance 6

Barriers and Issues 7

Definition of Terms 9

List of Acronyms 12

Summary 13

### **2. Review of the Literature 15**

Overview of Reviewed Topics 15

Cyber Situation Awareness 16

Intrusion Detection 22

Probabilistic Intrusion Detection 31

Clustering Ensembles 36

Intrusion Detection Datasets 52

Summary 58

### **3. Methodology 59**

Introduction 59

Solution Design 60

Experiment 1: Cluster Generation 65

Experiment 2: Probabilistic Anomaly Detection 68

Experiment 3: Active Learning 71

Resource Requirements 72

Summary 73

### **4. Results 75**

Introduction 75

Experiment Design and Implementation 76

Input Dataset Analysis and Preparation 79

Experiment 1: Cluster Generation 82

Experiment 2: Probabilistic Anomaly Detection 88

Experiment 3: Active Learning 97  
Summary 101

**5. Conclusions, Implications, Recommendations, and Summary 103**

Conclusions 103  
Implications 105  
Recommendations 106  
Summary 108

**References 147**

**Appendices**

- A. Source Code Availability and Usage 115**
- B. Dataset Descriptions 119**
- C. Python Package Versions 125**
- D. Detailed Anomaly Detection Results 126**

## List of Tables

### Tables

1. Summary of Cyber Situation Awareness Literature 19
2. Summary of Machine Learning Approaches to Intrusion Detection 23
3. Summary of Co-Occurrence Cluster Evaluation Approaches 45
4. Summary of Median Partition Clustering Evaluation Approaches 49
5. Summary of Intrusion Detection Dataset Research 55
6. Original and Derived NSL-KDD Dataset Label Distributions 80
7. UNSW-NB15\_1 Dataset Label Distributions 82
8. Distribution of Attack and Normal Records in Sample Partition 85
9. Prediction of  $P(A)$  by srcip and dstip 94
10.  $P(A)$  Accuracy for Ten Runs 96
11. NSL-KDD Attributes and Datatypes 120
12. UNSW-NB15 Attributes and Datatypes 122
13. Python Package Versions 125
14. Experiment 2, Run 1 Results 127
15. Experiment 2, Run 2 Results 129
16. Experiment 2, Run 3 Results 131
17. Experiment 2, Run 4 Results 133
18. Experiment 2, Run 5 Results 135
19. Experiment 2, Run 6 Results 137
20. Experiment 2, Run 7 Results 139



21. Experiment 2, Run 8 Results 141
22. Experiment 2, Run 9 Results 143
23. Experiment 2, Run 10 Results 145

## List of Figures

### Figures

1. Multiple ways to cluster a deck of cards 4
2. Clustering ensemble overview 5
3. Confusion matrix for binary classification 29
4. Overview clustering ensembles with formal notation 39
5. High level solution design 60
6. Algorithm for cluster generation 67
7. Algorithm for probabilistic anomaly detection 69
8. Class diagram 76
9. Graph of votes for NSL-KDD dataset 89
10. Graph of votes for UNSW-NB15 dataset 90
11. Pseudocode for active learning algorithm 98
12. Accuracy of active learning compared to sample size 100

## Chapter 1

### Introduction

#### Background

Intrusion detection systems identify cyberattacks. Malicious or criminal cyberattacks take an average of 229 days to detect, and the length of time to detect and contain an attack increases the cost of response (Ponemon Institute, 2016). Consequently, detecting cyberattacks quickly is vital for organizations. Because of the volume of intrusion alerts, cyberattacks are sometimes miscategorized in the large number of alerts that require human analyst review (Julisch, 2003). After long hours of review, analysts make mistakes, and alerts may be miscategorized (Sawyer et al., 2014). Further, human analysts make inaccurate decisions and use preconceived biases when dealing with probabilistic reasoning (Tversky & Kahneman, 1974). Most intrusion detection research has focused on identifying individual events and has not provided meaning in the broader perspective of cyber situation awareness (CSA) (Sommer & Paxson, 2003; Erbacher, Frincke, Wong, Moody, & Fink, 2010; Sommer & Paxson, 2010; Tadda & Salerno, 2010). Detecting intrusions at the individual event level is prone to high false positive rates and overfitting (Sommer & Paxson, 2010). Thus, new approaches to intrusion detection are needed to provide better support for human decision-making under uncertain conditions.

This research developed a system for anomaly-based intrusion detection. The system incorporated multiple views of anomalies to find the probability that a computer

system was under attack or had been compromised. The novelty of this research was the application of clustering ensembles, probability analysis, and active learning to extend research in intrusion detection. This research also incorporated relevant concepts from CSA to add meaning to intrusion alerts.

### **Problem Statement**

There is currently no approach to intrusion detection that reduces the workload of human analysts by providing a probabilistic prediction that a computer is experiencing a cyberattack. Intrusion detection systems monitor increasingly large datasets that represent interconnected devices and sensors (Saeed, Ahmadinia, Javed, & Larijani, 2016; Al-Hamadi & Chen, 2015; Ali & Al-Shaer, 2015). The alerts generated by intrusion detection systems require human review to evaluate the accuracy of the alerts and to determine an appropriate course of action (Julisch, 2003). A significant problem is that intrusion alerts often have high false-positive rates, since intrusions are rare in large datasets (Kruegel, Mutz, Robertson, & Valeur, 2003; Scott, 2004). Thus, security analysts become fatigued and make poor decisions after spending hours reviewing alerts (Sawyer et al., 2014). Improvements in intrusion detection systems are needed to reduce false-positives, improve the context of alerts, and reduce the burden on human analysts.

### **Dissertation Goal**

The goal of this research was to improve anomaly-based intrusion detection by adding meaning to alerts through the use of probabilistic clustering ensembles. Adding meaning to alerts shifts the focus from the individual event level to the computer system

level. This research developed a method to reduce the workload of security analysts by focusing on the computer systems most likely experiencing attacks. The results of this research will allow security analysts to better prioritize their monitoring activity, which has been found to be important in practice (McElwee, Heaton, Fraley, & Cannady, 2017).

## **Discussion**

An important objective of this research was to apply clustering ensembles to intrusion detection. There is not one correct way to cluster network information to identify anomalies. Instead, there are multiple perspectives that, when taken together, improve accuracy. This idea is supported from prior research, which acknowledges that “clustering is in the eye of the beholder” (Estivill-Castro, 2002, p. 65). To illustrate this idea, consider that a reasonable person is given a deck of playing cards, with 52 cards of four suits, and two joker cards. The person is given direction to cluster the cards into meaningful groups. There are a wide variety of ways that the person might cluster the cards. The person might cluster the cards using an obvious feature, such as the face value, by grouping together all the aces in one group, the twos in another group, the threes in another group, and so on. Such a grouping results in 13 clusters of four cards each and a cluster with two jokers. The person might opt for four clusters, where each cluster is identified by the suit of the card, specifically diamonds, hearts, clubs, and spades. This grouping results in four clusters with 13 cards in each. In this case, the jokers do not cluster well, but may be considered anomalies. The person might group the cards using a feature derived from outside information, such as grouping the cards needed to assemble a deck for a special game. Such a group might consist of the nines, tens, jacks, queens,

kings, and aces for playing the game of pinochle. The second group is those not needed to play that game. Figure 1 illustrates this example using two sample clusterings.

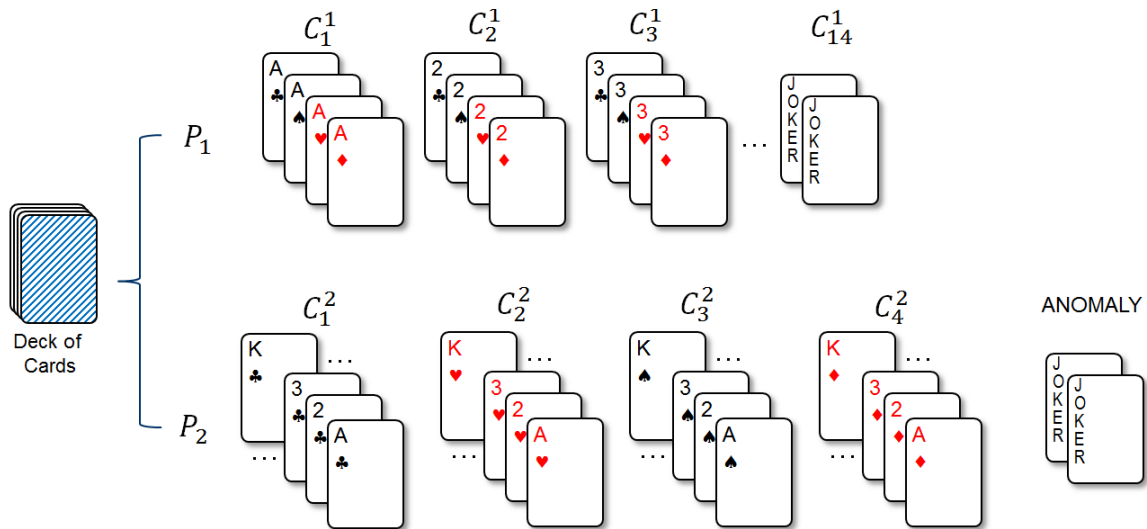


Figure 1. Multiple ways to cluster a deck of cards

From the playing card example above, it is obvious that there is not one correct way to cluster the cards. Each clustering result has different meaning for different reasons. A wide variety of clustering results is possible based on the features chosen to create the groups. Even the anomalies, as represented in  $P_2$  by the jokers, are only anomalies because of the meaning the person placed on the clustering. Thus, in some clustering approaches, the jokers cluster well, but in others they appear to be anomalies.

This research used clustering ensembles, which use many different clusterings of the data to create a clustering solution that works best (Strehl & Gosh, 2002; Fred & Jain, 2005). Figure 2 illustrates the general approach to clustering ensembles. The first stage generates a diverse set of clustering solutions,  $\mathbb{P}$ , from among all possible clustering solutions,  $\mathbb{P}_X$ . The second stage evaluates the results to arrive at a final clustering solution,  $P^*$ .

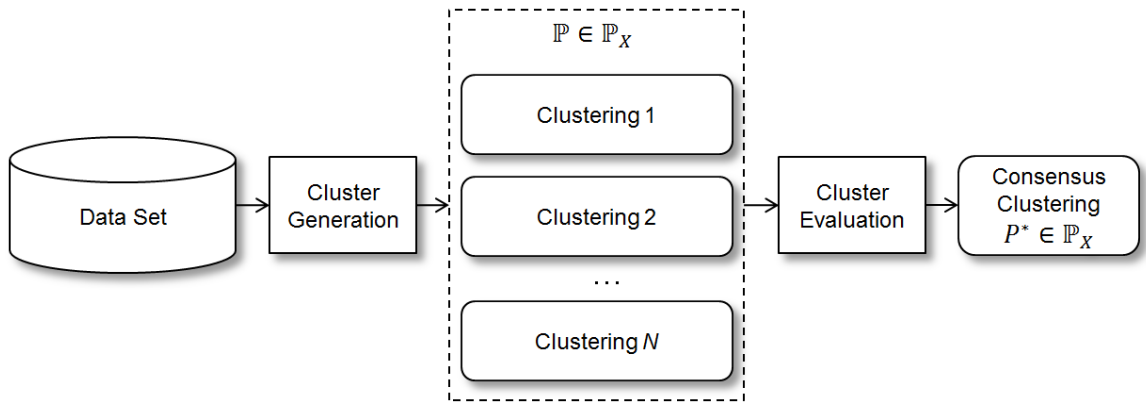


Figure 2. Clustering ensemble overview

This research used clustering ensembles because of their potential to be useful for intrusion detection, since they enable unsupervised machine learning using multiple sources of information, including domain knowledge, in the final clustering solution (Strehl & Ghosh, 2002). Ensemble approaches to machine learning have proven to be successful in resisting adversarial evasion (Šrndić & Laskov, 2014; Wang, Wang, Zheng, & Zhao, 2014). Evasion is a tactic to conduct cyberattacks without being detected. There have been few applications of clustering ensembles to intrusion detection (Weng, Jiang, Shi, & Wu, 2007; Gao, Zhu, & Wang, 2010). Other related studies have shown that clustering ensembles have the potential to be effective when applied to intrusion detection. Lazarevic and Kumar (2005) used an approach similar to clustering ensembles, called feature bagging, and found it to be successful for intrusion detection. Hou, Chen, Tas, Demihovski, and Ye (2015) found that clustering ensembles were better for the detection of malware than single base clustering algorithms.

The importance of adding meaning to alerts is established in prior research. Sommer and Paxson (2003) found that adding context to intrusion detection improved signature-based methods by detecting multi-step attacks. Machine learning algorithms for

intrusion detection can find what is abnormal, but generate false positives, because they lack meaning (Sommer & Paxson, 2010). As a result, security analysts must evaluate binary indications of cyberattacks with insufficient meaning to make appropriate decisions. Thus, this research created a semantic level that better represents how human analysts evaluate security alerts.

Computer network intrusions are generally a multi-step process that requires an attacker and a victim. (Zhou, Leckie, & Karunasekera, 2010). Understanding the meaning and significance of network events requires an understanding of the coordination of events (Zhou et al., 2010). The connection of these events is often uncertain and requires probabilistic reasoning. Thus, this research used probability analysis to evaluate if a computer system was being attacked.

### **Relevance and Significance**

Intrusion detection is an important topic in research as well as in practice, and identifying cyberattacks quickly is important for reducing response costs (Ponemon Institute, 2016). Unfortunately, intrusion alerts are sometimes missed because of the volume that must be reviewed (Julisch, 2003). After long hours of reviewing security alerts, human analysts make poor decisions, and alerts may be miscategorized (Sawyer et al., 2014). Humans tend to make inaccurate decisions and use preconceived biases when dealing with probabilistic reasoning (Tversky & Kahneman, 1974). In addition, adversarial tactics and evasion require new approaches for the identification of malicious activity (Šrندیć & Laskov, 2014; Wang et al., 2014). As a result, improving intrusion detection is a relevant topic for continued research to provide better support for decision-



making under uncertain conditions. Given the importance of intrusion detection and the limitations of human judgement, this research provides context for analysts, so they can make better decisions. The results of this research may be used to augment security analyst capabilities for responding to intrusion alerts.

### **Barriers and Issues**

This research addressed several barriers and issues that are present in anomaly-based intrusion detection research. First, anomaly-based intrusion detection is prone to high false-positive rates. This is largely due to the imbalanced nature of intrusion detection data, since there are large amounts of data, but only a few intrusion events (Scott, 2004). The base rate fallacy stipulates that when events of interest are rare, even highly accurate detection systems may have a high false positive rate (Kruegel et al., 2003). This research addressed this issue by focusing on the probability that a computer system was being attacked rather than on determining the accurate classification of individual events (Li, Ou, & Rajagopalan, 2010; Tadda & Salerno, 2010).

A second issue in intrusion detection research is overfitting. To achieve higher accuracy, researchers may over-train learning algorithms or use too much training data to be generalizable for novel anomalies. This is because machine learning algorithms are more suitable for detecting similarities than for detecting anomalies (Sommer & Paxson, 2010). Overfitting may also result from the selection of a dataset that contains too many duplicate records, such as the popular KDD Cup 1999 dataset (Tavallae, Bagheri, Lu, & Ghorbani, 2009). Since ensemble approaches to machine learning are less susceptible to overfitting, this research applied clustering ensembles that were generated using bagging.

A third issue in intrusion detection is adversarial evasion. To avoid detection, attackers may attack the intrusion detection system itself. Attacks against intrusion detection systems include evasion, tampering, and denial of service (Laskov & Lippmann, 2010; Šrndić & Laskov, 2014; Wang et al., 2014; Vasilomanolakis, Karuppayah, Mühlhäuser, & Fischer, 2015). Evasion amplifies the problem of overfitting by preventing the detection of novel attacks (Sommer & Paxson, 2010). To address this problem, this research used ensemble approaches, since they have been found effective in resisting adversarial evasion (Šrndić & Laskov, 2014; Wang et al., 2014). In addition, this research applied active learning, which has been shown to allow adaptation to evasive techniques (Miller et al., 2014).

A fourth issue in intrusion detection research is the use of a suitable dataset for evaluation. Most early intrusion detection research used the KDD Cup 1999 dataset, which was prepared from a DARPA packet capture for a KDD competition (Cao, Hoang, Nguyen, 2013). The KDD Cup 1999 dataset has been criticized because of its duplicate records, its outdated information because of older technologies, and its high volume of records (Qian, Xu, & Shi, 2006; Tavallae et al., 2009; Creech & Hu, 2013). To address this problem, a variety of researchers have created new intrusion detection evaluation datasets. None of the newer datasets has received as much widespread popularity and adoption as KDD Cup 1999. This research included a literature review of publicly available intrusion detection datasets. As a result, the NSL-KDD dataset was used for preliminary evaluation and the UNSW-NB15 dataset for evaluation with a more contemporary dataset (Tavallae et al., 2009; Moustafa & Slay, 2015).

## **Definition of Terms**

**Accuracy:** The ratio of the number of true positive and true negative results compared to the number of all results.

**Active Learning:** A semi-supervised machine learning approach in which a subset of unknown data is selected and presented to an oracle for labeling during training.

**Alert:** A notification or warning, which in the context of intrusion detection, represents a potential cyberattack.

**Anomaly:** An event that deviates from a normal event.

**Attack:** An aggressive action against a computer system that may include unauthorized modification of files to allow unauthorized access to system information, unauthorized access or modification of user information, unauthorized modification of information in network components, or unauthorized use of system resources, including unauthorized account creation (Chebrolu, Abraham, & Thomas, 2005).

**Bagging:** A machine learning method that provides a random subset of features and records to a machine learning algorithm with a goal of reducing overfitting.

**Bayes Theorem:** Methods of probabilistic inference that calculate a prior probability of a hypothesis based on evidence gathered from observations (Kruegel et al., 2003).

**Bayesian Networks:** Directed acyclic graphs, where each node in the graph represents a conditional probability table (Kruegel et al., 2003).

**Clustering:** Finding natural groupings in data such that data within each cluster is most similar to other data in that cluster and most dissimilar to the data of other clusters (Zhou & Tang, 2006; Jain, 2010).

**Clustering Ensemble:** An approach to clustering that uses combinations of multiple partitions of clustering results to find a consensus partition that improves accuracy compared to an individual clustering result (Topchy, Jain, & Punch, 2005; Ayad & Kamel, 2010).

**Cyber Situation Awareness:** A specialized application of situation awareness that applies to the analysis of cyberattacks and their impact on computer and network operations (Li et al., 2010; Erbacher et al., 2010).

**Cyberattack:** See Attack.

**Ensemble:** A multi-learner system, where each component learner attempts to solve the same task as the others (Strehl & Ghosh, 2002).

**Evasion:** Deceiving an intrusion detection system or rendering it ineffective through the modification of training or testing data (Šrندیć & Laskov, 2014; Wang et al., 2014; Laskov & Lippman, 2010).

**False Positive:** In the context of intrusion detection, a condition in which a predicted class indicates an attack, but the actual class was normal.

**Intrusion Detection:** The practice of examining information from computers and networks that are to be protected in order to identify attacks against those computers and networks (Debar, Dacier, & Wespi, 1999).

**KDD Cup 1999:** A dataset that has been widely used in intrusion detection research.

**Machine Learning:** A field of computer science that uses algorithms to learn patterns without being programmed with predefined rules.

**Mirkin Distance:** An algorithm for comparing two clusters by counting the number of point pairs that are exclusive to each of the two clusters (Meilă, 2007).

**Oracle:** An entity that represents a knowledgeable human subject matter expert and is able to provide the correct label for a data record in response to a query.

**Overfitting:** A modeling error in which a machine learning algorithm is trained to match a particular set of data but is not generalizable to other sets of data.

**Partition:** A set of clusters that represent the results from a single clustering algorithm.

**Security Analyst:** A job function that specializes in cyber defensive operations in the context of a business or organization.

**Security Monitoring:** A job function of security analysts for detecting and responding to potential cyberattacks.

**Signature:** A predefined pattern that matches characteristics of attacks.

**Situation Awareness:** The perception of information in an environment for a given time and space, the comprehension of the meaning of that information, and the projection of the future conditions in order to enable effective selection of an appropriate course of action (Endsley, 1995; Tadda & Salerno, 2010).

**Supervised Machine Learning:** An approach to machine learning in which records with known classes are used to train an algorithm so that it can then predict an output for records with unknown classes.

**Unsupervised Machine Learning:** An approach to machine learning in which no expected outcome is provided for training an algorithm.

**List of Acronyms**

**ANMI:** Average Normalized Mutual Information

**CSA:** Cyber Situation Awareness

**CSPA:** Cluster-Based Similarity Partitioning Algorithm

**CSV:** Comma Separated Value

**CTBN:** Continuous Time Bayesian Networks

**CVSS:** Common Vulnerability Scoring System

**DARPA:** Defense Advanced Research Projects Agency

**DOS:** Denial of Service

**EM:** Expectation Maximization

**FN:** False Negative

**FP:** False Positive

**GPU:** Graphics Processing Unit

**HGPA:** Hypergraph Partitioning Algorithm

**IDS:** Intrusion Detection System

**KDD:** Knowledge Discovery in Databases

**LAC:** Locally Adaptive Clustering

**LAN:** Local Area Network

**MCLA:** Meta Clustering Algorithm

**NM:** Normalized Mutual Information

**NMF:** Nonnegative Matrix Factorization

**QMI:** Quadratic Mutual Information

**ROC:** Receiver Operating Characteristics

**SCANN:** Stacking, Correspondence Analysis and Nearest Neighbor

**SVM:** Support Vector Machine

**TN:** True Negative

**TP:** True Positive

**WBPA:** Weighty Bipartite Partition Algorithm

**WSBPA:** Weighted Subspace Bipartite Partitioning Algorithm

**WSPA:** Weighty Similarity Partition Algorithm

## **Summary**

This chapter introduced an approach to anomaly-based intrusion detection using clustering ensembles. This chapter described how this research approached the problems of too much data and high false positives rates. Finally, this chapter established the goal of shifting the focus of intrusion detection from individual events to higher level of meaning. This research explored how to predict the probability that a computer system was under attack to enable security analysts to make better decisions under uncertain conditions.

The remainder of this dissertation report provides the supporting background for this research and describes the methodology that was used. Chapter 2 reviews the literature that established the basis for this research. Chapter 3 describes the experiments conducted to test the effectiveness of clustering ensembles with probabilistic analysis for intrusion detection as well as the testing approaches for evaluating the results. Chapter 4 presents the results of the experiments. Finally, Chapter 5 explores the conclusions of this

research, the implications for future research, and recommendations for building upon the foundation laid by this research.



## **Chapter 2**

### **Review of the Literature**

#### **Overview of Reviewed Topics**

The focus of this research was to develop an intrusion detection system that provides meaning for intrusion alerts by using clustering ensembles and probabilistic analysis. To accomplish this research, the following areas of literature were examined to synthesize these concepts:

- Cyber Situation Awareness (CSA)
- Intrusion Detection
- Probabilistic Intrusion Detection
- Clustering Ensembles
- Intrusion Detection Datasets

Each of these areas has an established body of existing research. The following sections describe the importance of each of these areas, review relevant research studies in each topic, and synthesize the key concepts needed for building a foundation for this research.

## **Cyber Situation Awareness (CSA)**

### *Background*

Situation awareness is the perception of information in an environment for a given time and space. It comprehends the meaning of information and projects the future conditions to enable the effective selection of a course of action (Endsley, 1995; Tadda & Salerno, 2010). Situation awareness is an essential function in various fields that require the interpretation of information about the environment for effective decision-making (Endsley, 1995). The goals of situation awareness are to identify what is happening, why it is happening, what will happen next, and what can be done about it (Erbacher et al., 2010). Seminal work in situation awareness was conducted by Endsley (1995) and resulted in a theoretical model for use in discussion and future research. Endsley (1995) developed a three-level model of situation awareness. At the first level, situation awareness deals with perceiving the elements of the current situation. After the elements are collected, the second level entails comprehension of the current situation. This second level includes correlation and integration of data to achieve a higher level of understanding (Erbacher et al., 2010). Finally, the third level focuses on projecting a future state and the potential impact on future operations (Erbacher et al., 2010). Decision-making regarding a course of action occurs after all three levels are developed to some extent by analysts.

CSA is a specialized application of situation awareness that applies to a first-level analysis of cyberattacks and their impact on computer and network operations (Li et al., 2010; Erbacher et al., 2010). Improving CSA increases the effectiveness of security analysts in dealing with attacks (Brynielsson, Frank, & Varga, 2016). CSA involves the

interpretation of raw security events to identify malicious actors, legitimate users, and system abnormalities in the context of system operating conditions and known vulnerabilities (Erbacher et al., 2010).

### *Review of CSA Literature*

Endsley (1995) introduced the theory of situation awareness to address the problem that there was no underlying theory that supported moving from discrete observations to a comprehension of the overall situation. As a result, Endsley (1995) defined a model for situation awareness to support future discussion and research. The model contains three levels of situation awareness that are driven by goals, objectives, and preconceived ideas: Level 1 deals with perceiving the elements of the current situation; Level 2 entails comprehension of the current situation; and Level 3 focuses on projecting a future state.

Ehrbacher et al. (2010) built upon Endsley's (1995) research by applying the model to CSA. Using cognitive task analysis, Ehrbacher et al. (2010) addressed the need for improved decision making for security analysts. The results of this study found that CSA includes impact identification, damage assessment, recovery, projection to the future, as well as characterization of attacks and attackers (Erbacher et al., 2020). To uncover the collaborative processes for threat analysts, Ahrend, Jirotko, and Jones (2016) conducted interviews with threat analysts on their day-to-day practices. This study found that too much data leads to decisions made with uncertainty and that information is critical for reducing uncertainty at all stages of CSA (Ahrend et al., 2016). In addition, Ahrend et al. (2016) found that analysts rely upon what they remember from their own

past investigations. As a result, biases from past incidents may incorrectly inform future decisions when evaluating alerts.

Gutzwiller, Hunt, and Lange (2016) used cognitive task analysis for studying CSA but focused on determining the goals and information elements needed to make decisions. The results demonstrated that CSA requires abstraction at several levels: 1) the network as well as its architecture and state; 2) the world, including emergent threats, abnormal behaviors, and attack signatures; and 3) the team, with a focus on how teams work and how they hand-off work to each other (Gutzwiller et al., 2016). Similarly, Newcomb, Hammell, and Hutchinson (2016) found that high levels of abstraction were necessary to enable decision making in their experimental study that addressed the problem of too many intrusion alerts for analysts to evaluate. Bartnes, Moe, and Heegaard (2016) studied CSA with a goal of improving security incident response. Their study used semi-structured interviews and found that, since there is an absence of major events during normal operating conditions as well as a low priority for training, security analysts are not prepared for incidents when they occur (Bartnes et al., 2016).

Rajivan and Cooke (2017) explored team-level CSA, including human collaboration and information sharing. Using constructs of shared mental models, transactive memory, and interactive team cognition, this study used a combination of cognitive task analysis and event analysis of systemic teamwork (EAST) to empirically test the results (Rajivan & Cooke, 2017). This study found that the role of teamwork is important for CSA at every level of cybersecurity defense (Rajivan & Cooke, 2017). Zhong et al. (2017) captured the cognitive processes of security analysts involved in triaging security alerts. This study created a framework for retrieving data that is relevant

to the triage process to provide context for alerts (Zhong et al., 2017). In addition, this study created a system with a user interface that automatically identified the information that analysts required for decision-making (Zhong et al., 2017).

Table 1 summarizes literature from empirical CSA studies. Results from these studies are important in characterizing the meaning of intrusion alerts in the context of CSA.

Table 1

*Summary of CSA Literature*

<b>Study</b>	<b>Problem</b>	<b>Methodology</b>	<b>Findings or Contributions</b>
Erbacher et al. (2010)	Need for improved decision making for security analysts	Cognitive task analysis	CSA goals include impact identification, damage assessment, recovery, projection to the future, and characterization of attacks and attackers.
Ahrend et al. (2016)	Uncover collaborative processes for threat analysts	Interviews with threat analysts on day-to-day practices	Too much data leads to making decisions under uncertainty. Information is critical to reducing uncertainty at all stages. Analysts rely upon what they remember from their own past investigations.
Gutzwiller et al. (2016)	Determine the goals and information elements needed for CSA	Cognitive task analysis	Abstraction for CSA includes: 1) the network and its architecture and state; 2) the world, including emergent threats, abnormal behaviors, and attack signatures; 3) and the team, with a focus on how the team works and hands-off work to each other.

Table 1

*Summary of CSA Literature (cont.)*

<b>Study</b>	<b>Problem</b>	<b>Methodology</b>	<b>Findings or Contributions</b>
Newcomb et al. (2016)	Too many intrusion alerts for security analysts to evaluate	Experimental study	CSA requires a high level of abstraction to enable decision-making. CVSS scores are not a good indicator of vulnerabilities for CSA.
Bartnes, Moe, & Heegaard (2016)	How to improve security incident response	Semi-structured interviews	Absence of major events prevents preparation for security incidents, and training for security incidents is not a priority in organizations.
Rajivan & Cooke (2017)	Understanding the role of teamwork in CSA	Cognitive task analysis and event analysis of systemic teamwork	Teamwork is important for improving CSA at every level of cybersecurity defense processes. Limitations in teamwork have a detrimental impact on defense.

### *Discussion*

CSA must consider the network topology (Brynielsson et al., 2016), vulnerabilities (Erbacher et al., 2010), cyber personas (Brynielsson et al., 2016), and the current threat landscape (Gutzwiller et al., 2016). In addition, CSA must include a time component that considers near real-time events, mid-term events, and long-term events (Brynielsson et al., 2016). Taking all of this information and the various time views into account, CSA requires a high level of abstraction that enables human decision-making (Ergacher et al., 2010; Gutzwiller et al., 2016; Newcomb et al., 2016).

In addition to technical security monitoring systems, CSA relies upon a variety of techniques for understanding current and projected conditions. These techniques include: timelines of attacks (Erbacher et al., 2010), attack trees (Li et al., 2010), kill chains

(Bhatt, Yano, Amorim, & Gustavsson, 2014), Bayesian networks (Franke & Brynielsson, 2014), and the diamond model (Al-Mohannadi et al., 2016). CSA requires both statistical techniques for understanding events and human knowledge to learn about novel attacks (Tadda & Salerno, 2010).

CSA is a challenging practice for a variety of reasons. Most current work in cybersecurity monitoring focuses on single, isolated attacks and does not develop a full comprehension of the current situation or the projected state (Tadda & Salerno, 2010). The volume of computer and network data and alerts makes it impossible for security analysts to know the detailed operation of each computer in a network (Li et al., 2010). CSA relies upon uncertain, imperfect information (Li et al., 2010). As new information becomes available, security analysts must update their existing beliefs to address the uncertainty (Tadda & Salerno, 2010). In addition, CSA is challenging because, under normal conditions, there is an absence of major security incidents. As a result, security analysts often lack preparation, training, and documentation to support CSA when cyberattacks occur (Bartnes et al., 2016).

## **Intrusion Detection**

### *Background*

Intrusion detection is the practice of examining information from computers and networks to identify attacks (Debar et al., 1999). Intrusion detection identifies anomalies that represent computer network intrusions or uses signature-based approaches that detect patterns that match known intrusion techniques (Mukherjee, Heberlein, & Levitt, 1994). An important factor in detecting intrusions is deciding the source of data that will be monitored. Intrusion detection systems generally perform either network-based or host-based intrusion detection (Mukherjee et al., 1994). A variety of attacks may be found by intrusion detection systems, including: 1) unauthorized modification of files to allow unauthorized access to system information; 2) unauthorized access or modification of user information; 3) unauthorized modification of information in network components; and 4) unauthorized use of system resources, including unauthorized account creation (Chebrolu et al., 2005).

Intrusion detection systems are divided into signature-based detection and anomaly detection. Signature-based systems rely upon predefined patterns that match the characteristics of attacks. Anomaly detection learns normal patterns and detects patterns that have not been encountered or predefined. Both signature-based detection and anomaly detection have advantages and disadvantages (Chebrolu et al., 2005). Intrusion detection is also categorized by the means in which data is collected as either network-based or host-based. Xiao, Chen, and Chang (2014) provide a more comprehensive listing of types of intrusion detection systems, including: 1) network-based, 2) host-based, 3) stack-based, 4) protocol-based, and 5) graph based.



Table 2

*Summary of Machine Learning Approaches to Intrusion Detection*

<b>School of Thought</b>	<b>Algorithms</b>	<b>Research Studies</b>
Symbolist	Decision trees, random forests	Sinclair, Pierce, & Matzner (1999); Zhang, Zulkernine, & Haque (2008); Sindhu, Geetha, & Kannan (2012); McElwee (2017)
Connectionist	Neural networks, self-organizing maps, deep neural networks	Cannady (1998); Rhodes, Mahaffey, & Cannady (2000); Stopel, Boger, Moskovitch, Shahar, & Elovici (2006); Ahmad, Abdullah, & Alghamdi (2009); Daliran, Nassiri, & Latif-Shabgahi (2010); Sindhu et al. (2012); McElwee & Cannady (2016); McElwee et al. (2017)
Evolutionary	Immune system, evolutionary neural networks, genetic programming	Dasgupta & González (2002); Han & Cho (2005); Song, Heywood, & Zincir-Heywood (2005); Toosi & Kahani (2007); Sindhu et al. (2012)
Bayesian	Bayesian networks, naïve Bayes	Valdes & Skinner (2000); Kruegel et al. (2003); Feng, Guan, Guo, Gao, & Liu (2004); Gowadia, Farkas, & Valtorta (2005); Tylman (2008); Perdisci, Ariu, Fogla, Giacinto, & Lee (2009); Xu & Shelton (2010); Koc, Mazzuchi, & Sarkani (2012); Yassin, Udzir, Muda, & Sulaiman (2013)
Analogistic	Support vector machine (SVM)	Mukkamala, Janoski, & Sung (2002); Chen, Hsu, & Shen (2005); Tsang, Kwok, & Cheung (2005); Khan, Awad, & Thuraisingham (2007)

Machine learning approaches can be divided into five primary schools of thought, as shown in Table 2: symbolist, connectionist, evolutionary, Bayesian, and analogistic (Domingos, 2015, p. 239). The symbolist approach uses inverse deduction with approaches like decision trees. The connectionist approach commonly applies neural networks and back propagation (Domingos, 2015, p. 239). Evolutionary machine learning

was inspired by Alan Turing (1950) and uses principles of biological evolution like random populations, mutation, and survival of the fittest. Bayesians, on the other hand, rely on probabilistic inference, as developed by Thomas Bayes and formalized by Pierre-Simon Laplace (McGrayne, 2014, p. 159). Finally, the analogistic approach relies on the similarity between objects and uses tools like support vector machines (Domingos, 2015, p. 239).

### *Review of Literature on Machine Learning for Intrusion Detection*

Pioneering work in intrusion detection began with Denning and Neumann (1985) and was further developed by Denning (1987). This work sought to detect abnormal patterns of system behaviors that may represent security violations. This initial work resulted in an expert system that was expanded upon by Lunt (1990) and was subsequently developed into network-based intrusion detection (Heberlein et al., 1990). The remainder of this subsection reviews literature that applied machine learning to intrusion detection, excluding Bayesian approaches, which are covered in the next section. This literature review is presented in chronological order, but in most cases, the literature does not build upon previous research. Instead each applies different types and combinations of machine learning to the general problem of intrusion detection pioneered by Denning and Neumann (1985).

Cannady (1998) conducted pioneering work in the application of machine learning to intrusion detection. This study applied neural networks to supervised misuse detection and achieved a detection accuracy of 97.5% for testing data. Sinclair et al. (1999) used both symbolist and evolutionary approaches by combining decision trees and genetic algorithms. This study found that, when combined, these two methods were

useful for defining network connection rules that could be used to create expert systems (Sinclair et al., 1999). Applying neural networks in a novel approach, Rhodes et al. (2000) applied Kohonen self-organizing maps to unsupervised intrusion detection. This study found that training self-organizing maps based on normal operating conditions allowed the unsupervised detection of buffer overflow attacks, which had not been present in the training data (Rhodes et al., 2000). Using evolutionary approaches, Dasgupta and González (2002) used an immune system model for intrusion detection and found that positive characterization was more precise than negative characterization but required more resources.

Mukkamala et al. (2002) applied SVMs to intrusion detection and evaluated their effectiveness using the KDD Cup 1999 dataset. Their study compared the results of SVMs to neural networks and found that the accuracy was comparable, but SVMs were limited by their binary output (Mukkamala et al., 2002).

Julisch and Dacier (2002) and Julisch (2003) studied how to cluster intrusion detection system alerts for root cause analysis. These studies found that by iteratively identifying alerts that could be categorized as low criticality or false positives, clustering reduced the quantity of alerts that required review by human analysts (Julisch & Dacier, 2002; Julisch, 2003).

Chen et al. (2005) used SVMs and compared their accuracy to neural networks. Although Mukkamala et al. (2002) found comparable results between SVMs and neural networks, Chen et al. (2005) found that SVMs outperformed the neural networks in terms of detection accuracy.

Han and Cho (2005) applied evolutionary neural networks to intrusion detection using the IDEVAL dataset. This study found that using an evolutionary algorithm for defining the structure of the neural network reduced the required training time and improved detection accuracy (Han & Cho, 2005). Also using an evolutionary approach, Song et al. (2005) used genetic programming and applied it to the KDD Cup 1999 dataset. They found that hierarchical genetic programming was successful in detecting previously unseen attacks (Song et al., 2005).

Using an analogistic approach, Tsang et al. (2005) used SVMs for intrusion detection. They introduced a new algorithm, called a Core Vector Machine, which reduced the computational complexity of the training process (Tsang et al., 2005). Stopel et al. (2006) compared neural networks,  $k$ -nearest neighbor, and decision trees to detect computer worms. They found that both neural networks and  $k$ -nearest neighbor had similar accuracy and that neural networks performed classification faster than  $k$ -nearest neighbor (Stopel et al., 2006). Neural networks are generally slower to train than other algorithms because of many iterations of back propagation, so their conclusion that neural networks were faster to train is surprising.

Several researchers have combined multiple machine learning methods for detecting intrusions. As an example, Khan et al. (2007) combined hierarchical clustering with support vector machines and found that it improved the overall accuracy. Similarly, Toosi and Kahani (2007) combined soft computing methods for classification with a genetic algorithm for fuzzy inference and found that this combination successfully detected normal events and denial of service attacks. Zhang et al. (2008) combined multiple approaches by applying random forests simultaneously to misuse and anomaly

detection. They found that the combination had a higher detection rate and a lower false positive rate than either independent approach (Zhang et al., 2008).

Hu, Hu, Xie, and Maybank (2009) found that hierarchical graph-theoretic clustering was effective in active learning for intrusion detection. Ahmad et al. (2009) focused on detecting denial of service attacks using neural networks. They found that neural networks were capable of very high detection rates, except for teardown attacks (Ahmad et al., 2009). Daliran et al. (2010) used neural networks for intrusion detection but used them for detection of malicious code in a honeypot environment. They found that neural networks achieved 80% accuracy when using labeled data from this environment (Daliran et al., 2010).

Sindhu et al. (2012) combined three different schools of thought by applying genetic algorithms, neural networks, and decision trees simultaneously. This study used the genetic algorithm for feature selection, followed by a neural network for preprocessing the data (Sindhu et al., 2012). Finally, this study used a decision tree to classify the data and found that it had a higher detection rate than using either a neural network or a C4.5 classifier independently (Sindhu et al., 2012).

Clustering is another machine learning approach that has been applied to intrusion detection. Dubey and Dubey (2015) used clustering to preprocess data for machine learning for intrusion detection. Li, Kao, Zhang, Chuang, and Yen (2015) also used clustering but applied it to network flow data, which was effective in detecting botnet activity. Silva and Hruschka (2016) found that SLS-IBkM clustering was effective for data streams, including the KDD Cup 1999 dataset. Expanding upon the work of Rhodes et al. (2000) in using self-organizing maps for intrusion detection, McElwee and Cannady

(2016) focused on preprocessing intrusion detection data for imbalanced datasets. They found that filtering normal events and using Principal Component Analysis for feature reduction prior to training Kohonen self-organizing maps improved the detection accuracy and reduced the clustering time (McElwee & Cannady, 2016).

McElwee (2017) applied active learning to random forest classification by beginning with unlabeled data and presenting a subset of the data to an oracle for labeling. This study found that by using  $k$ -means clustering to select a sample for the oracle, 90% the KDD Cup 1999 records could be classified accurately by manually labeling 0.13% of the total records (McElwee, 2017). Finally, expanding on the approach of Julisch (2003) in categorizing alerts, McElwee et al. (2017) applied deep neural networks to classifying alerts from a signature-based intrusion detection system and found that they were highly accurate for categorizing alerts. This approach was successful for assigning alerts to the appropriate security analysts as well as for automating routine reporting tasks (McElwee, Heaton, Fraley, & Cannady, 2017).

### *Discussion*

Regardless of the approach used, intrusion detection systems can be evaluated using several different measures. Koc et al. (2012) highlight important evaluation criteria, including accuracy, error rate, and the area under receiver operating characteristics (ROC) curve. For binary classification, Figure 3 shows the possible outcomes of intrusion detection (Koc et al., 2012).

		Predicted Class	
		Normal	Attack
Actual Class	Normal	True Negative (TN)	False Positive (FP)
	Attack	False Negative (FN)	True Positive (TP)

Figure 3. Confusion matrix for binary classification

Using the criteria shown in Figure 3, it follows that accuracy can be defined as the number of true positives (TP) and true negatives (TN) divided by all possible outcomes:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Further, it follows that, given the accuracy, the error rate can be described as:

$$Error\ Rate = 1 - Accuracy \quad (2)$$

Intrusion detection is an area of research that faces several on-going challenges. One of the most important research challenges is the reduction of false positives. Most false positives are generated because of under-specified signatures, intent guessing, and limited abstraction capability (Gowadia et al., 2005). To address the problem of false positives, some studies have focused on identifying the attackers rather than including all events across all computers and users (Burroughs, Wilson, & Cybenko, 2002). In addition, intrusions are rare and may be hidden in massive amounts of data (Scott, 2004). As a result, the prior probability of an attack is very low, so even a highly accurate intrusion detection system will have high false positive rates (Kruegel et al., 2003).

Another challenge in intrusion detection is the large amount of data and the repetitive work of analyzing and prioritizing the events. Sawyer et al. (2014) found that after prolonged monitoring activity, there was a noticeable drop in the accuracy of security analysts. Thus, for routine monitoring of intrusion detection alerts, solutions that require less human interaction are beneficial. The amount of data is voluminous and contains redundant features, which also makes detection difficult for intrusion detection systems because of imbalanced and irrelevant data (Chebrolu et al., 2005).

Finally, intrusion detection systems may be considered high value targets for attackers. Intrusion detection systems are subject to adversarial evasion, including deceiving the intrusion detection system or rendering it ineffective (Šrندیć & Laskov, 2014; Wang et al., 2014). Laskov and Lippman (2010) found that adversaries may attempt to modify the training or testing data to alter the detection results. As a result, intrusion detection must utilize robust classifiers capable of dealing with attacks and uncertainty.



## **Probabilistic Intrusion Detection**

### *Background*

Probabilistic methods for intrusion detection apply statistical formulas and algorithms. Bayes theorem refers to methods of probabilistic inference that were initially developed by Thomas Bayes and later formalized by Pierre-Simon Laplace (McGrayne, 2014, p. 159). The purpose of Bayes theorem and Bayesian networks is to allow the calculation of a prior probability of a hypothesis based on evidence gathered from observations (Kruegel et al., 2003). Bayesian updating is the process of estimating the probability of a hypothesis given that an event has been observed (Kruegel et al., 2003; Chivers, Clark, Nobles, Shaikh, & Chen, 2013). Bayesian approaches also allow the combination of information from several sources (Scott, 2004; Chivers et al., 2013).

Bayesian methods are preferable over frequency-based statistics for intrusion detection, since frequency-based methods are more prone to evasion (Swarnkar & Hubballi, 2016). In addition, Bayesian methods provide a simple way to include prior information (Scott, 2004) and allow knowledge representation that enables reasoning with uncertain information (Chebrolu et al., 2005). Probabilistic methods, like Bayes theorem, allow intrusions to be detected based on soft evidence, or beliefs, rather than hard evidence (Gowadia et al., 2005). This allows attacks to be presented as probabilities rather than as binary decisions (Gowadia et al., 2005).

Probabilistic methods help to address the challenges of intrusion detection, in that there are massive amounts of data, but criminal intrusions are rare (Scott, 2004). This data imbalance contributes to the base rate fallacy, which shows that even a highly

accurate intrusion detection system will have a high false positive rate (Kruegel et al., 2003). In formal terms:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (3)$$

Thus, if the prior probability of an actual intrusion event,  $P(B)$ , is extremely low, it will force  $P(B|A)$  to be low, making detection more challenging. Perdisci et al. (2009) found that a false positive rate for imbalanced data must therefore be  $10^{-5}$  or lower. As a result, detecting intrusions for rare attacks is more challenging than for common attacks, since one of the goals of intrusion detection is to keep false alarms low (Pajouh, Dastghaibyfar, & Hashemi, 2017).

One probabilistic approach is the use of Bayesian networks, which are directed acyclic graphs, where each node in the graph represents a conditional probability table (Kruegel et al., 2003). Another probabilistic approach is naïve Bayes, which is the simplest Bayesian classifier and performs well for datasets that have conditional independence of their features (Koc et al., 2012). A variation of naïve Bayes is hidden naïve Bayes, which creates an additional layer to represent a hidden parent for each node (Koc et al., 2012).

Bayesian approaches show promise for intrusion detection, but there are many challenges as well. One problem with Bayesian networks is their node ordering requirement, which may require expert knowledge to develop (Chebrolu et al., 2005) or the use of an additional algorithm that can estimate the node ordering (Gowadia et al., 2005). Another challenge with some Bayesian approaches is the computational complexity, especially for Type 2 algorithms, where the order is not given, which exhibit  $O(N^4)$  complexity (Chebrolu et al., 2005). Bayesian networks are also known to have

suboptimal models that require large training datasets, which may not be available in intrusion detection for rare attacks, thus requiring additional methods to alleviate this constraint (Xiao et al., 2014).

### *Review of Probabilistic Intrusion Detection Literature*

Helman and Liepins (1993) provided a foundational paper for statistical-based intrusion detection that other researchers have built upon. Valdes and Skinner (2000) addressed the shortcomings of anomaly and signature-based detection by modeling attacks as hypothesis and using events to adapt probabilities. This study found that combining signature-based and anomaly-based intrusion detection using probabilities improved the detection of distributed attacks (Valdes & Skinner, 2000).

Bayesian networks have been applied to intrusion detection in a variety of research studies. Kruegel et al. (2003) used full Bayesian networks to model interdependencies of events and found a reduction in false alerts. Feng et al. (2004) found that dynamic Bayesian networks for recognizing time-varying plans were effective for predicting normal and anomalous call sequences. Gowadia et al. (2005) used agent graphs and Bayesian networks to evaluate beliefs, rather than hard values. This study found that Bayesian networks can be created by asking experts to create directed acyclic graphs either manually or by using an algorithm (Gowadia et al., 2005). Tylman (2008) combined misuse and anomaly detection methods, similar to Valdes and Skinner (2000). This combined approach used Bayesian classification of Snort alerts at the session level and found that it uncovered the structure of belief networks (Tylman, 2008).

Perdisci et al. (2009) applied Bayesian methods to combine the results of an ensemble of support vector machine classifiers. The goal of this research was to reduce false positives, and the results demonstrated high accuracy, especially for shellcode and polymorphic attacks (Perdisci et al., 2009). Xu and Shelton (2010) used continuous time Bayesian networks (CTBN) for anomaly detection. This application of Bayesian networks focused purely on event timing and outperformed existing methods for anomaly detection (Xu & Shelton, 2010).

Koc et al. (2012) used hidden naïve Bayes for improving the accuracy of intrusion detection. This study found that hidden naïve Bayes outperformed naïve Bayes for accuracy and error rate while maintaining the simplicity of naïve Bayes (Koc et al., 2012). Chivers et al. (2013) focused specifically on the problem of detecting insider attacks. This research study combined sources using hypotheses and Bayesian updating and found that updating beliefs based on evidence is effective in detecting attacker nodes (Chivers et al., 2013).

Yassin et al. (2013) used  $k$ -means clustering to separate data and subsequently used Bayes classification. Their study found that using clustering as an initial step significantly reduced the false positive rate and increased the true negative rate (Yassin et al., 2013). Xiao et al. (2014) addressed the shortcoming of Bayesian networks using an ensemble approach. Bayesian network model averaging selects the best network from a set of trained networks and performs better than regular Bayesian networks or naïve Bayes (Xiao et al., 2014). Bayesian network model averaging requires less data for training, so it is effective for smaller training data sets (Xiao et al., 2014).

Swarnkar and Hubballi (2016) used multinomial Bayesian one-class classifiers of  $n$ -grams with probability trees to address the shortcomings of frequency-based and one-class classifiers. Although this approach had a high detection rate, it was accompanied with a moderately high false positive rate and high computational complexity (Swarnkar & Hubballi, 2016). As a result, this approach is not suitable for intrusion detection, since moderately high false positive rates for highly imbalanced datasets results in much lower overall accuracy (Kruegel et al., 2003). To address the problem of high computational complexity and the problem of imbalanced datasets, Pajouh et al. (2017) used two-tier classification with linear discriminant analysis. This study found that linear discriminant analysis provided optimal feature reduction and made naïve Bayes more efficient for classification (Pajouh et al., 2017).

### *Discussion*

Several important themes can be established from previous research. First, intrusion detection using probabilistic methods should avoid generalization across an entire dataset, since it makes it easier for criminals to hide (Scott, 2004). For example, data can be segmented to look at anomalies per user (Scott, 2004; Dash, Reddy, & Pujari, 2011) or per host (Burroughs et al., 2002; Chivers et al., 2013). Second, time is an important element that should be considered, with a preference to more recent behaviors (Scott, 2004; Xu & Shelton, 2010; Chivers et al., 2013). Third, categorical data has been shown to perform better than parametric or continuous data for probabilistic methods for detecting intrusions (Scott, 2004). Fourth, several previous studies found that

probabilistic methods were successful in combining information from several sources, which is useful for aggregating event-level information to create a more abstract level.

## **Clustering Ensembles**

### *Background*

The purpose of clustering is to understand natural groupings in data (Jain, 2010). Clustering algorithms divide data into clusters, such that the data within each cluster is most similar to other data in the cluster, and the data between clusters is most dissimilar to that of other clusters (Zhou & Tang, 2006). Clusters in data appear in various shapes, sizes, sparseness, and degrees of separation (Fred & Jain, 2005). Clustering identifies natural structures in data when the structure, the number of clusters, or shapes of the clusters may be unknown (Dimitriadou, Weingessel, & Hornik, 2001). Thus, clustering is a primary technique for unsupervised machine learning (Zhou & Tang, 2006). In addition to finding natural groupings and structure in data, clustering can perform natural classification or compression of data into cluster prototypes (Jain, 2010).

A wide variety of clustering algorithms exist, but most can be placed into four categories. First, iterative square-error partitional clustering, such as  $k$ -means clustering, finds a distance between centroids and data elements and does not impose a structure on the data (Frossyniotis, Likas, & Stafylopatis, 2004; Jain, 2010). Since iterative square-error partitional clustering uses a distance measure, it creates hyperspherical clusters and does not identify novel cluster shapes (Fred & Jain, 2005). Second, hierarchical clustering organizes data into nested sequences of groups that can be visualized as trees (Frossyniotis et al., 2004). Third, density-based clustering finds the densest regions of the feature space that are separated by low density space (Jain, 2010). Fourth, grid-based

clustering uses spatial data mining techniques to subdivide a hyperspace into sections that represent clusters (Frossyniotis et al., 2004).

Clustering is considered a difficult problem (Jain, 2010). As a form of unsupervised learning, it is hard to select a clustering method in advance that can identify the same clusters that match those identified by a human expert (Ayad & Kamel, 2008). Clustering results for different clustering algorithms may be very different using the same data (Jain, 2010). Further, different clustering runs using the same clustering algorithm can have different results because of different initialization parameters (Dimitriadou et al., 2001).

There is no best clustering algorithm (Jain, 2010). No single algorithm exists that can identify all of the cluster shapes and structures (Fred & Jain, 2005). No clustering method is available that will find the correct underlying structure for all data sets (Vega-Pons & Ruiz-Shulcloper, 2011, p. 337). As a result, researchers have found it is best to use several different clustering algorithms on a given data set and see what works best (Fred & Jain, 2005). Clustering algorithms are generally optimization problems that reduce mean-square error, minimize some other type of error, or use similarity functions (Dimitriadou et al., 2001). The quality of clustering can be evaluated using *R*-squared, intra-over inter-variation quotient, BD-index, and SD validity index (Frossyniotis et al., 2004).

Ensemble approaches have been applied to address many of the challenges associated with clustering. An ensemble is a multi-learner system, where each component learner attempts to solve the same task as the others (Strehl & Ghosh, 2002). Clustering ensembles were the result of research in multiple classifier systems (Dimitriadou et al.,

2001; Hadjitodorov, Kuncheva, & Todorova, 2006). The goal of clustering ensembles is to find a combination of multiple partitions that improves the overall clustering of the data (Topchy et al., 2005). As a result, clustering ensembles find a consensus partition that improves that accuracy of individual clustering results (Ayad & Kamel, 2010).

Different clustering algorithms produce different clustering results, and different runs of the same algorithm provide different results, because of different initialization parameters. Thus, using multiple clustering approaches simultaneously helps to find the best clustering solution (Frossyniotis et al., 2004; Jain, 2010). The benefit of clustering ensembles is that the decision of a group may be more reliable than that of any individual (Vega-Pons & Ruiz-Shulcloper, 2011, p. 338). As a result, clustering ensembles reduce the risk of picking the wrong clustering method for a given dataset (Hadjitodorov et al., 2006).

Previous research has applied clustering ensembles to face recognition, character recognition, scientific image analysis, and medical diagnosis (Zhou & Tang, 2006). Research has also evaluated clustering ensembles with large datasets and has found an improvement in clustering, even for incomplete partitions (Lourenco et al., 2015). Clustering ensembles can find the right number of clusters in data (Dimitriadou et al., 2001; Ayad & Kamel, 2010). They also improve the quality and robustness of clustering (Strehl & Ghosh, 2002; Topchy et al., 2005; Vega-Pons & Ruiz-Shulcloper, 2011, p. 365; Lourenco et al., 2015). Clustering ensembles identify hidden structures in data (Bakker & Heskes, 2003) and find clusters of arbitrary and complex shapes (Dimitriadou et al., 2001; Frossyniotis et al., 2004; Hadjitodorov et al., 2006). As a result, they enable new insights into a dataset and lower the prediction error (Bakker & Heskes, 2003). Because



of their diversity of clustering solutions, clustering ensembles, are more generalizable than individual clustering algorithms (Zhou & Tang, 2006). Finally, clustering ensembles can be implemented in a distributed computing environment, allowing them to scale well for large ensembles (Strehl & Ghosh, 2002).

Clustering ensembles can be viewed as a two-step process. The first step is to generate the various clustering results, also known as partitions, and the second step is to evaluate the results using a consensus function (Topchy, Law, Jain, & Fred, 2004; Vega-Pons & Ruiz-Shulcloper, 2011, p. 338). Figure 4 shows how these two steps fit into the overall concept of clustering ensembles and the notations that are used in this paper, which have been adapted from Vega-Pons and Ruiz-Shulcloper (2011, p. 339).

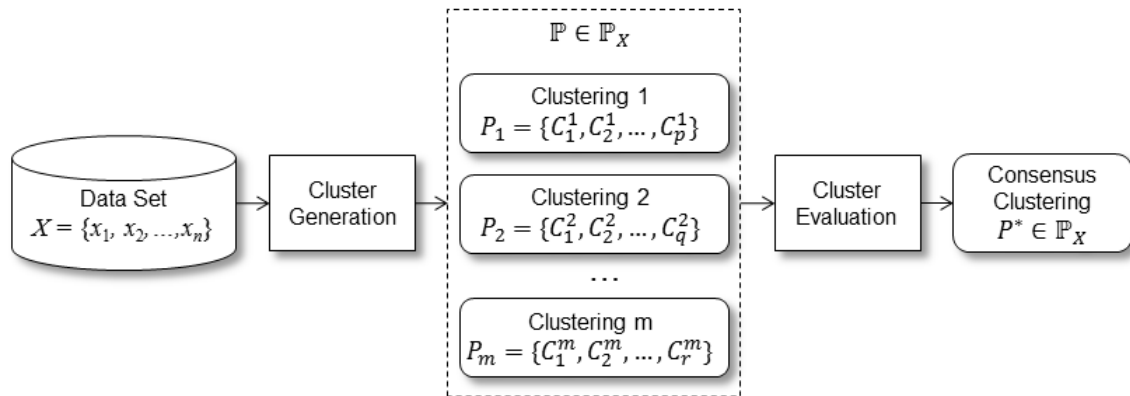


Figure 4. Overview clustering ensembles with formal notation

In this notation,  $X = \{x_1, x_2, \dots, x_n\}$  is a set of objects in which each  $x_i$  is a multi-dimensional tuple of  $\alpha$  dimensions and where  $i = 1 \dots n$ .  $\mathbb{P}_X$  is the set of all possible partitions in  $X$ , and  $\mathbb{P} = \{P_1, P_2, \dots, P_m\}$  is the set of partitions generated by the cluster generation process, such that  $\mathbb{P} \in \mathbb{P}_X$ . As a result, each partition,  $P_i \in \mathbb{P}$ , is made

up of multiple symbolic cluster labels,  $C_j^i$ , such that  $P_i = \{C_1^i, C_2^i, \dots, C_d^i\}$  is a set of objects in  $X$  with  $d$  clusters. In Figure 4, each partition can have varying numbers of clusters, so the final element in the set is represented as  $C_p^1$ ,  $C_q^2$ , and  $C_r^m$ , where  $m = |\mathbb{P}|$ . The variables  $p$ ,  $q$ , and  $r$ , are arbitrary numbers of clusters produced in each partition, such that  $C_j^i$  represents the  $j$ th cluster in  $i^{\text{th}}$  partition,  $P_i$ . The objective of clustering ensembles is to find a consensus partition by evaluating the partitions,  $P_i \in \mathbb{P}$ , to find a clustering result,  $P^* \in \mathbb{P}_X$ , where  $P^*$  is a better clustering solution than any  $P_i \in \mathbb{P}$ .

The cluster generation step creates multiple partitions,  $\mathbb{P} \in \mathbb{P}_X$ , that can be used to find the final clustering solution,  $P^* \in \mathbb{P}_X$ . There are few constraints on how the clusters are generated (Vega-Pons & Ruiz-Shulcloper, 2011, p. 340). The most important requirement of cluster generation is that multiple, diverse partitions are created (Strehl & Ghosh, 2002). Without diverse partitions, clustering ensembles will not be able to outperform single clustering algorithms (Hadjitodorov et al., 2006). The initial partitions can be thought of as noisy versions of true partitions (Topchy et al., 2004). As a result, weak and less computationally expensive clustering algorithms can be used in the initial cluster generation with comparable or better results than an individual clustering algorithm (Topchy et al., 2005).

A variety of approaches exist for creating a diverse initial set of partitions. One approach for creating diverse partitions is to use different clustering algorithms (Hadjitodorov et al., 2006; Jain, 2010; Vega-Pons & Ruiz-Shulcloper, 2011). For example, Dimitriadou et al. (2001) implemented  $k$ -means and a competitive learning algorithm to generate diverse initial partitions. Another approach for creating diverse partitions is to use different initialization parameters with the same clustering algorithm

(Hadjitodorov et al., 2006; Jain, 2010; Vega-Pons & Ruiz-Shulcloper, 2011). This is commonly implemented using the  $k$ -means algorithm, since it has a variety of initial parameters, is not computationally expensive, and is very popular (Jain, 2010).

Another widely used approach for generating diverse initial partitions is to use different representations of the data (Jain, 2010). The different representations of the data may include different subsets of features or a different subset of the objects in the dataset (Fred & Jain, 2005; Hadjitodorov et al., 2006; Vega-Pons & Ruiz-Shulcloper, 2011). Different representations of the data may also include projecting the data to different feature spaces (Fred & Jain, 2005), using algorithms like Principal Component Analysis (Strehl & Gosh, 2002). Examples of these approaches include random feature selection (Strehl & Gosh, 2002), bootstrapping (Bakker & Heskes, 2003), boosting (Frossyniotis et al., 2004), and bagging (Fred & Jain, 2005).

Bakker and Heskes (2003) found that bootstrapping allows local summaries that are not possible for a single model and reduces model bias by creating a more complex model. In addition, bootstrapping reduces the tendency to overfit, since the models are not trained on the full data (Bakker & Heskes, 2003). Frossyniotis et al. (2004) found that boosting a clustering algorithm for a few iterations provided better results than running the algorithm several times and choosing the best run.

The cluster evaluation step is one of the most difficult challenges in clustering ensembles (Vega-Pons & Ruiz-Shulcloper, 2011, p. 341). For ensembles of classifiers, there are labels, which make the evaluation step a straightforward problem. For clustering ensembles, there are no labels, which results in a correspondence problem (Dimitriadou et al., 2001; Strehl & Ghosh, 2002; Frossyniotis et al., 2004). Thus, cluster evaluation

combines diverse partitions, without labeled data, to find the true underlying partition that represent the natural organization of the data (Topchy et al., 2004; Tumer & Agogino, 2008).

### *Review of Clustering Ensemble Literature*

The earliest clustering research began with Dimitriadou et al. (2001). This seminal work addressed the problem that there was no clear way to combine results from different clustering algorithms to find a clear partition (Dimitriadou et al., 2001). Using voting, Dimitriadou et al. (2001) found that clustering ensembles were effective in finding fuzzy partitions and in finding the right number of clusters. Although this study was the earliest identified research on clustering ensembles, most subsequent work in clustering ensembles built upon the work of Strehl and Ghosh (2002). Addressing the same problem, Strehl and Ghosh (2002) created three new clustering algorithms: 1) cluster-based similarity partitioning, 2) hypergraph partitioning, and 3) meta clustering. They found that consensus clustering using ensembles using any of their heuristic algorithms was better than individual clustering results (Strehl & Ghosh, 2002).

Bakker and Heskes (2003) contributed to ensemble-based approaches by applying bootstrapping. They found that using bootstrapping to summarize large ensembles into smaller numbers of representative models reduced overfitting, provided better prediction, and detected hidden structures in the data (Bakker & Heskes, 2003). Frossyniotis et al. (2004) addressed the combination of multiple clustering solutions by using sequential clustering with boosting and found that boost clustering improved the quality and performance of clustering (Frossyniotis et al., 2004).

Another important thread of literature emanated from Topchy et al. (2004). Their research acknowledged that clustering ensembles work but addressed the problem that there was no theoretical basis for why they worked (Topchy et al., 2004). This study used both stochastic and mean partition generation and found that consensus solutions converge to a true clustering solution as the number of partitions in the ensemble increases (Topchy et al., 2004). They also found that the probability of not discovering the true partition decreases exponentially as the number of partitions in the ensembles increases (Topchy et al., 2004). Continuing this work, Fred and Jain (2005) addressed the problem of identifying all cluster shapes and structures by using evidence accumulation. As part of this research, they proposed a theoretical framework and provided criteria for analysis of the combination of clustering results (Fred & Jain, 2005). Topchy et al. (2005) examined how to combine partitions using a consensus function by using expectancy maximization and mutual information consensus functions. They found that weak partitions may be used in clustering ensembles and still achieve comparable or better performance than single clustering approaches (Topchy et al., 2005).

Hadjitodorov et al. (2006) explored how to select partitions in clustering ensembles by using a diversity measure. Using the Adjusted Rand Index, they found that ensembles with a wide spread of individual diversity were better than ensembles with less spread and that medium diversity clusters were the best approach (Hadjitodorov et al., 2006). Zhou and Tang (2006) compared voting, weighted voting, selective weighting, and selective weighted voting and found that selective weighted voting was significantly better for cluster evaluation. Tumer and Agogino (2008) further compared a meta

clustering algorithm to voting active clusters with reinforcement learning and found that meta clustering was better.

Not all previous research focused on the approaches for combining clustering results. Ayad and Kamel (2008) focused on reducing the computing complexity by beginning with the idea that consensus clustering has  $O(n^2)$  complexity. By applying cumulative voting for identifying clustering solutions, this study improved accuracy and reduced computational complexity to  $O(n)$  (Ayad & Kamel, 2008).

Azimi and Fern (2009) explored conflicting results regarding diversity of clustering ensembles. By using adaptive clustering ensemble selection, this study concluded that selection must be adaptive to accommodate the datasets, since no approach worked consistently for all of the datasets that were studied (Azimi & Fern, 2009). Ayad and Kamel (2010) applied cumulative voting as a special case of linear regression for finding the optimum labeling of clustering ensembles. This study found that cumulative voting improved the accuracy and stability of results, as well provided an accurate estimation of the number of clusters (Ayad & Kamel, 2010). Lourenco et al. (2015) examined the problem that the clustering correspondence does not reflect uncertainty. This study used a probabilistic interpretation of Evidence Accumulation Clustering by using Bregman divergence and resulted in improved clustering, even for incomplete partitions and large datasets (Lourenco et al., 2015).

*Review of Co-occurrence Consensus Function Literature*

Co-occurrence consensus functions analyze the number of times objects belong to clusters as well as the number of times two objects belong to the same clusters. Co-occurrence functions include: 1) relabeling and voting, 2) co-association matrix, 3) graph and hypergraph, 4) Locally Adaptive Clustering (LAC) algorithms, 5) fuzzy methods, 6) information theoretic methods, and 7) finite mixture models (Vega-Pons & Ruiz-Shulcloper, 2011, p. 353). Table 3 summarizes the co-occurrence consensus evaluation methods using these categories.

Table 3

*Summary of Co-Occurrence Cluster Evaluation Approaches*

<b>Method</b>	<b>Description</b>	<b>Studies</b>
Relabeling and Voting	Voting process after solving labeling correspondence problem	Dimitriadou et al. (2001); Zhou & Tang (2006); Ayad & Kamel (2008); Tumer & Agogino (2008)
Co-Association Matrix	Cluster results into an intermediate co-association matrix	Fred & Jain (2005); Wang et al. (2009)
Graph and Hypergraph	Create graphs and hypergraphs of partitions and evaluate for consensus partition	Strehl & Ghosh (2002); Fern & Brodley (2004)
Locally Adaptive Clustering (LAC)	Evaluate centroids and weights for numerical data	Domeniconi & Al-Razgan (2009)
Fuzzy Methods	Evaluate clusters as soft partitions rather than hard partitions	Frossyniotis et al. (2004); Punera & Ghosh (2008); Ayad & Kamel (2010)

Table 3

*Summary of Co-Occurrence Cluster Evaluation Approaches (cont.)*

<b>Method</b>	<b>Description</b>	<b>Studies</b>
Information Theoretic	Minimize entropy within partitions	Strehl & Ghosh (2002); Punera & Ghosh (2008); Jain (2010)
Finite Mixture Models	Probabilistic modeling of subpopulations using a mixture distribution	Topchy et al. (2005)

Among the relabeling and voting approaches, Dimitriadou et al. (2001) used voting, based upon classification ensembles, followed by a merging procedure. Zhou and Tang (2006) measured similarity by counting overlap within clusters and found that selective weighted voting was the best of the approaches they evaluated. Ayad and Kamel (2008) developed a relabeling and voting approach that used cumulative voting. Tumer and Agogino (2008) used voting active clusters with reinforcement learning and used average normalized mutual information (ANMI) as an objective function.

Co-association matrix approaches map clustering results into an intermediate representation, called a co-association matrix (Vega-Pons & Ruiz-Shulcloper, 2011, p. 346). For example, Fred and Jain (2005) split the data into a large number of small spherical clusters, using  $k$ -means clustering. Next, they combined the small clusters using a similarity matrix. Because of this intermediate step, co-association approaches have a complexity of  $O(n^2)$  and are limited to smaller data sets. Wang, Yang, and Zhou (2009) introduced probabilistic methods using a co-association matrix and introduced Bayesian clustering ensembles.



Cluster evaluation approaches that use graph and hypergraph methods transform the partitions into a graph or hypergraph and cut the graph to obtain a consensus partition (Vega-Pons & Ruiz-Shulcloper, 2011, p. 347). Many of the foundational approaches to cluster evaluation were developed by Strehl and Ghosh (2002), who developed three different heuristics to evaluate hypergraphs: 1) cluster-based similarity partitioning algorithm (CSPA), 2) hypergraph partitioning algorithm (HGPA), and 3) meta clustering algorithm (MCLA). Similarly, Fern and Brodley (2004) used a graph partitioning approach. This approach did not actually solve for normalized mutual information in clusters, but instead acted more like a co-occurrence evaluation method (Vega-Pons & Ruiz-Shulcloper, 2011, p. 349).

LAC algorithms identify partitions within numerical features as two sets of information: 1) the centroids identified in the clusters, and 2) their associated weights (Vega-Pons & Ruiz-Shulcloper, 2011, p. 355). Domeniconi and Al-Razgan (2009) developed three consensus functions using this approach: 1) Weighty Similarity Partition Algorithm (WSPA), 2) Weighty Bipartite Partition Algorithm (WBPA), and 3) Weighted Subspace Bipartite Partitioning Algorithm (WSBPA). These algorithms were limited in their use to numerical data and require that the number of clusters be specified initially (Vega-Pons & Ruiz-Shulcloper, 2011, p. 355).

Fuzzy methods rely on the soft nature of clustering approaches and recognize that there may be “fuzzy” partitions in the data (Dimitriadou et al., 2001). Some clustering algorithms that may be used in the cluster generation stage, such as fuzzy c-means and EM, already produce soft clustering results (Frossyniotis et al., 2004; Vega-Pons & Ruiz-Shulcloper, 2011, p. 360). As a result, cluster evaluation methods that use fuzzy methods

do not attempt to convert the initial soft-clustering results into hard clusters. Some fuzzy consensus algorithms, such as voting, can perform soft or hard clustering (Ayad & Kamel, 2010).

Information theoretic approaches minimize entropy within groupings (Jain, 2010). Strehl and Ghosh (2002) used concepts from information theory and focused on normalized mutual information (NMI) and average normalized mutual information (ANMI) as objective functions. Punera and Ghosh (2008) used soft base clustering and used an information theoretic approach.

Topchy et al. (2005) used a fusion method with probabilities and based their solution to the consensus problem on a finite mixture model. The result was two new consensus functions, called quadratic mutual information (QMI) and expectation maximization (EM), that eliminated the need to solve the label correspondence problem (Topchy et al., 2005). This approach required that the data be modeled as random, independent variables and requires a fixed number of clusters in the final clustering solution (Vega-Pons & Ruiz-Shulcloper, 2011, p. 353).

#### *Review of Median Partition Cluster Evaluation Literature*

Median partition-based approaches are optimization problems that maximize similarity or minimize dissimilarity and can be divided into the following categories: 1) genetic algorithms, 2) nonnegative matrix factorization (NMF) methods, 3) kernel models, and 4) Mirkin distance (Vega-Pons & Ruiz-Shulcloper, 2011, p. 350). Table 4 summarizes median partition-based cluster evaluation approaches using these categories.

Table 4

*Summary of Median Partition Cluster Evaluation Approaches*

<b>Method</b>	<b>Description</b>	<b>Studies</b>
Genetic Algorithms	Utilize search capabilities of genetic algorithms to minimize or maximize distance functions	Yoon, Ahn, Lee, Cho, & Kim (2006); Luo, Jing, & Xie (2006); Analoui & Sadighian (2006)
Nonnegative Matrix Factorization (NMF)	Find factors and dissimilarity using a nonnegative matrix	Li, Ding, & Jordan (2007)
Kernel Models	Similarity measure between solutions with approximation	Vega-Pons, Correa-Morris, & Ruiz Schulcloper (2008, 2010)
Mirkin Distance	Counting pairs of points within clusters and using a symmetric distance metric	Gionis, Mannila, & Tsaparas (2007)

Genetic algorithm approaches rely upon the search capabilities of genetic algorithms and use the highest fitness value, after some stopping criterion is reached (Vega-Pons & Ruiz-Shulcloper, 2011, p. 354). What distinguishes genetic algorithm approaches is the type of fitness function employed. One approach, called heterogeneous clustering ensembles, used ordered pairs with a population generation mechanism and a fitness function to evaluate the number of overlaps between partitions within each pair (Yoon et al., 2006). Another approach used an information theoretic fitness function that minimized entropy between partitions (Luo et al., 2006). Yet another approach implemented the fitness function as a maximization of probability using a finite mixture method (Analoui & Sadighian, 2006). A challenge in using genetic algorithms for cluster evaluation is that different runs may produce different results because of the heuristic

nature of these algorithms (Vega-Pons & Ruiz-Shulcloper, 2011, p. 354). It may also be inferred that such algorithms may settle on local minima or maxima.

NMF methods for cluster evaluation have been used to create a median partition using a non-negative matrix by finding factors of the matrix and a dissimilarity measure between partitions (Li et al., 2007; Vega-Pons & Ruiz-Shulcloper, 2011, p. 357). Kernel models have been used to create a median partition using a similarity measure between partitions and find an approximate solution (Vega-Pons et al., 2010).

Mirkin distance approaches have been used to create a median partition by counting pairs. Using this approach, given two clusters,  $C$  and  $C'$ , where  $N_{01}$  is the number of point pairs in  $C'$  but not in  $C$ , and  $N_{10}$  is the number of point pairs in the same partition in  $C$  but not in  $C'$ , then the Mirkin distance for comparing two clusters is  $\mathcal{M}(C, C') = 2(N_{01} + N_{10})$  (Meilä, 2007). When applied to clustering ensembles, the objective is to minimize the Mirkin distance between the partitions, and a number of heuristic approaches approximate this function (Gionis et al. 2007, Vega-Pons & Ruiz-Shulcloper, 2011, p. 342).

### *Discussion*

A number of design criteria should be considered when using clustering ensembles. In cluster generation, one important consideration is the number of partitions generated in the first stage. Dimitriadou et al. (2001) found through experimentation that large numbers of clusters provided the best clustering solution. Topchy et al. (2004) found that clustering ensembles converged more closely to a true clustering solution as the number of partitions in the ensemble increased. The probability of not discovering the

true partition decreases exponentially as the number of partitions in the ensemble increases (Topchy et al., 2004). Another important consideration in the clustering generation step is the diversity of partitions generated. There are conflicting results related to the diversity of initial partitions (Azimi & Fern, 2009). Ensembles with a wide spread of individual diversity are better than ensembles with less spread, but spread did not relate to accuracy (Hadjitodorov et al., 2006). As a result, medium diversity clusters were found to be the best approach (Hadjitodorov et al., 2006).

In the cluster evaluation stage, there are several design criteria to consider. One important consideration is the objective function for determining the quality of clustering, such as a similarity or dissimilarity function (Zhou & Tang, 2006; Jain, 2010; Vega-Pons & Ruiz-Shulcloper, 2011). Another important consideration is the stopping criterion for determining when the best clustering solution has been identified (Dimitriadou et al., 2001).

Finally, several cluster evaluation strategies are computationally complex. Most consensus clustering algorithms have  $O(n^2)$  complexity, although some have achieved  $O(n)$  complexity (Ayad & Kamel, 2008). Thus, the selection of a cluster evaluation strategy should be based on the volume of data and the computational cost of the algorithm.

## **Intrusion Detection Datasets**

### *Background*

Creating datasets for intrusion detection system evaluation is subjective, and new datasets face several challenges. One of the primary challenges to obtaining realistic intrusion data is privacy. This is because data from operational networks is the most realistic but is considered confidential by most network operators (Orfila, Tapiador, & Ribagorda, 2009). One approach to solving privacy issues is to use simulations, but these can be unrealistic (Orfila et al., 2009). Another approach to solving privacy concerns is to create datasets in test beds, but if they are too simple, they will lack realism as well (Milenkoski, Vieira, Kounev, Avritzer, & Payne, 2015).

Another significant challenge in the creation of intrusion datasets is the labeling of normal and attack data (Orfila et al., 2009). One approach to labeling attacks is to use penetration testing to develop the attack data, but this has been criticized for producing unrealistic datasets (Wheelus, Khoshgoftaar, Zuech, & Najafabadi, 2014; Milenkoski et al., 2015). Another approach is to use honeypots for collection of attack data, but since honeypots contain mostly attack data, this too can be considered unrealistic (Milenkoski et al., 2015). Other approaches develop traces of normal network conditions and separate traces of attacks (Salem, Reissmann, & Buehler, 2014). Still other approaches use combinations of operational network data, penetration testing data, and simulation to create a more diverse, complex dataset for intrusion detection testing (Shiravi, Shiravi, Tavallaei, & Ghorbani, 2012; Moustafa & Slay, 2015; Singh, Kumar, & Singla, 2015; Haider, Hu, Slay, Turnbull, & Xie, 2017).

Other research on intrusion detection datasets includes methods and software tools that aid in the creation of intrusion datasets. For example, Shiravi et al. (2012) created software agents for generating normal network activity and attack activity in test beds. Salem et al. (2014) developed the OptiFilter toolkit, which can be deployed in large networks to create continuous datasets for intrusion detection evaluation.

Vasilomanolakis, Cordero, Milanov, and Mühlhäuser (2016) developed the ID2T toolkit for injecting synthetic attacks into real network packet capture data. Lin, Lin, Wang, Chen, and Lai (2016) developed PCAPLib to automatically extract, classify, and anonymize packet capture data.

#### *Review of Intrusion Detection Dataset Literature*

The KDD Cup 1999 intrusion detection dataset set a standard for many years for evaluating intrusion detection approaches. The source data for this dataset came from DARPA's MIT Lincoln Labs collection of network packet information called IDEVAL (Cao et al., 2013). The KDD Cup 1999 dataset was prepared by Stolfo, Fan, Lee, Prodromidis, and Chan (2000) and was based upon the IDEVAL network packet data, which contained seven weeks of network traffic. The KDD Cup 1999 dataset was specifically prepared for the KDD competition (Cao et al., 2013). The KDD Cup 1999 training dataset includes approximately 4.9 million connection records and is labeled as either normal or with a specific attack vector (Tavallaee et al., 2009). The attacks in the KDD Cup 1999 dataset fall into four categories: 1) denial of service attack, 2) user to root attacks, 3) remote to local attacks, and 4) probing attacks (Tavallaee et al., 2009).

Over time, the KDD Cup 1999 dataset has been criticized by researchers for three primary reasons. First, the characteristics of networks have changed since KDD Cup 1999 was created (Qian et al., 2006). For example, the KDD Cup 1999 dataset was collected on the Solaris operating system, used older applications and operating system approaches, and represented a time when attacks generally were against only a single system process at one time (Creech & Hu, 2013). Second, the KDD Cup 1999 dataset contains 78% duplicate records in the training dataset and 75% duplication in the testing dataset, which may lead to problems of overfitting (Tavallae et al., 2009). Third, the number of records in the KDD Cup 1999 dataset are too numerous, so many researchers use only subsets of these datasets. This leads to an inconsistent basis for comparison between intrusion detection systems (Tavallae et al., 2009).

In response to the criticisms of the KDD Cup 1999 dataset, a number of researchers have proposed alternative datasets. Table 5 provides a summary of many notable intrusion detection datasets that have been created as potential alternatives to the KDD Cup 1999 dataset. Most notable among these are Gure KDD Cup (Perona et al. 2008), NSL-KDD (Tavallae et al., 2009), MAWILab (Fontugne, Borgnat, Abry, Fukuda, 2010), ADFA-LD12 (Creech & Hu, 2013), UNSW-NB15 (Moustafa & Slay, 2015), and NGIDS-DS (Haider et al., 2017).



Table 5

*Summary of Intrusion Detection Dataset Research*

<b>Study</b>	<b>Data Type</b>	<b>Approach</b>	<b>Contributions</b>
Stolfo et al. (2000)	Network packet data	Extraction of features from DARPA 1998 dataset	Produced the KDD Cup 1999 dataset. Found that fraud detection can be generalized to intrusion detection.
Qian et al., (2006)	Network packets and audit logs	Simulation based on university laboratory LAN	Feasibility of creating synthetic IDS testing data. Data is more useful at user level than at packet level.
Perona et al. (2008)	Network packed data	Combined KDD Cup 1999 dataset with DARPA 1998 payload data	Produced Gure KDD Cup dataset. Including payload and header data improves detection.
Tavallaee et al. (2009)	Network and host data	Subset of KDD-Cup 1999 dataset to remove redundancy and duplicates	Created NSL-KDD dataset, which is an improved distribution of data for IDS testing. Still suffers from unrealistic network data.
Fontugne et al. (2010)	Network data	Combined anomaly detection results from MAWI archive by using SCANN	Created MAWILab dataset, which is updated daily. Labeled as Anomalous, Suspicious, Notice, and Benign.
Gogoi, Bhuyan, Bhattacharyya, & Kalita (2012)	Network packet and flow data	Captured from test bed with over 350 nodes with automated attacks	Produced TUIDS DDoS dataset.
Shiravi et al. (2012)	Network packet data	Test bed with 21 workstations using $\alpha$ and $\beta$ profiles to generate dynamic test data	Established criteria for evaluating datasets: realistic network, realistic traffic, labeled dataset, total interaction capture, complete capture, and diverse attacks.

Table 5

*Summary of Intrusion Detection Dataset Research (Cont.)*

<b>Study</b>	<b>Data Type</b>	<b>Approach</b>	<b>Contributions</b>
Cao et al. (2013)	Network packet data	Test bed similar to that used for DARPA 1998 data	Produced LUT13 dataset. Better in comparison to the KDD Cup 1999 dataset for generalizable detection.
Creech & Hu (2013)	Audit log data	Single Ubuntu Linux server with common software applications with normal and attack traces	Produced ADFA-LD12 dataset. Evaluation shows this dataset has more complexity than KDD Cup 1999.
Wheelus et al. (2014)	Network packet and flow data	Collected data from internet service provider and manually labeled	Produced SANTA dataset, which features realistic normal traffic, penetration testing traffic, real attacks, and modern attack types.
Moustafa & Slay (2015)	Network packet data	Synthetic generation using IXIA Perfect Storm hardware	Produced synthetically realistic, labeled dataset called UNSW-NB15.
Singh et al. (2015)	Network and host data	Statistical approach to generate a new dataset using NSL-KDD as a base dataset	Created Panjab University Intrusion Data Set (PU-IDS).
Haider et al. (2017)	Network packet and audit log data	Synthetic generation using IXIA Perfect Storm hardware	Produced synthetically realistic, labeled dataset called NGIDS-DS. Developed evaluation criteria for intrusion datasets.

*Discussion*

At present, there is not an accepted standard for evaluating the quality and usefulness of intrusion detection datasets. Milenkoski et al. (2015) surveyed intrusion detection research literature and developed a method of categorizing intrusion detection

evaluation. Their work was inconclusive in offering a standard for evaluation, but it did provide an overview of the complexities inherent in this area of research. Most recently, Haider et al. (2017) developed evaluation criteria that consists of six factors: 1) completeness of capture of audit logs and network packets, 2) inclusion of maximum possible attacks, 3) representative of current attack behaviors, 4) inclusive of real world normal traffic that includes realistic timing and complexity, 5) capture of system maintenance activity that occurs in real operational networks, and 6) ground truth labeling to represent normal traffic. Since it is a more recent study the Haider et al. (2017) evaluation criteria may become an accepted standard, but this will require time and a critical analysis by future researchers.

Since its inception, the UNSW-NB15 dataset has been gaining adoption from researchers. It is commonly used alongside other datasets. For example, Bamakan, Wang, and Shi (2017) applied both the UNSW-NB15 and NSL-KDD datasets to multi-class intrusion detection using Ramp Loss  $K$ -Support Vector Classification-Regression. Kamarudin, Maple, Watson, and Safa (2017) used both UNSW-NB15 and NSL-KDD to test an ensemble classifier used for anomaly detection. Hajisalem and Babaie (2018) used both the UNSW-NB15 and the NSL-KDD datasets to test a new hybrid intrusion detection system using artificial bee colony and artificial fish swarm algorithms. Papamartzivanos, Mármol, and Kambourakis (2018) used UNSW-NB15, NSL-KDD, and KDD Cup 1999 for rule induction for intrusion detection. With these recent studies in mind, it has become an accepted practice to use both the NSL-KDD and UNSW-NB15 datasets to test intrusion detection algorithms.

## Summary

This chapter reviewed and synthesized relevant research in the areas of CSA, intrusion detection, probabilistic intrusion detection, clustering ensembles, and intrusion detection datasets. CSA uses observations of individual events in the broader context of the situation at hand, with a goal of predicting future states. Intrusion detection is a method for detecting attacks based upon either signatures of known attacks or the identification of anomalies. Intrusion detection has contributed to the first level of CSA, in that it provides individual events for evaluating the situation under conditions of uncertainty. Probabilistic intrusion detection methods have been effective in intrusion detection, which provides support for the approach in this research. Clustering ensembles provide multiple perspectives of a dataset and are effective in detecting patterns and anomalies in data without prior knowledge of the structures of the data.

In addition, this chapter provided a detailed overview of intrusion detection datasets. It presented the challenges associated with selecting and generating such datasets for research. Several criteria should be considered to ensure the suitability of datasets for evaluating intrusion detection systems. Datasets should be complete and be representative of current attack behaviors with suitable complexity to reflect operational networks. Further, they should either be labeled with specific attack types or include a ground truth labeling to separate normal from anomalous data. Using both the UNSW-NB15 and NSL-KDD datasets to test intrusion detection systems has become a common practice in research.

## **Chapter 3**

### **Methodology**

#### **Introduction**

This chapter describes the approach that was used for implementing and evaluating the effectiveness of the experiments. This research used clustering ensembles, bagging, probability analysis, and active learning. The result was the probability that a computer system was being attacked, based on the event-level observation of anomalies that were identified using clustering ensembles. The resulting solution not only provided unsupervised intrusion detection but also, by incorporating active learning, allowed a level of human interaction by subject matter experts with domain knowledge. This solution evolved through three experimental stages.

In the first experiment, an algorithm for cluster generation with bagging was developed. This experiment evaluated and compared different existing cluster generation parameters to determine their suitability for deriving meaning from intrusion detection datasets. The second experiment developed an algorithm for probabilistic anomaly detection, using the clustering ensemble results. The third and final experiment incorporated active learning to allow domain knowledge from subject matter experts.

## Solution Design

A multi-stage algorithm was developed that generated a diverse set of initial partitions, evaluated the results of the initial clustering to detect anomalies, and processed the output, while incorporating subject matter expert feedback. Figure 5 summarizes the high-level design of the solution and includes important design considerations that are described in more detail in this chapter.

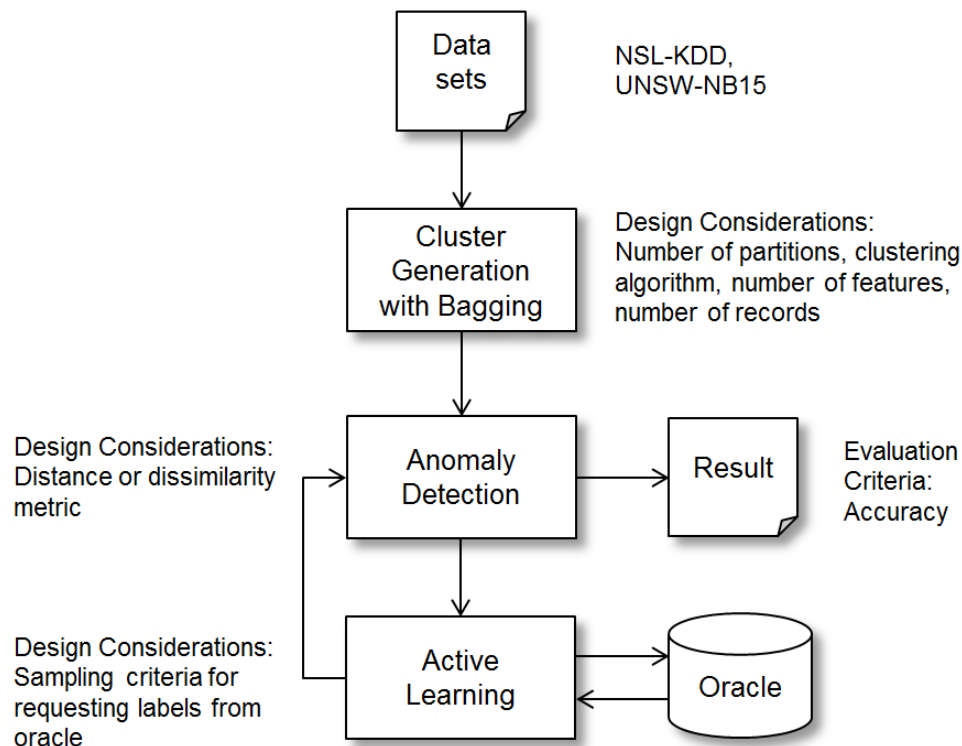


Figure 5. High level solution design

## Dataset

The selection of a dataset for evaluating intrusion detection systems is important for allowing algorithms and approaches to be compared to each other (Milenkoski et al., 2015). Prior research has found that intrusion detection datasets should be realistic, publicly available, and provide ground truth data (Shiravi et al., 2012; Haider et al.,

2017). For this research, it was also important that the dataset identified the specific computer systems that were experiencing normal or attack activity.

Two datasets were used in evaluating this solution. First, the NSL-KDD dataset was used as a general dataset that connected this research with prior studies. Although this dataset is outdated, it provided a basis of comparison with a dataset that lacks the complexity of more contemporary intrusion datasets. Since the NSL-KDD dataset does not identify specific computer systems, it could only be used to evaluate the clustering ensemble approaches and preliminary anomaly detection at the event level. It was not used to evaluate the probability that a specific computer system was being attacked.

The second dataset that was be used throughout this research was the UNSW-NB15 dataset, which was created by Moustafa and Slay (2015). One of the most important features of the UNSW-NB15 dataset was that it identified specific computer systems with a source IP address and a destination IP address. This allowed the evaluation of the probability that a specific computer system was under attack. This dataset is more current than the NSL-KDD dataset, and so it reflects more current attacks. It is also a more complex dataset for intrusion detection, which means that it is more difficult to detect attacks, and better tested the capabilities of the algorithm.

### *Cluster Generation with Bagging*

The cluster generation stage of the solution was evaluated using a variety of criteria. First, an appropriate clustering algorithm was needed that had reasonable computational complexity and that was well-suited for the dataset. It was expected that  $k$ -means clustering would be suitable for this clustering. The need for diversity in the

clustering results was accomplished using bagging, which selected random features from the dataset to provide diversity. As a result, it was not expected that this solution would require diverse algorithms.

Another important criteria in developing the cluster generation stage was the number of partitions to generate. Previous studies found that sufficient partitions are needed to provide diversity, but they also found that a medium diversity approach performed better than a larger diversity solution (Azimi & Fern, 2009). This was important for evaluation to identify the number of clusters that provide the optimal diversity.

### *Anomaly Detection*

The anomaly detection stage contained two important algorithms. The first algorithm evaluated the clustering ensemble results to find anomalies. This algorithm differed from other clustering ensemble approaches in that its objective was not to find common clusters, but instead to find the anomalies. This overcame the clustering correlation problem, which arises from the lack of labels, thus simplifying the problem of cluster evaluation.

This algorithm first evaluated each partition to determine which clustering labels represented anomalies. To accomplish this, the algorithm used counting and statistical analysis of the clusters to determine which of the clusters in the partition were inconsistent with the others. It was expected that anomalous clusters could be detected as follows:



$$C_{anomaly} = \begin{cases} 0, & \mu_c - K\sigma_c \leq |C_i| \leq \mu_c + K\sigma_c \\ 1, & (|C_i| < \mu_c - K\sigma_c) \cup (|C_i| > \mu_c + K\sigma_c) \end{cases} \quad (4)$$

In this case,  $|C_i|$  is the number of events identified in cluster,  $C_i$ . This number of records was compared to the mean of the number of events in all of the clusters,  $\mu_c$ , minus a constant,  $K$ , number of standard deviations,  $\sigma_c$ . Thus, a cluster would be considered an anomaly when the number of records in it is outside of the interval of  $K$  standard deviations from the mean number of records in each cluster. The value for  $K$  was determined experimentally to find an appropriate threshold. This algorithm was tested and updated based on observations in the data.

Next, the algorithm evaluated each event,  $E$ , in the dataset to determine the probability that the event was an anomaly,  $P(E_{anomaly})$ . This evaluation would be based on the number of clusters to which it was assigned that were considered anomalies divided by the total number of clusters. It was expected that event-level anomalies could be detected as:

$$P(E_{anomaly}) = \frac{1}{n} \sum_{i=1}^n C_{anomaly} \quad (5)$$

The result,  $P(E_{anomaly})$ , represents the probability that an event is an anomaly, based on the number of partitions that found it was an anomaly. If this probability needed to be converted into a binary result, a threshold probability would be selected to determine when the probability represents an anomaly. Instead, the probability was preserved as a soft metric and was passed to the second algorithm. This algorithm also was tested and updated based on observations in the data.

The second algorithm in the anomaly detection stage determined the probability that a computer system was being attacked, based on the observation of one or more anomalies. Given the probability that a computer system was being attacked as  $P(A)$  and the probability that an individual event is an anomaly,  $P(E)$ , this solution assumed that prior to any observations, the probability that a computer was under attack would be uncertain, thus:

$$P(A) = 0.5 \quad (6)$$

Further, given the initial probability that the computer system was under attack and an observation of an anomalous event, the probability of an event being an attack, given that a computer system being attacked, would be:

$$P(E|A) = P(A) \times P(E) \quad (7)$$

Following the observation of an event, it would be reasonable to update the prior probability to reflect the greater certainty, based on the probability of the observation. Therefore, given a number of observations,  $N$ , it was expected that the probability that a computer system is being attacked would be:

$$P(A) = P(A) + \frac{1}{N} \sum_{i=1}^N P(E_i|A) - 0.5 \quad (8)$$

The probability portion of the algorithm was developed based on these assumptions and was modified, as needed, through evaluation of the experimental results.

### *Active Learning*

The active learning portion of this solution was built upon previous approaches developed by McElwee (2017). Active learning is successful in separating normal from attack traffic using minimal labeling (McElwee, 2017). Areas for improvement include more detailed evaluation of the sampling, as well as the use of an improved oracle that allows the detection of certain rare attacks (McElwee, 2017).

In addition to sampling the events to be sent to the oracle for labeling, this stage of the algorithm determined how to use this feedback to influence the outcome of the anomaly detection. The two most likely opportunities evaluated were to override the probability that an event was indeed anomalous,  $P(E)$ , or to use the oracle's response to update the prior probability that the computer system was under attack,  $P(A)$ . The approach for active learning was finalized during the implementation and testing of the third experiment.

### **Experiment 1: Cluster Generation**

The first experiment evaluated cluster generation strategies. This experiment focused on the  $k$ -means clustering algorithm and used bagging to generate a variety of clustering results. This experiment first required the implementation of a cluster generation algorithm that was configurable for a number of parameters. The  $k$ -means

clustering algorithm was used for this experiment, since it is a widely accepted algorithm that produces hyperspherical clusters with a relatively low computational complexity.

This experiment determined the number of partitions,  $\mathbb{P}$ , to generate, such that the partitions were a subset of all possible partitions,  $\mathbb{P} \in \mathbb{P}_K$ . A partition is a result from a clustering solution that contains one or more clusters,  $P_N \in \mathbb{P}$ , such that  $P_N = \{C_1, C_2, C_3, \dots, C_k\}$ . As a result, there were a configurable number of partitions in  $\mathbb{P}$ .

By implementing bagging, each partition had a pseudorandom number of clusters that were generated and were based on a pseudorandom number of features. The result was a diverse set of features, a diverse set of partitions with different clustering results, and a generalizable clustering solution. Using bagging helped to prevent the problem of overfitting.

### *Experimental Design*

The algorithm that was used to implement the cluster generation is shown in Figure 6. After predetermining the number of partitions,  $N$ , to generate, this algorithm created a sampling plan that was based upon the number of features provided in the dataset. For each partition, the sampling plan included a selection of features to use from the dataset. The values that were selected in the sampling plan were determined experimentally, with the objective of optimizing the clustering results while minimizing computational complexity.

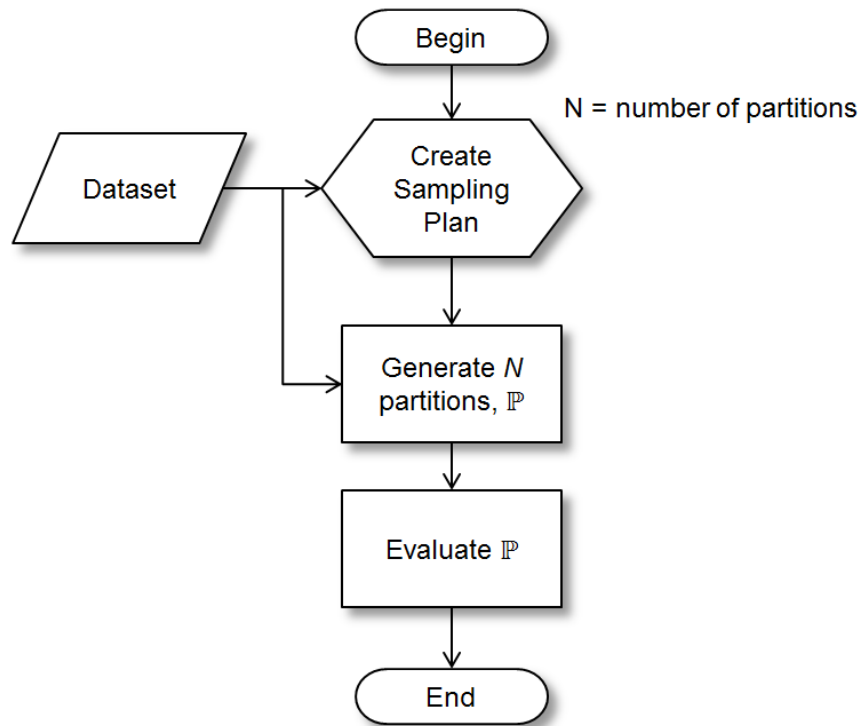


Figure 6. Algorithm for cluster generation

Following the sampling plan, the algorithm generated  $N$  partitions using the  $k$ -means clustering algorithm. The partitions were stored in a data structure that was used in subsequent stages of the intrusion detection solution, but this data was also available for export to a CSV file for off-line analysis. An evaluation step was included in this algorithm, since it is expected that some of the pseudorandom clustering solutions defined in the sampling plan would not cluster well and could be discarded immediately.

### *Evaluation*

The results of this experiment were analyzed using statistical frequency analysis of a variety of experimental runs. The focus of this experiment was to determine the optimal settings for cluster generation before moving on to the next experiment.

## **Experiment 2: Probabilistic Anomaly Detection**

The second experiment focused on two algorithms. First, it implemented and evaluated the effectiveness of using the cluster generation results to identify anomalous events. Second, it implemented and evaluated the probability that a computer host was being attacked.

### *Experimental Design*

Figure 7 shows an overview of the algorithm that was developed for this experiment. The input to this algorithm was the set of partitions that were developed in the first experiment. For each partition in the set, the algorithm determined if each cluster in the partition was an anomaly. From this a probability was calculated to determine if an event represented an attack,  $P(E)$ . After computing  $P(E)$  for each event, these probabilities were used to update a list of each computer system to reflect the probability that the computer system was being attacked,  $P(A)$ .

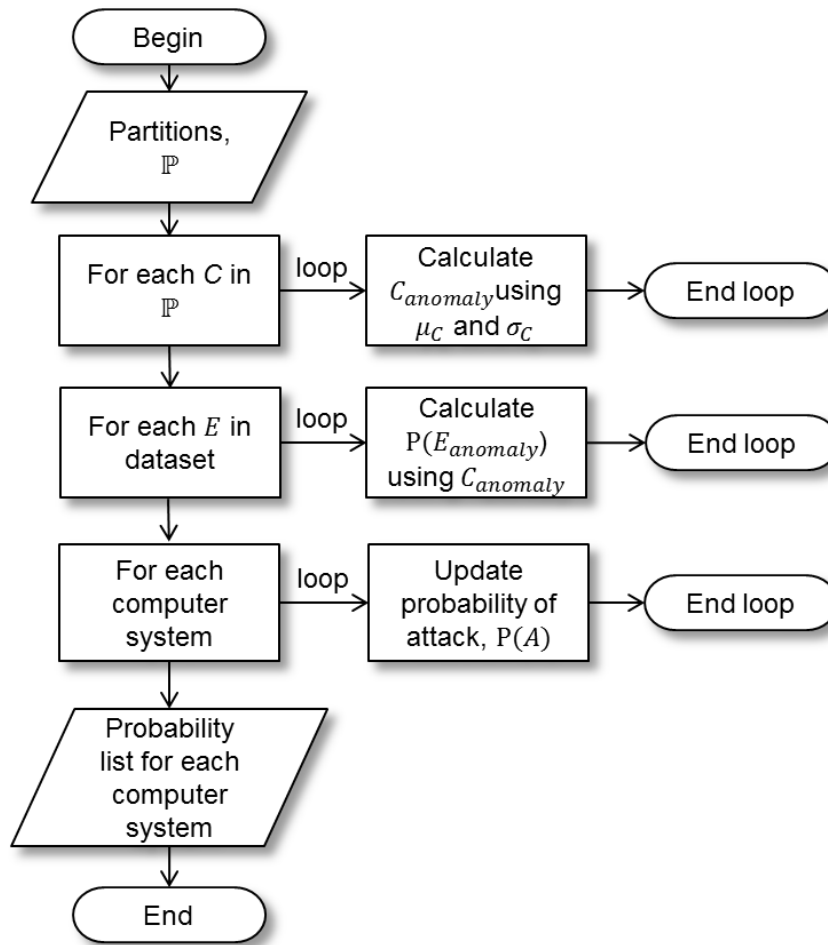


Figure 7. Algorithm for probabilistic anomaly detection

It was expected that the computational complexity would be approximately  $O(2n)$ , where  $n$  is the number of events in the dataset. It was also expected that the computational complexity of cluster anomaly evaluation would be negligible, since it was limited by the number of partitions. Further, it was expected that the loop through each event would have a linear computation requirement and would thus be  $O(n)$ . The evaluation to determine if a computer system was compromised would be limited to the number of computer systems times the average number of events per computer system,

which should be approximately  $O(n)$ . This appears to be reasonable for large intrusion datasets and this was evaluated as part of this experiment.

### *Evaluation*

The results of this experiment were evaluated for accuracy compared to the ground truth data provided with the datasets. For the NSL-KDD dataset, this experiment only evaluated the accuracy of the anomaly detection at the event level, since individual computer systems were not identified in this dataset. For the UNSW-NB15 dataset, this experiment evaluated both the accuracy of the anomaly detection and threshold at which a computer system's probability of attack was reasonable.

To evaluate the accuracy of the probability that an event is anomalous, it was expected that a threshold probability would be selected to represent an event as an attack. Then the accuracy and error rates would be calculated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

$$Error Rate = 1 - Accuracy \quad (10)$$

To evaluate the probability that a computer system was being attacked, it was expected that subsets of the data would need to be presented to the algorithm to create a variety of scenarios to simulate targeted attacks against a reduced number of computer systems; however, after analysis of the datasets, the entire dataset was presented to the algorithm. The evaluation included the numbers of normal and attack records injected



into the algorithm along with a comparison of the probabilities of a computer system being attacked.

### **Experiment 3: Active Learning**

The third experiment incorporated active learning into the overall solution. In this context, active learning was implemented to sample certain unlabeled data that would be sent to the oracle. An oracle is an entity that knows what the correct label is for the data and represents a human subject matter expert. The objective was to minimize the amount of data that needed to be sent to the oracle while ensuring that sufficient labels were provided to improve the overall machine learning output. Since this experiment included datasets that had ground truth data to distinguish between attacks and normal events, the oracle was created programmatically, rather than relying on a human subject matter expert.

The queries to the oracle were used to determine if an event was an attack or normal, as well as to determine if the probability that a computer system being attacked was correct. Thus, the oracle stored ground truth information for both types of events. The specific events and computer systems that were presented to the oracle were based on a sampling strategy that was built upon prior research (McElwee, 2017). The results were compared to arrive at a recommended sampling approach.

The incorporation of the feedback into the probabilistic anomaly detection portion of the solution was determined after the second experiment had been completed. It was expected that the results of the active learning would be used to update  $P(E)$  prior to updating  $P(A)$  for a computer system. It was also expected that the active learning results

could also update  $P(A)$  directly, based on feedback that a computer system was not being attacked.

### *Evaluation*

The accuracy results from Experiment 2 were used as a basis of comparison for this experiment. The objective was to improve the accuracy of the overall solution by incorporating domain knowledge from subject matter experts, as represented by the oracle. In addition, this experiment evaluated the accuracy compared to the number of requests sent to the oracle, since it was obvious that if all events and computer system-level decisions were sent to the oracle, then 100% accuracy could have been achieved. Thus, it was expected that there would be a sigmoid curve when the accuracy was plotted against the number of requests sent to the oracle.

### **Resource Requirements**

This research had three primary resource requirements. First, datasets for intrusion detection were required as the input. This research did not create a new intrusion detection dataset, but rather relied upon existing, publicly available datasets. As mentioned previously, the two datasets that were used for this research were the NSL-KDD and UNSW-NB15 datasets.

The second resource requirement was a development environment, including computing hardware, programming languages, and libraries for data handling and machine learning. The development environment consisted of a laptop computer running Microsoft Windows 10. The intrusion detection system and algorithms were developed

using Python and the PyCharm integrated development environment. Programming libraries that were implemented in this research included *scikit-learn* machine learning algorithms and *pandas DataFrames*. TensorFlow was considered for its deferred processing capabilities and its ability to use GPU processing, but it was not used in this research.

The third resource requirement was an environment for processing the data using the algorithms that had been developed. Most of these algorithms performed satisfactorily in the development environment, but cloud computing services were used to provide additional processing capabilities that reduced the computation time for cluster generation.

## **Summary**

This chapter introduced the approach for probabilistic clustering ensembles with active learning for intrusion detection. It provided a high-level overview of the solution and the algorithms, as well as details regarding the datasets that were used. This chapter included the approach that was used for cluster generation and the method for probabilistic anomaly detection. This chapter also provided an overview of how active learning would be applied after the probabilistic anomaly detection function was finalized.

This chapter described the testing and evaluation approaches that were used through a series of three experiments. The first experiment focused on cluster generation to evaluate the optimal way to generate diverse clustering solutions that were used later for anomaly detection. The second experiment focused on probabilistic anomaly

detection by applying probabilistic reasoning to the observation of events, leading to the probability that a computer system was being attacked. Finally, the third experiment enhanced the solution with active learning approaches and compared the accuracy to that of the probabilistic anomaly detection experiment.

## Chapter 4

### Results

#### Introduction

This chapter describes the implementation and results of the three experiments conducted in this research. It details the design and implementation of the algorithms by using a class diagram and a description of the classes. Next, this chapter describes the selected datasets and preprocessing.

This chapter presents the design of each experiment as well as the evaluation approach, observations, and preliminary conclusions. The first experiment implemented and evaluated cluster generation, and the results, which were surprising, were foundational to the remainder of the experiments. The second experiment implemented the anomaly detection algorithm and evaluated the use of the partitions of clustering results to find both the probability that an event was an anomaly,  $P(E)$ , and the probability that a computer system was under attack,  $P(A)$ . Once again, the results were different than expected, but demonstrated the success of the algorithm. Finally, the third experiment implemented active learning to update  $P(A)$  by correctly labeling a sampling of events for the computer systems that were found to have anomalies. The result was that false positives were eliminated after updating the labels of a small number of events.

## Experiment Design and Implementation

The experiments implemented the algorithms defined in Figures 6 and 7 in Chapter 3, Methodology. The experiments were implemented using object-oriented design with Python 2.7 using custom classes. In addition, existing libraries were used, such as *pandas* and *scikit-learn*. Figure 8 illustrates the class diagram that was used for the experiments.

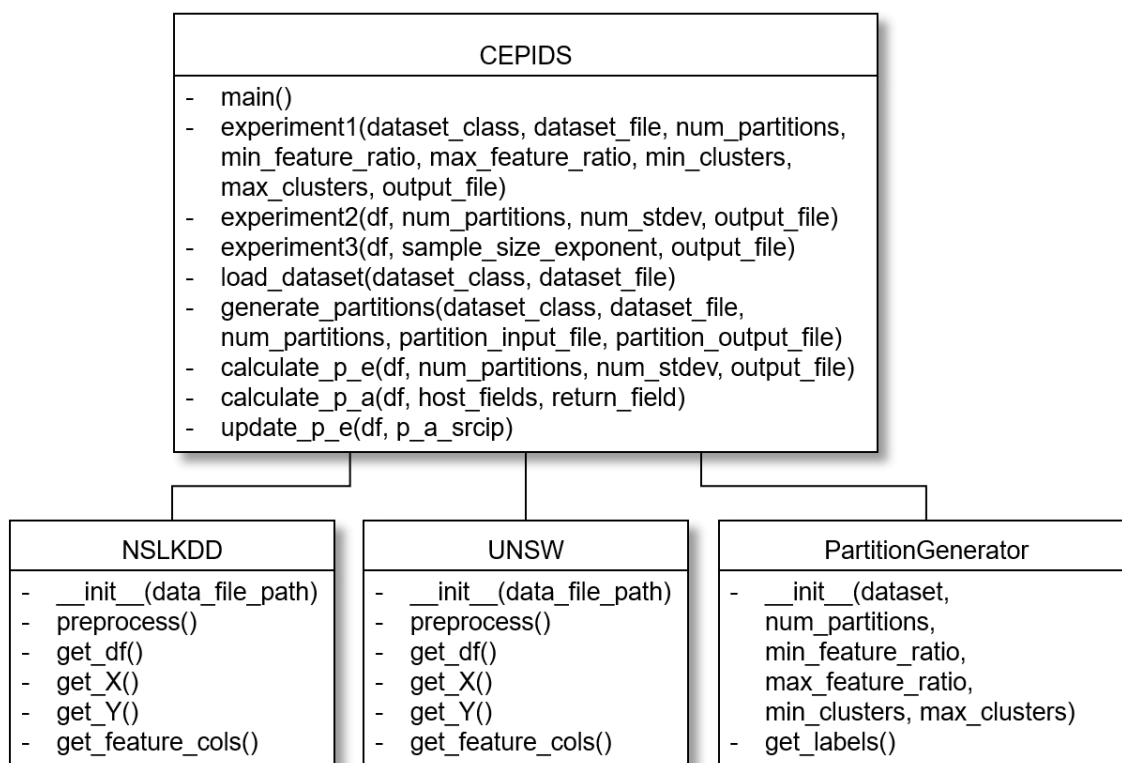


Figure 8. Class diagram

### *CEPIDS Class*

The class name, *CEPIDS*, is the acronym for Clustering Ensemble Probabilistic Intrusion Detection System. This class is responsible for the high-level execution of the algorithm, the calculation of  $P(E)$  for events, the calculation of  $P(A)$  for computer

systems, and for implementing the active learning. It contains a *main()* function that allows it to be executed directly for complete execution of cluster generation, probabilistic anomaly detection, and active learning. The *main()* function orchestrates the experiments. The *CEPIDS* class instantiates objects from the subsequent classes, *NSLKDD*, *UNSW*, and *PartitionGenerator*.

#### *NSLKDD and UNSW Classes*

The *NSLKDD* and *UNSW* classes allowed their representative datasets to be loaded from a file and preprocessed. After loading and preprocessing the datasets, these classes could return a *DataFrame* that represented the full dataset, a list of features, or a list of labels to allow for post processing and evaluation. Although these two datasets have different sets of features, using the same functions in each class allowed the implementation of each dataset to be abstracted and allowed the classes to be used interchangeably in the *CEPIDS* class. As a result, these experiments can be extended to other datasets by implementing the specific details of the dataset in a new dataset class without modifying the logic of the algorithms. The *NSLKDD* and *UNSW* classes were implemented in a single Python package, called *datasets.py*, which contained global functions for preprocessing, such as filling in missing data values, encoding categorical features, and scaling values to a range.

### *PartitionGenerator Class*

The *PartitionGenerator* class implemented the cluster generation. The initialization of this class required an instantiated dataset object, which could be either of the *NSLKDD* or *UNSW* classes. The class initialization also included configurable parameters for selecting the number of partitions to generate, the minimum and maximum ratio of features to include in generating the partitions, as well as the minimum and maximum number of clusters to generate in each partition. Using the upper and lower bounds of features and the number of clusters, the initialization of this class selected pseudorandom numbers within those bounds. After the initialization was completed, the cluster labels were retrieved using the *get\_labels()* function, which returned a list of numeric cluster labels for each partition. Each list of cluster labels followed the order of the original dataset, which allowed them to be joined directly with the original dataset as new features.

### *Development Environment*

The development environment for the experiments in this research used PyCharm for Python development. Source code was controlled in a private github repository. Most experiments were conducted on a Lenovo laptop with an Intel Core i7 CPU running at 2.1 GHz, with 8 GB of memory, and with the Windows 10 operating system. Cluster generation was found to be the most computationally complex problem, so to generate sufficient numbers of clustering ensemble results for validation, Amazon EC2 instances were also used. These instances were optimized for computational work, with 3 GHz, Intel Xeon Platinum CPUs and 8 GB of memory. The EC2 instances allowed 10



simultaneous executions of the cluster generation, with 100 partitions per execution. This reduced the time to generate 10 sets of partitions to the same time it took to produce a single set.

### **Input Dataset Analysis and Preparation**

The first step in conducting the experiments was to evaluate the datasets and preprocessing requirements that were needed. This preparation work was performed using Microsoft Excel pivot tables. This allowed various ways of examining the datasets to understand their characteristics.

#### *NSL-KDD*

NSL-KDD was one of the datasets used for this research. The NSL-KDD dataset is a subset of the KDD Cup 1999 dataset that eliminates duplication. An analysis of the composition of the NSL-KDD training dataset found that it contained only 53% normal records. An important assumption of this research is that network events are highly imbalanced, with a predominant number of normal records and a very small number of attack records. Since nearly half of this dataset contained attacks, it could not be considered as representative of normal network conditions, which might only contain a very small percentage of attacks. To compensate for this even distribution of normal and attack records, a derived dataset was prepared that contained a subset of the events from the NSL-KDD dataset. By eliminating the denial of service records, the resulting dataset contained 98% normal records and was more representative of a realistic network that was experiencing targeted attacks. All subsequent experiments using the NSL-KDD

dataset used this derived dataset. Table 6 shows the distribution of classes in both the original and the derived datasets.

Table 6

*Original and Derived NSL-KDD Dataset Label Distributions*

<b>Original Dataset</b>			<b>Derived Dataset</b>		
Label	Records	% Total	Label	Records	% Total
Normal	67,343	53%	normal	67,343	98%
neptune	41,214	33%	warezclient	890	1%
Satan	3,633	3%	guess_passwd	53	0%
ipsweep	3,599	3%	buffer_overflow	30	0%
portsweep	2,931	2%	warezmaster	20	0%
Smurf	2,646	2%	land	18	0%
Nmap	1,493	1%	imap	11	0%
Back	956	1%	rootkit	10	0%
teardrop	892	1%	loadmodule	9	0%
warezclient	890	1%	ftp_write	8	0%
Pod	201	0%	multihop	7	0%
guess_passwd	53	0%	phf	4	0%
buffer_overflow	30	0%	perl	3	0%
warezmaster	20	0%	spy	2	0%
Land	18	0%	<b>Total</b>	<b>68,408</b>	<b>100%</b>
Imap	11	0%			
Rootkit	10	0%			
loadmodule	9	0%			
ftp_write	8	0%			
multihop	7	0%			
Phf	4	0%			
Perl	3	0%			
Spy	2	0%			
<b>Total</b>	<b>125,973</b>	<b>100%</b>			

### *UNSW-NB15*

The UNSW-NB15 dataset includes a variety of files. First, this dataset includes full packet capture data that was the source data for all subsequent files generated for this dataset. Next, it includes a training and testing dataset that was created using the packet capture data and is well-suited for machine learning. This data is similar in composition to the NSL-KDD dataset in that it does not include computer system identifiers, such as IP addresses. As a result, these files were not useful for this research. The UNSW-NB15 dataset includes four comma separated value files that were generated from the packet capture data using Argus, Bro, and customized algorithms. These files each represent network connection events that include source and destination IP addresses for each event. It is this set of data that was used for this research, since it included the classes of the events and uniquely identified the source destination IP addresses. The classes of events in this dataset consisted of: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms, and Normal.

The experiments used one of the four files, UNSW-NB15\_1.csv, which contained 700,001 events, 44 unique destination IP addresses, and 40 unique source IP addresses. It was representative of the other datasets. This file was highly imbalanced, consisting of 97% normal records. As a result, no modifications or derived datasets were needed. Table 7 shows the distribution of classes in this dataset.

Table 7

*UNSW-NB15\_1 Dataset Label Distributions*

Label	Records	PctTotal
Normal	677,786	96.8%
Generic	7,522	1.1%
Exploits	5,409	0.8%
Fuzzers	5,051	0.7%
Reconnaissance	1,759	0.3%
DoS	1,167	0.2%
Backdoors	534	0.1%
Analysis	526	0.1%
Shellcode	223	0.0%
Worms	24	0.0%
Total	700,001	100%

**Experiment 1: Cluster Generation***Design*

The purpose of the first experiment was to build a foundation of cluster generation that could be used in subsequent experiments. All clustering performed in this experiment used the  $k$ -means clustering algorithm. It was important to ensure that each of the partitions of clustering results was diverse, so that the resulting partitions represented multiple perspectives of the data groupings. To generate diverse clusters, this experiment began with a bagging plan that included a pseudorandom set of features and a pseudorandom number of clusters per partition. As a result, each partition was built using a different number of features from the original dataset, and those features were randomly selected. Each partition also had a different number of clusters.

To select the features to be used in each partition, this experiment was constructed to be generalizable to other datasets. As a result, the bagging plan included a maximum of 75% of the available features from the original dataset, and a lower limit of 25%. For example, the UNSW-NB15 dataset contains 49 features, including a label and an attack category. Since the label and the attack category represent the class of each event, this dataset has 47 features that are useful for clustering. Thus, the number of features that were clustered in each of the partitions generated for the UNSW-NB15 dataset ranged from 12 to 35.

To select the number of clusters to generate for each partition, this experiment included parameters for the minimum and maximum number of clusters to generate. Initial experiments used low numbers of clusters, such as a minimum of four and a maximum of 20. The results were analyzed using Microsoft Excel pivot tables, and the result was that each cluster had a predominantly high number of normal records. The ranges were expanded until a more reasonable distribution of normal records was identified. The final range for the number of clusters in each partition was a minimum of 40 and a maximum of 100. Increasing the upper limit further may have improved the distribution of events in each cluster, but higher numbers of clusters required more computational processing time, so a maximum of 100 was selected as a trade-off between performance and diversity. Prior research showed that ensembles of weak clusterers were better than single clustering algorithms, so it was expected that any limitations of cluster generation would be offset in the cluster evaluation stage (Topchy et al., 2005).

The first experiment produced a *DataFrame* and an output file that were used for the evaluation of the results as well as for the input to the second experiment. Each

partition that was generated was represented as a feature appended to the original dataset. Each partition was assigned a feature name that began with the letter P and was followed by the partition number, such as P0 through P99. The values that were populated in the partitions were integers that represented the clustering labels for each partition. Initial experiments began with as few as 10 partitions and went as high as 100 partitions to provide a diverse set of clustering solutions for the second experiment.

### *Analysis*

The most important analysis in this first experiment was to determine if there was a clustering generation strategy that would allow anomalies to be detected according to the formula:

$$C_{anomaly} = \begin{cases} 0, & \mu_c - K\sigma_c \leq |C_i| \leq \mu_c + K\sigma_c \\ 1, & (|C_i| < \mu_c - K\sigma_c) \cup (|C_i| > \mu_c + K\sigma_c) \end{cases} \quad (11)$$

Using this formula, normal clusters were defined by counting the number of records in each cluster,  $|C_i|$ , and determining if it was within  $K$  standard deviations of the mean number of events in each cluster. Anomalies were defined by evaluating if the count of records in each cluster was outside of the range of normal events.

The cluster generation performed in this experiment began with the NSL-KDD dataset because of its smaller size and lower computational resource requirements. The output file from this experiment was imported into Microsoft Excel and was analyzed using pivot tables. This analysis isolated a sample partition, calculated the mean number of records per cluster, calculated the standard deviation of the number of records per

cluster, and calculated a variable number of standard deviations above and below the mean.

This experiment found that the results of the clustering ensembles were not distributed as originally expected. Instead, clusters with the highest numbers of records predominantly had 100% normal classes. Clusters with the least number of records had a mixture of normal and anomalous classes, but none of these clusters could be identified as exclusively anomalous. Table 8 shows an example of what was found for a sample partition in one of the tests. In this test, the partition had 13 clusters, labeled from 0 to 12. Using the total record count for each cluster, two standard deviations above the mean was 14,944. The only cluster that had more records than this threshold was cluster 6, which had 14,974 records. Cluster 6 consisted exclusively of normal records. It is also important to note that two standard deviations below the mean was -4,420, making it impossible for any clusters to have a number of records below this threshold.

Table 8

*Distribution of Attack and Normal Records in Sample Partition*

Label	Attack	Normal	Total
6	0	14,974	14,974
8	4	14,466	14,470
2	0	8,704	8,704
1	0	6,668	6,668
4	582	5,234	5,816
5	44	4,534	4,578
3	79	3,555	3,634
7	4	2,705	2,709
9	0	2,337	2,337
12	0	2,087	2,087
0	311	821	1,132
10	12	845	857
11	29	413	442

After observing this distribution in the sample of partitions, this experiment implemented a Python function that generated the pivot tables for all partitions generated using the NSL-KDD dataset. The result was consistent across approximately 99% of the partitions. Exploring this observation further, the function was applied to the UNSW-NB15 dataset and found that clusters with a number of records above the threshold were consistently normal classes with a high degree of accuracy. As a result, this experiment modified the original assumption by finding that:

$$C_{anomaly} = \begin{cases} 0, & |C_i| \geq \mu_c + K\sigma_c \\ < 0.5, & |C_i| < \mu_c + K\sigma_c \end{cases} \quad (12)$$

This updated function resulted in a high degree of certainty of what was normal when two standard deviations was selected for  $K$ . For clusters with less than this threshold number of events, there was uncertainty, which is reflected as a probability of 0.5 or less that an event is anomalous. It was expected that a higher number of standard deviations above the mean would result in a more accurate prediction of normal events, but this resulted in less records that met the criteria and did not improve the accuracy. After evaluating the clustering results using ranges of  $K$  standard deviations from 1.5 to 4, this experiment found that 2 standard deviations performed consistently well in identifying the normal classes.

### *Computational Efficiency*

Cluster generation was the most computationally expensive algorithm of the experiments. Although this was not a significant problem for the NSL-KDD dataset,



because of the lower number of records, it was a problem for the UNSW-NB15 dataset. When this experiment used a maximum number of 20 clusters per partition, the cluster generation was relatively quick; however, to obtain a more diverse set of clusters, a maximum of 100 cluster centers per partition was selected. For the UNSW-NB15 dataset, to generate 100 partitions with up to 100 clusters per partition, the algorithm took approximately 12 hours to complete.

The computational complexity for the *scikit-learn* implementation of  $k$ -means clustering is  $O(knT)$ , where  $k$  is the number of clusters,  $n$  is the number of samples, and  $T$  is the number of iterations. This was prohibitive, since all 700,001 samples in the UNSW-NB15 dataset were used, and since the maximum number of clusters per partition was as high as 100. In addition, a maximum of 300 iterations was selected for the  $k$ -means clustering algorithm, which was time consuming to run as many as 100 partitions per cluster generation run.

The generation of partitions can be scaled by running the algorithm in parallel. This experiment overcame some of the computational complexity by submitting the input dataset to the same algorithm running on multiple servers. This allowed the generation of the multiple sets of partitions needed for this research to be created in the same time it took to create a single set of partitions.

### *Observations*

This experiment created a diverse set of partitions using the  $k$ -means clustering algorithm to generate a range of clusters per partition from 40 to 100 for the NSL-KDD and UNSW-NB15 datasets. It also found that a random bagging plan that ranged from

25% to 75% of available features was effective in generating diverse partitions. Most importantly, this experiment found that the distribution of clustering results by counting the events in each cluster did not clearly identify anomalous events, but rather could be used accurately to identify certain normal records. The result of this experiment indicated that  $P(E)$  for each event was not a hard probability but rather was a soft belief that could be used as an observation for updating  $P(A|E)$ . The observations from this experiment were carried forward into the second experiment to determine if the results could be used for anomaly detection.

### **Experiment 2: Probabilistic Anomaly Detection**

This experiment consisted of two algorithms. Beginning with the partitions generated in the first experiment, this algorithm first calculated the probability that an event was an anomaly,  $P(E)$ . As found in the first experiment, this turned out to be a belief that had more accuracy for normal events than for anomalies. The second algorithm used  $P(E)$  to predict the probability that a computer system was experiencing a cyberattack,  $P(A)$ .

#### *Design of $P(E)$ Calculation*

Using the proposed algorithm from the methodology section, it was expected that in this experiment,  $P(E)$  would be calculated as the average of the anomalous clusters:

$$P(E_{anomaly}) = \frac{1}{n} \sum_{i=1}^n C_{anomaly} \quad (13)$$

In the first experiment, this research found that it was more predictive to look at which clusters contained normal records rather than which clusters contained anomalies.

To evaluate this algorithm and update this formula, this experiment used Microsoft Excel pivot tables to find the number of partitions that classified each event as normal. Figure 9 illustrates the distribution of the event-level classes according to the number of votes that a set of 100 partitions generated using the NSL-KDD dataset. In this graph, 100 represents all the partitions voting that the event was normal and 0 represented no votes that the event was normal. The normal and attack classes are shown on different axes and at different scales because of the imbalanced nature of the dataset. It is important to note in this graph that above the midpoint number of votes, which was 50, this approach demonstrated high accuracy for detecting normal events.

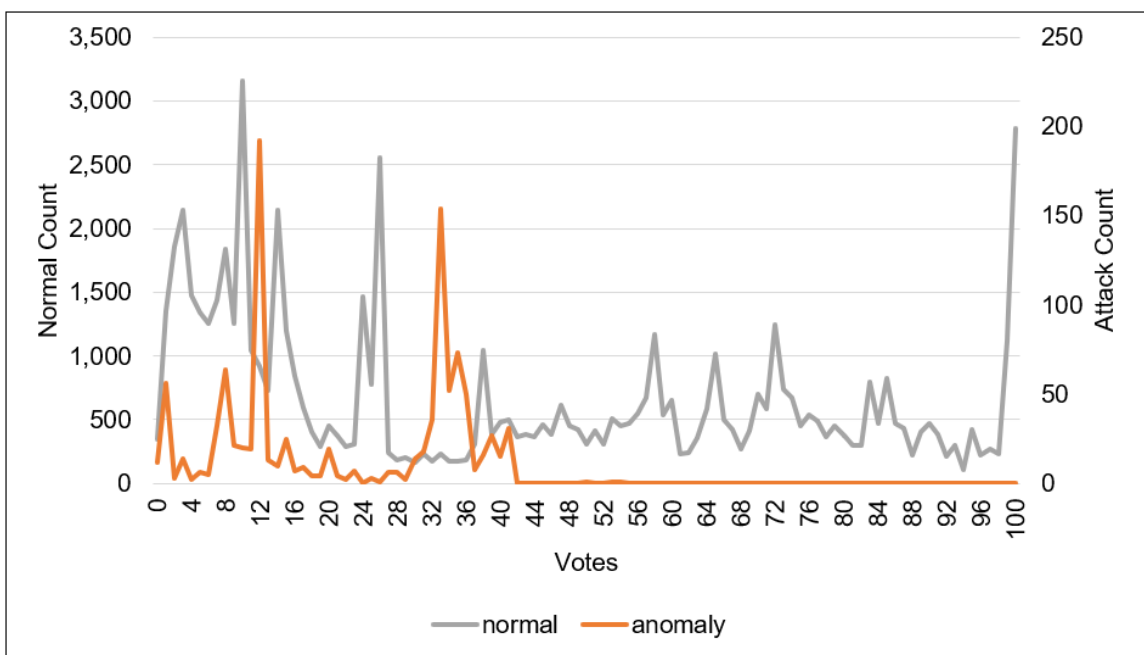


Figure 9. Graph of votes for NSL-KDD dataset

A different pattern was found when evaluating the partitions generated using the UNSW-NB15 dataset with 100 partitions. Figure 10 shows the distribution of events compared to the number of votes. An interesting observation with this dataset was that there were no instances where all the partitions voted that an event was normal. The highest number of votes was 42. The pattern looked smoother, with less but more pronounced peaks. Despite the differences with the NSL-KDD dataset, it still held that above the midpoint number of votes, which was 21, this number of votes was still an accurate method for detecting normal events with a high degree of accuracy.

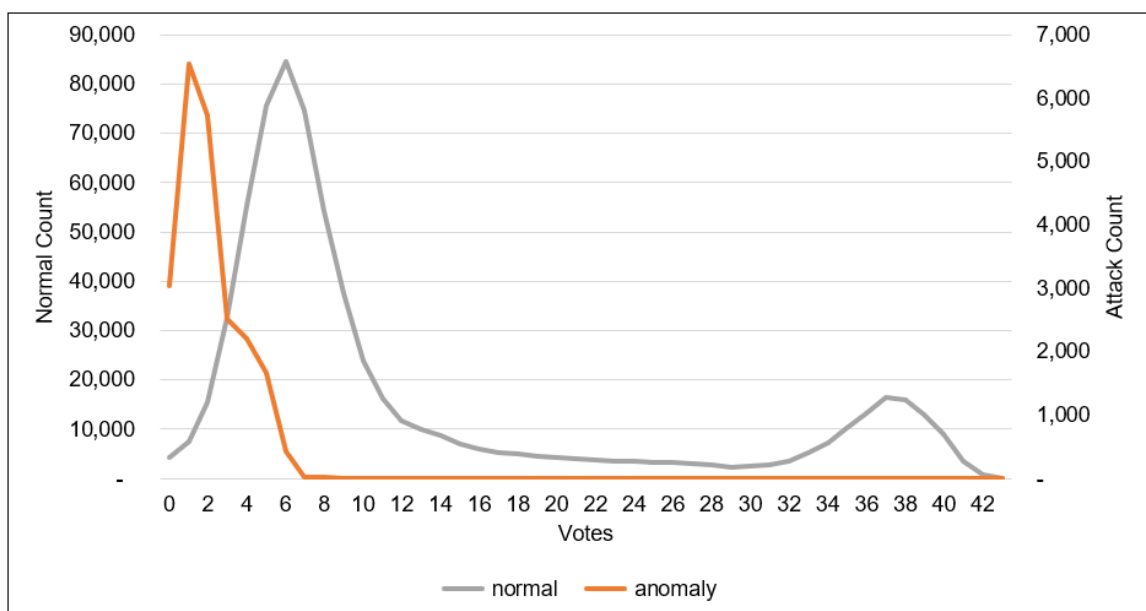


Figure 10. Graph of votes for UNSW-NB15 dataset

Since this evaluation found that  $P(E)$  was certain for highly normal clusters and uncertain for others, the calculation of  $P(E)$  was updated to assign a belief ranging from 0, for normal clusters, to 0.5, for uncertain clusters. To calculate  $P(E)$ , this experiment first evaluated  $C_{\text{anomaly}}$  using two standard deviations as the threshold. Then using voting,

for each event, this experiment counted the number of partitions that indicated the event was normal. At this point, it was found that all events, with a few exceptions, were normal when the number of normal votes ranged from the maximum of votes to half of the maximum of votes. Thus, the algorithm was developed to assign  $P(E) = 0$  for events in this upper half of the votes. Below this range, the algorithm scaled  $P(E)$  to range from 0, for the most votes below the midpoint, to 0.5, for events with the least number of votes.

$$midpoint = \frac{\max(votes)}{2} \quad (14)$$

$$P(E) = \begin{cases} 0, & votes_E \geq midpoint \\ \left(1 - \frac{votes_E}{midpoint}\right), & votes_E < midpoint \end{cases} \quad (15)$$

#### *Analysis of P(E) Calculation*

The accuracy of  $P(E)$  was calculated for events that fell above the midpoint of the maximum votes, since below the midpoint  $P(E)$  was found to represent only a level of uncertainty. Below this threshold, events were assigned a belief, which was suggestive that there may have been anomalies, but did not reflect an accurate prediction of which events are anomalies. Thus, for events above the midpoint, there were no true positives and no false positives, since above this threshold, only true negative and false negative results were expected:

$$\begin{aligned}
Accuracy &= \frac{TP + TN}{TP + FP + TN + FN} \\
&= \frac{0 + TN}{0 + 0 + TN + FN} \\
&= \frac{TN}{TN + FN}
\end{aligned} \tag{16}$$

For the NSL-KDD dataset, the accuracy of  $P(E)$  above the midpoint was found to be:

$$\begin{aligned}
Accuracy &= \frac{TN}{TN + FN} \\
&= \frac{27,394}{27,394 + 3} \\
&= 0.999890
\end{aligned} \tag{17}$$

For the UNSW-NB15 dataset, the accuracy of  $P(E)$  above the midpoint was found to be:

$$\begin{aligned}
Accuracy &= \frac{TN}{TN + FN} \\
&= \frac{129,508}{2129,508 + 6} \\
&= 0.999954
\end{aligned} \tag{18}$$

Although  $P(E)$  was not highly predictive of anomalies, it was highly predictive of normal events. Thus, for determining that a computer system was under attack,  $P(A)$ , this experiment proceeded to determine if the accumulation of uncertainty was predictive that a computer system was being attacked.

### *Design of P(A) Calculation*

After calculating  $P(E)$  for each event, this experiment then calculated  $P(A)$  as the probability that a computer system was being attacked. Through experimentation, it was found that calculating  $P(A)$  could be reduced to:

$$P(A) = \frac{1}{N} \sum_{i=1}^N P(E_N) \quad (19)$$

In this formula,  $P(A)$  is the probability that a computer system is experiencing an attack, and  $N$  is the number of events attributable to each computer system. Thus,  $P(A)$  represents the mean of  $P(E)$  when grouped by the computer system.

In calculating  $P(A)$  it was necessary to evaluate both computer systems involved in each event, the source IP address (*srcip*) and the destination IP address (*dstip*). Using the UNSW-NB15 dataset, it was not clear which of these two addresses represented the attacker or the target of the attack, so it was important to consider both addresses in calculating  $P(A)$ .

### *Analysis of P(A) Calculation*

Results varied when calculating  $P(A)$  for these two different IP addresses in each event. Experimentation found that to calculate  $P(A)$  for the *srcip*, it was more effective to group the events using a combination of *srcip* and *dstip* and to calculate the average  $P(E)$  for each pair. Further grouping the resulting dataset on just the *srcip*, the result was that for  $P(A) \geq 0.8$ , the *srcip* addresses involved in attacks were consistently predicted with 100% accuracy. For the *dstip* address, using just the average  $P(E)$  allowed the *dstip*

addresses to be identified for  $P(A) \geq 0.8$ ; however, there were some false positives of *dstip* addresses that were identified but were not involved in attacks. The inaccurate classifications were generally for destination IP addresses that had few events, which made them appear to be anomalies when compared to the entire dataset. Table 9 shows the results for a sample run including both the *srcip* and the *dstip*. The IP addresses that have  $P(A) \geq 0.8$  are shown in bold. The Result column indicates if the result was a true positive (TP), false positive (FP), or true negative (TN). There were no false negatives.

Table 9

*Prediction of P(A) by srcip and dstip*

<i>srcip</i>	Result	P(A)	<i>dstip</i>	Result	P(A)
<b>175.45.176.1</b>	<b>TP</b>	<b>1.000</b>	<b>149.171.126.12</b>	<b>TP</b>	<b>1.000</b>
<b>175.45.176.3</b>	<b>TP</b>	<b>0.920</b>	<b>149.171.126.14</b>	<b>TP</b>	<b>0.997</b>
<b>175.45.176.2</b>	<b>TP</b>	<b>0.907</b>	<b>149.171.126.15</b>	<b>TP</b>	<b>0.996</b>
<b>175.45.176.0</b>	<b>TP</b>	<b>0.842</b>	<b>149.171.126.11</b>	<b>TP</b>	<b>0.993</b>
149.171.126.0	TN	0.622	<b>149.171.126.13</b>	<b>TP</b>	<b>0.992</b>
149.171.126.2	TN	0.621	<b>149.171.126.19</b>	<b>TP</b>	<b>0.988</b>
149.171.126.4	TN	0.619	<b>149.171.126.16</b>	<b>TP</b>	<b>0.987</b>
149.171.126.5	TN	0.609	<b>149.171.126.10</b>	<b>TP</b>	<b>0.977</b>
149.171.126.3	TN	0.608	<b>149.171.126.17</b>	<b>TP</b>	<b>0.963</b>
149.171.126.6	TN	0.608	<b>224.0.0.1</b>	<b>FP</b>	<b>0.920</b>
149.171.126.9	TN	0.604	<b>149.171.126.18</b>	<b>TP</b>	<b>0.903</b>
149.171.126.7	TN	0.601	<b>10.40.170.2</b>	<b>FP</b>	<b>0.887</b>
149.171.126.1	TN	0.600	<b>32.50.32.66</b>	<b>FP</b>	<b>0.861</b>
149.171.126.8	TN	0.597	10.40.182.3	TN	0.784
59.166.0.0	TN	0.465	10.40.85.30	TN	0.656
59.166.0.9	TN	0.462	59.166.0.9	TN	0.616
59.166.0.4	TN	0.459	59.166.0.8	TN	0.607
59.166.0.1	TN	0.459	59.166.0.2	TN	0.599
59.166.0.8	TN	0.440	224.0.0.5	TN	0.597
59.166.0.3	TN	0.420	59.166.0.6	TN	0.596
59.166.0.7	TN	0.412	59.166.0.1	TN	0.591
59.166.0.6	TN	0.409	59.166.0.7	TN	0.589



Table 9

*Prediction of P(A) by srcip and dstip (cont.)*

<i>srcip</i>	Result	P(A)	<i>dstip</i>	Result	P(A)
59.166.0.5	TN	0.389	59.166.0.0	TN	0.587
59.166.0.2	TN	0.376	59.166.0.3	TN	0.583
10.40.85.1	TN	0.231	59.166.0.4	TN	0.582
149.171.126.18	TN	0.174	59.166.0.5	TN	0.564
149.171.126.15	TN	0.169	175.45.176.1	TN	0.525
10.40.182.1	TN	0.160	10.40.85.1	TN	0.513
149.171.126.10	TN	0.158	175.45.176.2	TN	0.509
149.171.126.19	TN	0.110	192.168.241.243	TN	0.449
149.171.126.11	TN	0.090	149.171.126.1	TN	0.343
149.171.126.16	TN	0.086	149.171.126.5	TN	0.339
10.40.170.2	TN	0.078	175.45.176.0	TN	0.312
10.40.182.3	TN	0.078	149.171.126.0	TN	0.305
149.171.126.13	TN	0.066	149.171.126.7	TN	0.301
149.171.126.12	TN	0.061	149.171.126.6	TN	0.294
10.40.85.30	TN	0.056	149.171.126.8	TN	0.293
192.168.241.243	TN	0.053	149.171.126.9	TN	0.273
127.0.0.1	TN	0.026	175.45.176.3	TN	0.262
149.171.126.17	TN	0.022	149.171.126.3	TN	0.231
			149.171.126.2	TN	0.228
			149.171.126.4	TN	0.225
			10.40.198.10	TN	0.137
			127.0.0.1	TN	0.000

To ensure that the results were consistent, this experiment included ten runs of the algorithm, using a different set of randomly generated partitions for each run. Table 9 shows the accuracy of calculating  $P(A)$  for the *srcip* and *dstip*. As Table 10 demonstrates, the results were reproducible for each of the runs. Thus, the algorithm did not require a specific set of features or feature engineering to be successful, since the random partitions of clusters proved to be successful for all runs.

Table 10

*P(A) Accuracy for Ten Runs*

<i>srcip</i>						<i>dstip</i>				
Run #	TP	TN	FP	FN	Accur.	TP	TN	FP	FN	Accur.
1	4	41	0	0	1.00	10	28	6	0	0.86
2	4	41	0	0	1.00	10	27	7	0	0.84
3	4	41	0	0	1.00	10	26	8	0	0.82
4	4	41	0	0	1.00	10	31	3	0	0.93
5	4	41	0	0	1.00	10	31	3	0	0.93
6	4	41	0	0	1.00	10	30	4	0	0.91
7	4	41	0	0	1.00	10	28	6	0	0.86
8	4	41	0	0	1.00	10	25	9	0	0.80
9	4	41	0	0	1.00	10	30	4	0	0.91
10	4	41	0	0	1.00	10	30	4	0	0.91

*Computational Efficiency*

It was expected that the computational complexity for this algorithm would be  $O(2n)$ . By using pandas *DataFrame* objects extensively, this algorithm did not directly loop through each event. It did loop through each partition, which resulted in 100 iterations. The algorithm relied upon the built-in optimizations of the *DataFrame* class to perform calculations for the entire dataset. As a result, it can be estimated that this algorithm's complexity may be expressed as less than  $O(n)$ . In testing, algorithms for calculating  $P(E)$  and  $P(A)$  completed in less than one minute. Thus, the use of the algorithm for anomaly detection did not add any significant computational penalty beyond the performance of multiple runs of the  $k$ -means clustering that were used in the first experiment to generate the partitions.

### *Observations*

The results of this experiment demonstrated that unsupervised anomaly detection at the event-level was not accurate for detecting attack classes. Instead, event-level detection using this approach found a portion of the normal classes with highly accurate results. Using this observation to calculate a belief that an event may be anomalous was found to be highly effective in calculating the probability that a computer system was under attack. By examining the *srcip* and *dstip* separately, the accuracy was consistently 100% for the *srcip* and ranged from 80% to 93% for the *dstip*. As a result, this experiment demonstrated that the clustering ensemble probabilistic intrusion detection system detected the computer systems that were under attack. It accomplished this prediction without relying upon labeled data for training and without training during attack-free time periods to detect abnormal events.

## **Experiment 3: Active Learning**

### *Design*

The third experiment incorporated active learning into the overall intrusion detection system. When initially proposing this research, it was unclear how active learning would be applied, since it depended on how the first two experiments were implemented. Since active learning queries an oracle to label of a select number of records, the primary design consideration was how large of a sample to submit to the oracle. Options considered were to sample from the events to update  $P(E)$  or to sample from the computer system level to update  $P(A)$ .

Figure 11 lists the pseudocode for this algorithm. This algorithm selects all *srcip* and *dstip* computer systems that had  $P(A) \geq 0.8$ . From this subset of all computer systems, a pseudorandom sample of events for each of these computer systems was selected. These sampled events were submitted to the oracle for labeling. If the oracle returned a response that the event was an attack, then the  $P(A)$  for that *srcip* or *dstip* was updated to 1.0, signifying that the computer system was experiencing an attack.

```

For each srcip with  $P(A) \geq 0.8$ :
    Collect events with srcip
    Create sample of  $N$  records from collected events
    For each record in sample:
        Ask oracle for labels
        If number of attack labels  $> 0$ :
            Update  $P(A)$  to 1
        Else:
            Update  $P(A)$  to 0
For each dstip with  $P(A) \geq 0.8$ :
    Collect events with dstip
    Create sample of  $N$  records from collected events
    For each record in sample:
        Ask oracle for labels
        If number of attack labels  $> 0$ :
            Update  $P(A)$  to 1
        Else:
            Update  $P(A)$  to 0

```

*Figure 11.* Pseudocode for active learning algorithm

Unlike McElwee (2017), in which a separate oracle class was constructed to simulate the human analyst, this experiment relied upon the *DataFrame* that was used as a data structure for holding both the original dataset and the results. Attack labels were identified by querying the label column of the *DataFrame*. As a result, the queries always resulted in accurate labels, even for rare events.

### Analysis

Since the goal of this research was to reduce the workload of human analysts, the purpose of this experiment was to find the minimum number of samples to submit to the oracle for labeling that would still result in a high accuracy. To be representative of the events assigned to each *srcip* and *dstip*, a sample size of  $\sqrt{N}$  was selected as the starting point, where  $N$  was the number of events for each *srcip* or *dstip*. Initial experimentation with this sample size concluded that active learning consistently identified computer systems under attack with 100% accuracy. Next, this experiment proceeded to reduce the number of samples and evaluate the accuracy, as shown in Table 10.

Table 10

#### Active Learning Accuracy

	<i>srcip</i>		<i>dstip</i>	
	Sample	Accuracy	Sample	Accuracy
$\sqrt{N}$	317	100.0%	499	100.0%
$\sqrt[3]{N}$	73	100.0%	134	100.0%
$\sqrt[4]{N}$	35	100.0%	70	100.0%
$\sqrt[5]{N}$	23	100.0%	47	97.7%
$\sqrt[10]{N}$	10	100.0%	22	95.4%

The results in Table 10 show that for *srcip*, the accuracy is 100% for all sample sizes, but this is because the original accuracy of the anomaly detection in the second experiment was already 100%. Thus, the *dstip* is a better indicator of sample sizes for active learning. Each sample size improved the original accuracy of anomaly detection

for the *dstip*. Sample sizes of  $\sqrt[4]{N}$  and above achieved accuracies of 100%. To confirm this sample size was the minimum for achieving this accuracy, this sample size was run 10 times and achieved the same results each time.

Figure 12 shows the accuracy of the active learning compared to the sample size. For sample sizes smaller than  $\sqrt[4]{N}$ , the accuracy shows a downward slope.

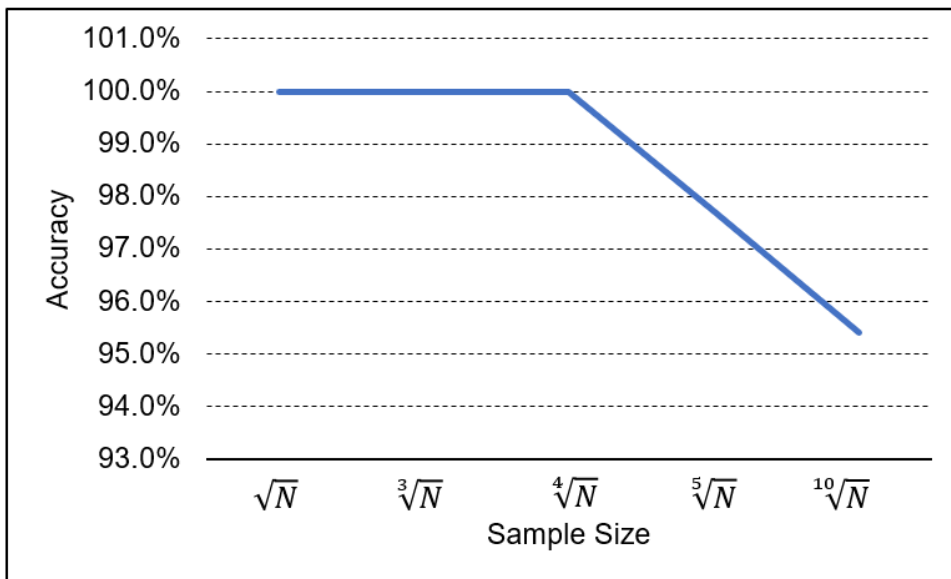


Figure 12. Accuracy of active learning compared to sample size

### *Computational Efficiency*

The algorithm for active learning with a simulated oracle was highly efficient, since it had access to  $P(E)$ ,  $P(A)$ , and the original features, including the label. Thus, by creating the random sample using the *DataFrame*, the algorithm could stop as soon as a single attack was identified. For each run, the computation time for this algorithm was a few seconds. It was estimated that, where  $N$  is the total number of that could be sent to

the oracle for labeling, the complexity was  $O(2^{\sqrt[4]{N}})$ , since for each sampled record, the algorithm completed the same loop for both the *srcip* and the *dstip*.

### *Observations*

This experiment demonstrated that by sampling events for *srcip* and *dstip* with a sample size of  $\sqrt[4]{N}$ , the residual error from anomaly detection could be reduced to achieve 100% accuracy. This approach relies on the anomaly detection algorithm identifying all of the true positives, since this active learning approach focuses on eliminating false positives, not on reducing false negatives. As a result, when the active learning algorithm was applied to the UNSW-NB15 dataset with 700,001 events, a human analyst would have been required to review a maximum of 105 events to identify all of the computer systems involved in an attack with 100% accuracy.

### **Summary**

This chapter described the design and implementation of the experiments performed in this research. It showed the class diagram of the major components of the system and reviewed the function of each. In addition, this chapter provided a more in-depth description of the dataset characteristics and preprocessing needed to ensure that the datasets were highly imbalanced.

This chapter also reviewed each experiment, including specific design considerations, analysis, and observations. Since computational efficiency has been an important consideration when implementing clustering ensemble evaluation, the computational efficiency was discussed for each experiment. Each of the experiments

contributed to the overall demonstration that clustering ensembles were effective for anomaly-based intrusion detection. The first experiment identified a characteristic of imbalanced datasets that allowed the isolation of a significant portion of normal events. Using this characteristic of imbalanced data, the second experiment assigned a belief to the events to reflect uncertainty. In addition, the second experiment estimated the probability that a computer system was experiencing an attack by calculating the probability as the average belief at the event level and by scaling the result to range from zero to one. The second experiment found that for source IP addresses involved in attacks, the algorithm was 100% accurate for probabilities  $\geq 0.8$ . For destination IP addresses involved in attacks, the algorithm was between 80% and 93% accurate for probabilities  $\geq 0.8$ . Finally, the third experiment used the results of the previous experiment to incorporate active learning, which allowed a maximum of 105 events to be labeled by the oracle, thus improving the accuracy to 100% for the destination IP address.



## Chapter 5

### Conclusions, Implications, Recommendations, and Summary

#### Conclusions

This research set out to address the problem that there was no approach to intrusion detection that reduced the workload of human analysts by providing a probabilistic prediction that a computer was experiencing a cyberattack. The goal for addressing this problem was to improve anomaly-based intrusion detection by adding meaning to alerts by using probabilistic clustering ensembles. By adding meaning to alerts, the desired outcome was to reduce the workload of security analysts.

Through the implementation of three experiments and the analysis of their results, five primary conclusions emerge. The first conclusion was that, as proposed, clustering ensembles provided multiple perspectives on the event data. These different perspectives were important in this research, since each partition only identified a single cluster of normal events. In some cases, the partition did not identify any normal events, but together the partitions provided sufficient observations to determine which computer systems were being attacked.

Second, for highly imbalanced datasets, which are characteristic for intrusion detection, clustering ensembles were effective in identifying certain normal events with a high degree of accuracy. This was likely because of the highly imbalanced nature of the input dataset. By identifying clusters that contained more than two standard deviations above the mean of events in each cluster, approximately 95% or more similar events from

the partition were represented in the cluster. Although normal events were scattered throughout the remaining clusters and mixed with attack events, the identification of a large population of normal events enabled them to be eliminated from the belief that they contained attacks.

The third conclusion of this research was that unsupervised intrusion detection did not require an accurate probability that an event was an attack. Prediction of events in this research resulted in a range of uncertainty, represented by a probability of 0.5, to a level of accurate identification of some normal events. Using the event-level prediction as a belief and aggregating that belief to the computer system level enabled prediction of computer systems under attack with 80% to 93% accuracy.

The fourth conclusion of this research was that active learning enabled a minimum level of interaction by human analysts while increasing accuracy to 100%. The algorithms developed in this research detected the computer systems experiencing attacks in a dataset with 700,001 events by requesting information from the oracle for 105 events. As a result, this form of anomaly detection combined with active learning may be effective for reducing the workload of human analysts in practice.

The fifth conclusion of this research was that the use of clustering ensembles for probabilistic intrusion detection, when combined with active learning, provided a highly accurate method for identifying computer systems that were experiencing a cyberattack. This method used unsupervised machine learning to identify the computer systems with the highest probability of an attack. It then used a minimal number of interactions with the oracle to accurately identify the affected systems.

## **Implications**

This research contributed to intrusion detection in several ways. First, the application of clustering ensembles to intrusion detection is a relatively new area of study. This research provided a new approach for anomaly-based intrusion detection that relied on the highly imbalanced nature of the data used to detect intrusions.

Next, this research found that combining uncertain event-level probabilities allowed the estimation of the probability that a computer system was under attack with reasonable accuracy. This was an important contribution to research, since it demonstrated that event-level detection does not need to be highly accurate to provide a higher level of meaning to alerts. This opens the possibility that combining alerts from existing intrusion detection methods may also be effective when aggregated at the computer system level.

This research also contributed to intrusion detection by contributing research that supports the use of a new dataset for intrusion detection – the UNSW-NB15 dataset. Since its release in 2015, it has been used in a growing number of research studies. This research further strengthens the support of a more contemporary dataset for intrusion detection and helps to better position the UNSW-NB15 to replace the outdated KDD Cup 1999 dataset.

Lastly, this research contributed to existing research in active learning for intrusion detection. At present, there does not appear to be research that applies active learning to use events to detect the probability of a computer system level attack. This contribution makes it possible to further reduce the workload of human analysts in reviewing alerts.

## **Recommendations**

This research laid a foundation for additional intrusion detection research. Three specific areas for future research should be considered. First, improvements in cluster partition evaluation will help to better qualify which partitions should be included in the anomaly detection. A better understanding of which partitions contribute to a good solution may help to improve the accuracy of anomaly detection at the event level, which will contribute to an improved calculation of the probability that a computer system is experiencing an attack. Second, this research should be extended to combine events from multiple security monitoring systems, such as host-based audit logs, signature-based alerts, and network flow data. Observations of potential attacks from these systems can be grouped using the computer system identifiers, such as IP addresses. As a result, this may enable improved CSA because of additional perspectives. Third, the experiments in this research combined observations at the event-level and grouped the results on the source and destination IP addresses. This approach may also be useful for combining event-level observations at the user-level, where user login names are provided within the event data. Future research should apply probabilistic clustering ensembles to insider threat detection to identify organization insiders who may pose a threat to the security of systems and data.

In addition, this research presents an approach that may be used to improve existing security monitoring practices in organizations. Security analysts generally respond to event-level security alerts. In many cases, these alerts provide insufficient information to determine the credibility, significance, and impact of the alert. Using the

results of this research, it is possible to provide a broader picture of CSA by focusing security analysts on the computer systems that are most likely being attacked. This change in focus will allow security analysts to more quickly determine a course of action without relying on their own observations to create a mental picture of what is occurring. In addition, the algorithms used in this research can be adapted to include additional datasets as well as to be verified in operational networks. Finally, this research should be applied to off-line analysis of network data to support the newly emerging practice of cyber threat hunting, in which security analysts examine various data sources to identify computer systems that may have become compromised but did not trigger alerts from regular monitoring systems (Sqrri Data, 2018).

## Summary

### *Introduction*

This research focused on the problem that there was no approach to intrusion detection that reduced the workload of human analysts by providing a probabilistic prediction that a computer is experiencing a cyberattack. Intrusion detection is the practice of examining information from computers and networks so that cyberattacks can be identified (Debar et al., 1999). Effective intrusion detection is important for organizations, since earlier detection of cyberattacks helps to reduce the impact and recovery costs (Ponemon Institute, 2016). Yet many intrusions are missed because of the volume of alerts that analysts must review, resulting in fatigue and errors in judgement (Julisch, 2003; Sawyer et al., 2014).

This research addressed several problems associated with intrusion detection. First, it addressed the high false-positive rates that accompany highly imbalanced data sets, where there are very few attacks scattered through large datasets of normal events. To address this problem, this research used observations of sparse attack events to predict the probability that a computer system was experiencing an attack. This was more accurate than predicting attacks at the event level. In addition, this research applied active learning, which allowed simulated human interaction to improve the overall accuracy. Second, this research addressed problems of overfitting and evasion. These are important problems, since machine learning algorithms that are overfitted are unable to find novel attacks and are not resilient to evasive adversarial tactics (Sommer & Paxson, 2010). Third, this research addressed the issue of using suitable datasets for evaluation of intrusion detection systems. The KDD Cup 1999 dataset has been the standard dataset for

evaluation since its inception, but it does not reflect current operating systems and does not contain identifiers of the source or destination computer systems involved in each event. This research addressed this problem by using the NSL-KDD dataset, to provide a connection to past research, as well as the UNSW-NB15 dataset, to provide a more contemporary view of network events.

This research was built upon a foundation of past research in intrusion detection. It applied CSA to establish a basis for deriving higher levels of meaning from intrusion alerts. It reviewed past machine learning approaches to intrusion detection research, especially probabilistic methods, to uncover the challenges and gaps in current research. It assessed the features and capabilities of clustering ensembles, with a focus on cluster evaluation. Finally, this research reviewed available intrusion detection datasets that had been developed and used in past research studies.

### *Methodology*

To address the problems associated with intrusion detection, this research implemented three experiments. The purpose of the experiments was to test the initial assumptions that clustering ensembles with probabilistic analysis and active learning may be effective for intrusion detection. The experiments were conducted using a prototype that was created using Python, *pandas*, and *scikit-learn*.

The first experiment evaluated cluster generation strategies and examined how to create diverse clustering results that could be used in subsequent experiments. The cluster generation approach used bagging to select a pseudorandom number and set of features for each partition of clustering results. Clusters were generated using the *k*-means

clustering algorithm. The generated clusters were evaluated to test the initial assumptions about the characteristics of what represented normal and attack events.

The second experiment implemented and tested an algorithm that used the partitions of clusters that were generated in the first experiment to predict the probability that an event was an anomaly. Next, the algorithm used the event-level probabilities to calculate the probability that a computer system was experiencing an attack. To test if this algorithm was effective, this experiment evaluated the accuracy of both the event-level and the computer system-level probabilities.

The third experiment added active learning, which allowed the simulation of human interaction, to fine tune the overall results. The active learning was implemented by selecting a random sampling of the event-level probabilities for the computer systems that had  $\geq 0.8$  probability of experiencing an attack. This experiment evaluated the minimum number of samples that could be used to achieve improvements in accuracy.

### *Results*

The first experiment found that using bagging and  $k$ -means clustering to generate a range of partitions with 40 to 100 clusters in each partition provided diverse results. This experiment found that the clustering results did not successfully identify anomalous events, but instead had higher accuracy in predicting the most normal clusters. By evaluating the mean number of events in each cluster of each partition, this experiment found that clusters that contained a number of events greater than or equal to two standard deviations above the mean number of events were consistently normal records. None of the events below this threshold were identified exclusively as attack events.



Using the results of the first experiment, the second experiment calculated the probability of an event being an anomaly from a range of 0 to 0.5. Thus, the event-level probability was more of a measure of uncertainty or belief. Next, it calculated the probability that a computer system was under attack by finding the mean of the event-level probabilities for each source and destination computer system. This experiment evaluated the event-level probabilities and determined that, above a midpoint threshold of votes from each partition, the prediction of normal events was highly accurate. Next, this experiment evaluated the accuracy of the probability that a computer system was experiencing an attack. This test was conducted 10 times, using a new bagging plan for each run. The accuracy of detection using the source IP address was 100% for all 10 runs. The accuracy of detection using the destination IP address ranged from 80% to 93%.

The third experiment evaluated the effect of adding active learning by selecting event-level samples for each of the computer systems that had probabilities of  $\geq 0.8$ . The sample sizes tested ranged from  $\sqrt{N}$  to  $^{10}\sqrt{N}$ . This experiment found that samples of  $^4\sqrt{N}$  consistently resulted in improved accuracy that a computer system was experiencing an attack. As a result, for the 700,001 events in the UNSW-NB15 dataset, sampling a total of 105 events resulted in 100% accuracy.

These results demonstrated that the use of clustering ensembles for probabilistic intrusion detection, when combined with active learning, provided a highly accurate method for identifying computer systems that were experiencing a cyberattack. The use of clustering ensembles provided multiple perspectives on the event data and enabled the prediction of attacks at the computer system-level by relying on the high-confidence normal events, even though the event-level prediction of anomalies was inaccurate.

### *Contributions and Future Work*

This research contributed to the field of intrusion detection by applying clustering ensembles to unsupervised anomaly detection. It is expected that this approach can be generalized to apply to other types of anomaly detection that are characterized by highly imbalanced datasets. Additional research should evaluate this in other applications, especially insider threat detection and cyber threat hunting.

Another important contribution of this research is that it demonstrated that combining uncertain event-level data to predict the probability that a computer system is experiencing an attack is highly accurate. This approach provided more meaning than individual events alone could provide and may be expanded in future research to combine alerts from a variety of intrusion detection systems as well as other security event monitoring systems. This will further reduce the workload of human analysts by creating a higher level of situational awareness.

This research also contributed to intrusion detection by applying a relatively new dataset, UNSW-NB15. As a result, this research strengthened support for a more contemporary dataset for intrusion detection. Future research should evaluate the use of probabilistic clustering ensembles with active learning to new datasets, as they are developed. In addition, future research should apply this approach using operational network data from an actual organization to validate that it can extend from research into practice.

Finally, this research added to previous studies related to reducing the human workload and resulting fatigue that are associated with security monitoring. It provided a

practical approach to apply unsupervised machine learning to prioritize the computer systems that are most suspicious, and it minimized the amount of human decision-making required. As a result, it may allow security analysts to more quickly determine a course of action when dealing with cyberattacks.

## **Appendices**

## **Appendix A**

### **Source Code Availability and Usage**

All the source code for the experiments in this research is available at the author's GitHub repository. It is available for researchers to enhance and extend this research in intrusion detection systems. This appendix describes how to obtain the source code, the package dependencies, configuration parameters, and execution instructions. Many of these instructions are specific to the Ubuntu operating system and may need to be adapted for other systems.

#### **Source Code**

The source code may be downloaded from GitHub at:

<https://github.com/stevenmcelwee/cepids>

From the command line, the source code may be cloned by:

```
git clone https://github.com/stevenmcelwee/cepids.git
```

#### **Package Dependencies**

##### *Python 2.7 and PIP*

The software used in this research was designed to work with Python 2.7. PIP was used to install additional packages. Python and PIP can be installed using:

```
sudo apt install python2.7 python-pip
```

### *scikit-learn*

scikit-learn is required for the KMeans class, which performs the clustering. It can be installed using:

```
pip install scikit-learn
```

### *pandas*

When installed as an operating system package, the pandas package satisfies several additional dependencies, such as numpy. This can be installed using:

```
sudo apt install python-pandas
```

## **Configuration Parameters**

The following parameters are configurable by updating variables at the beginning of the cepids.py file:

### *dataset\_file*

The filename of the input dataset. This must be either the absolute path to the file or relative to the directory from which the cepids.py package is executed. Example:

```
'datasets/derived/kdd_u2r_r2l.csv'.
```

### *dataset\_class*

The class name of the input dataset. This must be either UNSW or NSLKDD.

*num\_partitions*

The number of partitions that will be generated. This is used subsequently when evaluating the partitions as well. The recommended setting is: 100.

*min\_feature\_ratio*

The minimum number of features that will be used in the bagging plan for generating diverse clusters. The recommended setting is: 0.25.

*max\_feature\_ratio*

The maximum number of features that will be used in the bagging plan for generating diverse clusters. The recommended setting is: 0.75.

*min\_clusters*

The minimum number of clusters that will be created in each partition. The recommended setting is: 40.

*max\_clusters*

The maximum number of clusters that will be created in each partition. The recommended setting is: 100.

*input\_partition\_file*

Optional. The source file for a previously generated set of partitions. If a CSV was retained from Experiment 1, it can be used as the input for Experiment 2 to save time

and prevent recreation of partitions each time changes are made in the second experiment. If left blank, Experiment 1 will generate new partitions. If provided, it must contain either the absolute or relative path to the CSV partition file. Example:

'experiment1\_partitions\_unsw\_100p\_01.csv'.

#### *num\_stdev*

This is the number of standard deviations above the mean number of clusters that will be used as a threshold to determine the clusters that contain normal classes. The recommended setting is: 2.

#### *sample\_size\_exponent*

The exponent for creating the sample size for active learning. For example:  $\sqrt[4]{N} = N^{\frac{1}{4}}$ . To set the exponent, this can be set as a decimal value, or for readability, as an equation, such as: 1.0/4. Note that in Python, the decimal value of 1.0 is needed to prevent Python from truncating this to an integer. The recommended value is: 1.0/4.

#### *active\_learning\_output\_file*

The desired path to the results file to be created in Experiment 3. The output file is a CSV file that contains the srcip, dstip, P\_E, P\_A\_SRC, P\_A\_DST, and label for each event in the original dataset. Example: 'results/final\_output.csv'.



## Appendix B

### Dataset Descriptions

#### NSL-KDD

The derived version of the NSL-KDD dataset retains the characteristics of the NSL-KDD dataset, but removes the denial of service events. It includes only user-to-root attacks, remote-to-local attacks, and normal records is available at:

[https://github.com/stevenmcelwee/cepids/raw/master/datasets/derived/kdd\\_u2r\\_r2l.zip](https://github.com/stevenmcelwee/cepids/raw/master/datasets/derived/kdd_u2r_r2l.zip)

The data was created using a Python script that is available at:

[https://github.com/stevenmcelwee/cepids/blob/master/create\\_traces.py](https://github.com/stevenmcelwee/cepids/blob/master/create_traces.py)

The NSL-KDD dataset is composed of 41 attributes, a label, and a cluster ID that was created specifically for the NSL-KDD dataset. The table below shows the specific field names that are included as well as the datatypes.

Table 11

*NSL-KDD Attributes and Datatypes*

<b>Attribute</b>	<b>Datatype</b>	<b>Description</b>
duration	continuous	length (number of seconds) of the connection
protocol_type	symbolic	type of the protocol, e.g. tcp, udp, etc.
service	symbolic	network service on the destination, e.g., http, telnet, etc.
flag	symbolic	normal or error status of the connection
src_bytes	continuous	number of data bytes from source to destination
dst_bytes	continuous	number of data bytes from destination to source
land	continuous	1 if connection is from/to the same host/port; 0 otherwise
wrong_fragment	continuous	number of "wrong" fragments
urgent	continuous	number of urgent packets
hot	continuous	number of "hot" indicators
num_failed_logins	continuous	number of failed login attempts
logged_in	continuous	1 if successfully logged in; 0 otherwise
num_compromised	continuous	number of "compromised" conditions
root_shell	continuous	1 if root shell is obtained; 0 otherwise
su_attempted	continuous	1 if "su root" command attempted; 0 otherwise
num_root	continuous	number of "root" accesses
num_file_creations	continuous	number of file creation operations
num_shells	continuous	number of shell prompts
num_access_files	continuous	number of operations on access control files
num_outbound_cmds	continuous	number of outbound commands in an ftp session
is_host_login	continuous	1 if the login belongs to the "hot" list; 0 otherwise
is_guest_login	continuous	1 if the login is a "guest" login; 0 otherwise
count	continuous	number of connections to the same host as the current connection in the past two seconds
srv_count	continuous	number of connections to the same service as the current connection in the past two seconds
error_rate	continuous	% of connections that have "SYN" errors

Table 11

*NSL-KDD Attributes and Datatypes (cont.)*

<b>Attribute</b>	<b>Datatype</b>	<b>Description</b>
srv_serror_rate	continuous	% of connections that have "SYN" errors for same service connections
rerror_rate	continuous	% of connections that have "REJ" errors
srv_rerror_rate	continuous	% of connections that have "REJ" errors for same service connections
same_srv_rate	continuous	% of connections to the same service
diff_srv_rate	continuous	% of connections to different services
srv_diff_host_rate	continuous	% of connections to different hosts
dst_host_count	continuous	Number of connections having the same destination host IP address
dst_host_srv_count	continuous	Number of connections having the same port number
dst_host_same_srv_rate	continuous	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count
dst_host_diff_srv_rate	continuous	The percentage of connections that were to different services, among the connections aggregated in dst_host_count
dst_host_same_src_port_rate	continuous	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count
dst_host_srv_diff_host_rate	continuous	The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count
dst_host_serror_rate	continuous	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count
dst_host_srv_serror_rate	continuous	The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count
dst_host_rerror_rate	continuous	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count

Table 11

*NSL-KDD Attributes and Datatypes (cont.)*

<b>Attribute</b>	<b>Datatype</b>	<b>Description</b>
dst_host_srv_rerror_rate	continuous	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count
label	symbolic	Class of each event
cluster_id	symbolic	Integer cluster id that is representative of the class

**UNSW-NB15**

For the UNSW-NB15 dataset, the file UNSW-NB15\_1.csv was used for the experiments in this research. Table 12 shows the features, datatypes, and descriptions of this dataset. It is available from the original researchers at:

[https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/UNSW-NB15\\_1.csv](https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/UNSW-NB15_1.csv)

Table 12

*UNSW-NB15 Attributes and Datatypes*

<b>Name</b>	<b>Datatype</b>	<b>Description</b>
srcip	nominal	Source IP address
sport	integer	Source port number
dstip	nominal	Destination IP address
dsport	integer	Destination port number
proto	nominal	Transaction protocol
state	nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
dur	Float	Record total duration
sbytes	Integer	Source to destination transaction bytes
dbytes	Integer	Destination to source transaction bytes

Table 12

*UNSW-NB15 Dataset (cont.)*

<b>Name</b>	<b>Datatype</b>	<b>Description</b>
sttl	Integer	Source to destination time to live value
dttl	Integer	Destination to source time to live value
sloss	Integer	Source packets retransmitted or dropped
dloss	Integer	Destination packets retransmitted or dropped
service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service
sload	Float	Source bits per second
dload	Float	Destination bits per second
spkts	integer	Source to destination packet count
dpkts	integer	Destination to source packet count
swin	integer	Source TCP window advertisement value
dwin	integer	Destination TCP window advertisement value
stcpb	integer	Source TCP base sequence number
dtcpb	integer	Destination TCP base sequence number
smeansz	integer	Mean of the ?ow packet size transmitted by the src
dmeansz	integer	Mean of the ?ow packet size transmitted by the dst
trans_depth	integer	Represents the pipelined depth into the connection of http request/response transaction
res_bdy_len	integer	Actual uncompressed content size of the data transferred from the server's http service.
sjit	Float	Source jitter (mSec)
djit	Float	Destination jitter (mSec)
stime	Timestamp	record start time
ltime	Timestamp	record last time
sintpkt	Float	Source interpacket arrival time (mSec)
dintpkt	Float	Destination interpacket arrival time (mSec)
tcprtt	Float	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
synack	Float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
ackdat	Float	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
is_sm_ips_ports	Binary	If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0
ct_state_ttl	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and Post in http service.

Table 12

*UNSW-NB15 Dataset (cont.)*

<b>Name</b>	<b>Datatype</b>	<b>Description</b>
is_ftp_login	Binary	If the ftp session is accessed by user and password then 1 else 0.
ct_ftp_cmd	integer	No of flows that has a command in ftp session.
ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
ct_src_ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
attack_cat	nominal	The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
Label	binary	0 for normal and 1 for attack records

## Appendix C

### Python Package Versions

This research used Python 2.7 as well as packages at specific versions. Although the *cepids* package may function with newer versions of these packages, it was only tested with the versions shown in Table 13 below:

Table 13

*Python Package Versions*

<b>Package</b>	<b>Version</b>
dateutils	0.6.6
numpy	1.13.3
pandas	0.21.0
pip	9.0.1
python-dateutil	2.6.1
scikit-learn	0.19.1
scipy	1.0.0
setuptools	28.8.0

## Appendix D

### Detailed Anomaly Detection Results

This appendix provides the detailed results of Experiment 2 for each of the 10 experimental runs of the completed algorithm. Each run lists both the *srcip* and the *dstip* and their associated probabilities of experiencing an attack,  $P(A)$ . IP addresses with  $P(A) \geq 0.8$  are considered positive results. A results column indicates if the results are true positives (TP), true negatives (TN), or false positives (FP). There were no False negatives found in these experiments.



Table 14

*Experiment 2, Run 1 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000
175.45.176.3	TP	0.918
175.45.176.2	TP	0.876
175.45.176.0	TP	0.824
149.171.126.0	TN	0.659
149.171.126.4	TN	0.652
149.171.126.2	TN	0.649
149.171.126.9	TN	0.630
149.171.126.5	TN	0.624
149.171.126.8	TN	0.622
149.171.126.6	TN	0.622
149.171.126.3	TN	0.621
149.171.126.7	TN	0.620
149.171.126.1	TN	0.615
59.166.0.8	TN	0.543
59.166.0.9	TN	0.538
59.166.0.7	TN	0.527
59.166.0.5	TN	0.520
59.166.0.6	TN	0.513
59.166.0.1	TN	0.513
59.166.0.4	TN	0.511
59.166.0.0	TN	0.508
59.166.0.3	TN	0.507
59.166.0.2	TN	0.495
10.40.85.1	TN	0.296
149.171.126.18	TN	0.213
10.40.182.1	TN	0.175
149.171.126.15	TN	0.172
149.171.126.10	TN	0.169
149.171.126.19	TN	0.113
149.171.126.11	TN	0.103
149.171.126.16	TN	0.101
10.40.182.3	TN	0.082
10.40.170.2	TN	0.082
149.171.126.12	TN	0.074
192.168.241.243	TN	0.070

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
224.0.0.1	FP	1.000
149.171.126.18	TP	0.980
149.171.126.12	TP	0.962
224.0.0.5	FP	0.956
149.171.126.17	TP	0.918
149.171.126.13	TP	0.912
149.171.126.14	TP	0.907
149.171.126.11	TP	0.904
149.171.126.15	TP	0.898
10.40.170.2	FP	0.896
149.171.126.16	TP	0.894
149.171.126.10	TP	0.870
149.171.126.19	TP	0.869
10.40.182.3	FP	0.842
32.50.32.66	FP	0.841
175.45.176.1	FP	0.827
10.40.85.30	TN	0.772
192.168.241.243	TN	0.738
175.45.176.2	TN	0.677
59.166.0.9	TN	0.671
59.166.0.6	TN	0.645
59.166.0.3	TN	0.643
59.166.0.2	TN	0.642
59.166.0.8	TN	0.640
59.166.0.4	TN	0.638
59.166.0.0	TN	0.633
59.166.0.7	TN	0.632
59.166.0.5	TN	0.626
59.166.0.1	TN	0.622
10.40.85.1	TN	0.547
149.171.126.3	TN	0.527
149.171.126.9	TN	0.522
149.171.126.6	TN	0.508
149.171.126.0	TN	0.492
149.171.126.8	TN	0.492
149.171.126.1	TN	0.480

Table 14

*Experiment 2, Run 1 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.13	TN	0.069
10.40.85.30	TN	0.056
127.0.0.1	TN	0.043
149.171.126.17	TN	0.039

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.4	TN	0.475
175.45.176.3	TN	0.465
149.171.126.7	TN	0.461
149.171.126.2	TN	0.455
149.171.126.5	TN	0.448
175.45.176.0	TN	0.435
127.0.0.1	TN	0.372
10.40.198.10	TN	0.000

Table 15

*Experiment 2, Run 2 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	10.40.170.2	FP	1.000
175.45.176.3	TP	0.915	149.171.126.12	TP	0.992
175.45.176.2	TP	0.891	224.0.0.1	FP	0.965
175.45.176.0	TP	0.807	10.40.182.3	FP	0.953
149.171.126.4	TN	0.771	149.171.126.18	TP	0.948
149.171.126.0	TN	0.770	149.171.126.13	TP	0.920
149.171.126.2	TN	0.769	32.50.32.66	FP	0.910
149.171.126.8	TN	0.761	149.171.126.19	TP	0.908
149.171.126.6	TN	0.756	10.40.85.30	FP	0.893
149.171.126.5	TN	0.752	149.171.126.16	TP	0.875
149.171.126.7	TN	0.752	149.171.126.11	TP	0.872
149.171.126.9	TN	0.748	149.171.126.17	TP	0.869
149.171.126.1	TN	0.747	149.171.126.15	TP	0.868
149.171.126.3	TN	0.741	149.171.126.10	TP	0.864
59.166.0.7	TN	0.473	149.171.126.14	TP	0.844
59.166.0.5	TN	0.470	59.166.0.9	FP	0.815
59.166.0.2	TN	0.460	224.0.0.5	FP	0.813
59.166.0.4	TN	0.459	59.166.0.2	TN	0.789
59.166.0.3	TN	0.451	59.166.0.3	TN	0.780
59.166.0.8	TN	0.448	59.166.0.6	TN	0.778
59.166.0.0	TN	0.441	59.166.0.0	TN	0.771
59.166.0.1	TN	0.438	59.166.0.4	TN	0.768
59.166.0.9	TN	0.419	59.166.0.1	TN	0.766
59.166.0.6	TN	0.418	59.166.0.7	TN	0.759
10.40.85.1	TN	0.267	59.166.0.8	TN	0.758
10.40.182.1	TN	0.174	59.166.0.5	TN	0.747
149.171.126.18	TN	0.164	192.168.241.243	TN	0.619
149.171.126.15	TN	0.162	175.45.176.2	TN	0.572
149.171.126.10	TN	0.124	175.45.176.1	TN	0.538
149.171.126.11	TN	0.099	10.40.85.1	TN	0.520
149.171.126.16	TN	0.078	175.45.176.3	TN	0.466
149.171.126.19	TN	0.067	175.45.176.0	TN	0.361
10.40.182.3	TN	0.046	149.171.126.4	TN	0.292
10.40.170.2	TN	0.046	149.171.126.2	TN	0.288
149.171.126.12	TN	0.039	149.171.126.9	TN	0.277
192.168.241.243	TN	0.026	149.171.126.3	TN	0.261

Table 15

*Experiment 2, Run 2 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.13	TN	0.021
10.40.85.30	TN	0.020
149.171.126.17	TN	0.010
127.0.0.1	TN	0.000

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
10.40.198.10	TN	0.234
149.171.126.0	TN	0.193
149.171.126.5	TN	0.187
149.171.126.1	TN	0.180
149.171.126.8	TN	0.150
127.0.0.1	TN	0.138
149.171.126.6	TN	0.123
149.171.126.7	TN	0.000

Table 16

*Experiment 2, Run 3 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000
175.45.176.3	TP	0.919
175.45.176.2	TP	0.878
175.45.176.0	TP	0.838
149.171.126.0	TN	0.761
149.171.126.4	TN	0.753
149.171.126.2	TN	0.745
149.171.126.9	TN	0.727
149.171.126.3	TN	0.724
149.171.126.6	TN	0.724
149.171.126.1	TN	0.723
149.171.126.8	TN	0.723
149.171.126.5	TN	0.720
149.171.126.7	TN	0.716
59.166.0.7	TN	0.515
59.166.0.8	TN	0.494
59.166.0.3	TN	0.489
59.166.0.2	TN	0.488
59.166.0.9	TN	0.484
59.166.0.1	TN	0.472
59.166.0.4	TN	0.470
59.166.0.6	TN	0.467
59.166.0.5	TN	0.442
59.166.0.0	TN	0.430
10.40.85.1	TN	0.279
10.40.182.1	TN	0.180
149.171.126.10	TN	0.180
149.171.126.18	TN	0.177
149.171.126.15	TN	0.168
149.171.126.16	TN	0.118
149.171.126.19	TN	0.105
149.171.126.11	TN	0.092
149.171.126.13	TN	0.088
10.40.170.2	TN	0.083
10.40.182.3	TN	0.083
10.40.85.30	TN	0.075

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.13	TP	1.000
149.171.126.11	TP	1.000
149.171.126.18	TP	0.997
149.171.126.15	TP	0.994
149.171.126.17	TP	0.981
149.171.126.14	TP	0.981
10.40.170.2	FP	0.981
149.171.126.12	TP	0.972
149.171.126.19	TP	0.971
149.171.126.16	TP	0.967
149.171.126.10	TP	0.956
224.0.0.1	FP	0.945
10.40.182.3	FP	0.943
32.50.32.66	FP	0.898
10.40.85.30	FP	0.893
10.40.85.1	FP	0.814
59.166.0.9	FP	0.813
59.166.0.3	FP	0.804
59.166.0.6	TN	0.799
59.166.0.2	TN	0.798
59.166.0.4	TN	0.797
59.166.0.8	TN	0.792
59.166.0.0	TN	0.782
59.166.0.7	TN	0.770
59.166.0.5	TN	0.767
59.166.0.1	TN	0.759
175.45.176.2	TN	0.660
224.0.0.5	TN	0.653
175.45.176.1	TN	0.493
192.168.241.243	TN	0.441
175.45.176.3	TN	0.430
149.171.126.2	TN	0.365
149.171.126.9	TN	0.355
149.171.126.3	TN	0.349
127.0.0.1	TN	0.323
149.171.126.6	TN	0.320

Table 16

*Experiment 2, Run 3 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.12	TN	0.066
192.168.241.243	TN	0.055
149.171.126.17	TN	0.048
127.0.0.1	TN	0.048

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.8	TN	0.319
149.171.126.1	TN	0.310
149.171.126.4	TN	0.293
149.171.126.0	TN	0.282
149.171.126.7	TN	0.261
149.171.126.5	TN	0.225
10.40.198.10	TN	0.076
175.45.176.0	TN	0.000

Table 17

*Experiment 2, Run 4 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	149.171.126.12	TP	1.000
175.45.176.3	TP	0.920	149.171.126.14	TP	0.997
175.45.176.2	TP	0.907	149.171.126.15	TP	0.996
175.45.176.0	TP	0.842	149.171.126.11	TP	0.993
149.171.126.0	TN	0.622	149.171.126.13	TP	0.992
149.171.126.2	TN	0.621	149.171.126.19	TP	0.988
149.171.126.4	TN	0.619	149.171.126.16	TP	0.987
149.171.126.5	TN	0.609	149.171.126.10	TP	0.977
149.171.126.3	TN	0.608	149.171.126.17	TP	0.963
149.171.126.6	TN	0.608	224.0.0.1	FP	0.920
149.171.126.9	TN	0.604	149.171.126.18	TP	0.903
149.171.126.7	TN	0.601	10.40.170.2	FP	0.887
149.171.126.1	TN	0.600	32.50.32.66	FP	0.861
149.171.126.8	TN	0.597	10.40.182.3	TN	0.784
59.166.0.0	TN	0.465	10.40.85.30	TN	0.656
59.166.0.9	TN	0.462	59.166.0.9	TN	0.616
59.166.0.4	TN	0.459	59.166.0.8	TN	0.607
59.166.0.1	TN	0.459	59.166.0.2	TN	0.599
59.166.0.8	TN	0.440	224.0.0.5	TN	0.597
59.166.0.3	TN	0.420	59.166.0.6	TN	0.596
59.166.0.7	TN	0.412	59.166.0.1	TN	0.591
59.166.0.6	TN	0.409	59.166.0.7	TN	0.589
59.166.0.5	TN	0.389	59.166.0.0	TN	0.587
59.166.0.2	TN	0.376	59.166.0.3	TN	0.583
10.40.85.1	TN	0.231	59.166.0.4	TN	0.582
149.171.126.18	TN	0.174	59.166.0.5	TN	0.564
149.171.126.15	TN	0.169	175.45.176.1	TN	0.525
10.40.182.1	TN	0.160	10.40.85.1	TN	0.513
149.171.126.10	TN	0.158	175.45.176.2	TN	0.509
149.171.126.19	TN	0.110	192.168.241.243	TN	0.449
149.171.126.11	TN	0.090	149.171.126.1	TN	0.343
149.171.126.16	TN	0.086	149.171.126.5	TN	0.339
10.40.170.2	TN	0.078	175.45.176.0	TN	0.312
10.40.182.3	TN	0.078	149.171.126.0	TN	0.305
149.171.126.13	TN	0.066	149.171.126.7	TN	0.301
149.171.126.12	TN	0.061	149.171.126.6	TN	0.294

Table 17

*Experiment 2, Run 4 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
10.40.85.30	TN	0.056	149.171.126.8	TN	0.293
192.168.241.243	TN	0.053	149.171.126.9	TN	0.273
127.0.0.1	TN	0.026	175.45.176.3	TN	0.262
149.171.126.17	TN	0.022	149.171.126.3	TN	0.231
			149.171.126.2	TN	0.228
			149.171.126.4	TN	0.225
			10.40.198.10	TN	0.137
			127.0.0.1	TN	0.000



Table 18

*Experiment 2, Run 5 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	149.171.126.17	TP	1.000
175.45.176.3	TP	0.939	149.171.126.15	TP	0.998
175.45.176.2	TP	0.920	149.171.126.14	TP	0.987
175.45.176.0	TP	0.858	149.171.126.11	TP	0.986
149.171.126.0	TN	0.726	149.171.126.13	TP	0.978
149.171.126.4	TN	0.713	149.171.126.16	TP	0.972
149.171.126.2	TN	0.706	149.171.126.12	TP	0.963
149.171.126.9	TN	0.695	149.171.126.10	TP	0.945
149.171.126.5	TN	0.693	149.171.126.19	TP	0.901
149.171.126.3	TN	0.691	149.171.126.18	TP	0.876
149.171.126.6	TN	0.688	224.0.0.1	FP	0.866
149.171.126.7	TN	0.687	32.50.32.66	FP	0.814
149.171.126.1	TN	0.684	10.40.170.2	FP	0.807
149.171.126.8	TN	0.683	59.166.0.9	TN	0.745
59.166.0.5	TN	0.520	59.166.0.6	TN	0.717
59.166.0.4	TN	0.513	59.166.0.2	TN	0.717
59.166.0.1	TN	0.513	59.166.0.3	TN	0.716
59.166.0.7	TN	0.499	59.166.0.4	TN	0.715
59.166.0.3	TN	0.495	59.166.0.8	TN	0.715
59.166.0.6	TN	0.486	59.166.0.0	TN	0.706
59.166.0.9	TN	0.483	59.166.0.1	TN	0.693
59.166.0.8	TN	0.474	10.40.182.3	TN	0.686
59.166.0.2	TN	0.471	59.166.0.7	TN	0.683
59.166.0.0	TN	0.391	59.166.0.5	TN	0.683
10.40.85.1	TN	0.205	10.40.85.30	TN	0.534
149.171.126.18	TN	0.173	175.45.176.2	TN	0.524
149.171.126.10	TN	0.168	10.40.85.1	TN	0.445
149.171.126.15	TN	0.164	149.171.126.9	TN	0.414
10.40.182.1	TN	0.139	149.171.126.6	TN	0.412
149.171.126.16	TN	0.111	149.171.126.7	TN	0.397
149.171.126.11	TN	0.090	149.171.126.8	TN	0.395
149.171.126.13	TN	0.087	149.171.126.1	TN	0.373
10.40.170.2	TN	0.075	149.171.126.0	TN	0.362
10.40.182.3	TN	0.075	175.45.176.3	TN	0.362
149.171.126.19	TN	0.070	224.0.0.5	TN	0.360
149.171.126.12	TN	0.057	192.168.241.243	TN	0.341

Table 18

*Experiment 2, Run 5 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
10.40.85.30	TN	0.054	149.171.126.4	TN	0.327
192.168.241.243	TN	0.048	149.171.126.2	TN	0.323
149.171.126.17	TN	0.038	149.171.126.3	TN	0.320
127.0.0.1	TN	0.027	175.45.176.1	TN	0.300
			149.171.126.5	TN	0.236
			175.45.176.0	TN	0.132
			10.40.198.10	TN	0.060
			127.0.0.1	TN	0.000

Table 19

*Experiment 2, Run 6 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	224.0.0.1	FP	1.000
175.45.176.3	TP	0.926	149.171.126.16	TP	0.978
175.45.176.2	TP	0.906	149.171.126.11	TP	0.975
175.45.176.0	TP	0.836	149.171.126.14	TP	0.975
149.171.126.0	TN	0.737	149.171.126.15	TP	0.974
149.171.126.4	TN	0.722	149.171.126.17	TP	0.966
149.171.126.9	TN	0.721	149.171.126.13	TP	0.965
149.171.126.2	TN	0.718	32.50.32.66	FP	0.959
149.171.126.5	TN	0.716	10.40.170.2	FP	0.952
149.171.126.6	TN	0.713	149.171.126.12	TP	0.943
149.171.126.8	TN	0.710	149.171.126.10	TP	0.936
149.171.126.3	TN	0.708	149.171.126.19	TP	0.894
149.171.126.7	TN	0.706	149.171.126.18	TP	0.876
149.171.126.1	TN	0.700	10.40.182.3	FP	0.819
59.166.0.1	TN	0.541	59.166.0.9	TN	0.713
59.166.0.5	TN	0.525	59.166.0.6	TN	0.696
59.166.0.0	TN	0.519	59.166.0.2	TN	0.690
59.166.0.4	TN	0.515	59.166.0.8	TN	0.678
59.166.0.3	TN	0.506	224.0.0.5	TN	0.675
59.166.0.2	TN	0.499	59.166.0.3	TN	0.671
59.166.0.6	TN	0.498	192.168.241.243	TN	0.671
59.166.0.7	TN	0.498	59.166.0.4	TN	0.670
59.166.0.9	TN	0.483	10.40.85.30	TN	0.659
59.166.0.8	TN	0.468	59.166.0.0	TN	0.656
10.40.85.1	TN	0.268	59.166.0.1	TN	0.646
10.40.182.1	TN	0.188	59.166.0.5	TN	0.645
149.171.126.18	TN	0.185	59.166.0.7	TN	0.629
149.171.126.15	TN	0.185	10.40.85.1	TN	0.570
149.171.126.10	TN	0.163	175.45.176.2	TN	0.484
149.171.126.16	TN	0.115	175.45.176.1	TN	0.423
149.171.126.11	TN	0.113	175.45.176.3	TN	0.304
149.171.126.19	TN	0.096	127.0.0.1	TN	0.294
10.40.170.2	TN	0.083	149.171.126.6	TN	0.252
10.40.182.3	TN	0.083	149.171.126.3	TN	0.221
149.171.126.13	TN	0.076	149.171.126.7	TN	0.205
192.168.241.243	TN	0.072	149.171.126.0	TN	0.201

Table 19

*Experiment 2, Run 6 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
10.40.85.30	TN	0.067	149.171.126.4	TN	0.156
149.171.126.12	TN	0.066	149.171.126.8	TN	0.153
127.0.0.1	TN	0.056	149.171.126.2	TN	0.147
149.171.126.17	TN	0.052	149.171.126.9	TN	0.143
			149.171.126.5	TN	0.137
			149.171.126.1	TN	0.087
			10.40.198.10	TN	0.057
			175.45.176.0	TN	0.000

Table 20

*Experiment 2, Run 7 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	149.171.126.12	TP	1.000
175.45.176.3	TP	0.925	149.171.126.18	TP	0.975
175.45.176.2	TP	0.866	10.40.170.2	FP	0.964
175.45.176.0	TP	0.824	149.171.126.17	TP	0.956
149.171.126.0	TN	0.669	149.171.126.13	TP	0.949
149.171.126.4	TN	0.668	149.171.126.11	TP	0.945
149.171.126.2	TN	0.665	224.0.0.1	FP	0.937
149.171.126.6	TN	0.662	149.171.126.16	TP	0.934
149.171.126.3	TN	0.656	149.171.126.15	TP	0.926
149.171.126.5	TN	0.652	10.40.182.3	FP	0.897
149.171.126.9	TN	0.652	149.171.126.14	TP	0.889
149.171.126.7	TN	0.646	149.171.126.10	TP	0.885
149.171.126.1	TN	0.646	149.171.126.19	TP	0.865
149.171.126.8	TN	0.632	224.0.0.5	FP	0.824
59.166.0.1	TN	0.524	10.40.85.1	FP	0.822
59.166.0.8	TN	0.507	10.40.85.30	FP	0.812
59.166.0.6	TN	0.507	32.50.32.66	TN	0.750
59.166.0.2	TN	0.502	59.166.0.9	TN	0.706
59.166.0.5	TN	0.481	59.166.0.8	TN	0.690
59.166.0.7	TN	0.477	59.166.0.4	TN	0.682
59.166.0.4	TN	0.470	59.166.0.2	TN	0.678
59.166.0.9	TN	0.464	59.166.0.6	TN	0.672
59.166.0.0	TN	0.460	59.166.0.3	TN	0.668
59.166.0.3	TN	0.422	59.166.0.0	TN	0.660
10.40.85.1	TN	0.264	59.166.0.1	TN	0.660
149.171.126.18	TN	0.187	59.166.0.7	TN	0.652
149.171.126.10	TN	0.178	59.166.0.5	TN	0.635
10.40.182.1	TN	0.171	192.168.241.243	TN	0.617
149.171.126.15	TN	0.148	175.45.176.2	TN	0.598
149.171.126.16	TN	0.102	175.45.176.1	TN	0.557
149.171.126.19	TN	0.092	149.171.126.3	TN	0.499
10.40.182.3	TN	0.086	149.171.126.9	TN	0.459
10.40.170.2	TN	0.086	149.171.126.4	TN	0.453
149.171.126.11	TN	0.086	149.171.126.6	TN	0.445
149.171.126.13	TN	0.076	149.171.126.7	TN	0.429
10.40.85.30	TN	0.076	149.171.126.5	TN	0.410

Table 20

*Experiment 2, Run 7 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
192.168.241.243	TN	0.062	149.171.126.2	TN	0.400
149.171.126.12	TN	0.058	149.171.126.8	TN	0.399
149.171.126.17	TN	0.036	149.171.126.0	TN	0.390
127.0.0.1	TN	0.031	175.45.176.3	TN	0.387
			149.171.126.1	TN	0.332
			175.45.176.0	TN	0.257
			127.0.0.1	TN	0.169
			10.40.198.10	TN	0.000

Table 21

*Experiment 2, Run 8 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	10.40.170.2	FP	1.000
175.45.176.3	TP	0.913	149.171.126.12	TP	0.963
175.45.176.2	TP	0.877	149.171.126.19	TP	0.941
175.45.176.0	TP	0.833	149.171.126.14	TP	0.938
149.171.126.0	TN	0.761	149.171.126.17	TP	0.937
149.171.126.4	TN	0.736	149.171.126.16	TP	0.933
149.171.126.2	TN	0.730	149.171.126.11	TP	0.932
149.171.126.3	TN	0.714	149.171.126.13	TP	0.931
149.171.126.5	TN	0.711	149.171.126.15	TP	0.929
149.171.126.9	TN	0.711	224.0.0.1	FP	0.901
149.171.126.6	TN	0.710	149.171.126.10	TP	0.898
149.171.126.1	TN	0.706	149.171.126.18	TP	0.893
149.171.126.8	TN	0.706	32.50.32.66	FP	0.883
149.171.126.7	TN	0.702	10.40.182.3	FP	0.835
59.166.0.9	TN	0.578	59.166.0.9	FP	0.827
59.166.0.6	TN	0.573	59.166.0.3	FP	0.810
59.166.0.5	TN	0.557	59.166.0.8	FP	0.804
59.166.0.0	TN	0.548	59.166.0.6	FP	0.804
59.166.0.8	TN	0.540	59.166.0.4	FP	0.803
59.166.0.2	TN	0.537	59.166.0.2	TN	0.799
59.166.0.1	TN	0.531	59.166.0.0	TN	0.788
59.166.0.4	TN	0.528	59.166.0.5	TN	0.786
59.166.0.7	TN	0.527	59.166.0.7	TN	0.784
59.166.0.3	TN	0.514	59.166.0.1	TN	0.776
149.171.126.18	TN	0.190	149.171.126.7	TN	0.645
10.40.85.1	TN	0.186	149.171.126.8	TN	0.644
149.171.126.10	TN	0.181	175.45.176.1	TN	0.634
149.171.126.15	TN	0.146	10.40.85.30	TN	0.630
10.40.182.1	TN	0.108	149.171.126.9	TN	0.621
149.171.126.16	TN	0.102	149.171.126.3	TN	0.612
149.171.126.19	TN	0.095	175.45.176.2	TN	0.610
10.40.182.3	TN	0.089	149.171.126.6	TN	0.608
10.40.170.2	TN	0.089	149.171.126.2	TN	0.593
149.171.126.13	TN	0.081	149.171.126.5	TN	0.580
149.171.126.11	TN	0.081	149.171.126.1	TN	0.575
149.171.126.12	TN	0.059	149.171.126.4	TN	0.572

Table 21

*Experiment 2, Run 8 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
192.168.241.243	TN	0.043	149.171.126.0	TN	0.553
10.40.85.30	TN	0.041	224.0.0.5	TN	0.550
127.0.0.1	TN	0.041	192.168.241.243	TN	0.471
149.171.126.17	TN	0.023	10.40.85.1	TN	0.447
			175.45.176.3	TN	0.446
			127.0.0.1	TN	0.446
			175.45.176.0	TN	0.443
			10.40.198.10	TN	0.000



Table 22

*Experiment 2, Run 9 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	149.171.126.15	TP	1.000
175.45.176.3	TP	0.929	149.171.126.16	TP	0.998
175.45.176.2	TP	0.908	149.171.126.14	TP	0.997
175.45.176.0	TP	0.847	149.171.126.11	TP	0.996
149.171.126.0	TN	0.659	149.171.126.17	TP	0.985
149.171.126.2	TN	0.650	149.171.126.13	TP	0.981
149.171.126.4	TN	0.649	149.171.126.19	TP	0.980
149.171.126.6	TN	0.644	32.50.32.66	FP	0.962
149.171.126.5	TN	0.644	224.0.0.1	FP	0.929
149.171.126.9	TN	0.642	149.171.126.12	TP	0.927
149.171.126.3	TN	0.638	10.40.170.2	FP	0.912
149.171.126.8	TN	0.636	149.171.126.10	TP	0.907
149.171.126.7	TN	0.631	10.40.182.3	FP	0.824
149.171.126.1	TN	0.631	149.171.126.18	TP	0.819
59.166.0.0	TN	0.433	10.40.85.30	TN	0.707
59.166.0.3	TN	0.421	175.45.176.2	TN	0.656
59.166.0.4	TN	0.409	192.168.241.243	TN	0.600
59.166.0.7	TN	0.386	59.166.0.9	TN	0.598
59.166.0.2	TN	0.386	59.166.0.6	TN	0.595
59.166.0.6	TN	0.385	59.166.0.2	TN	0.594
59.166.0.9	TN	0.379	59.166.0.3	TN	0.588
59.166.0.8	TN	0.371	59.166.0.8	TN	0.583
59.166.0.1	TN	0.361	59.166.0.0	TN	0.576
59.166.0.5	TN	0.320	59.166.0.1	TN	0.572
10.40.85.1	TN	0.262	59.166.0.4	TN	0.571
149.171.126.15	TN	0.189	59.166.0.7	TN	0.555
149.171.126.18	TN	0.186	59.166.0.5	TN	0.551
10.40.182.1	TN	0.174	175.45.176.1	TN	0.495
149.171.126.10	TN	0.171	175.45.176.3	TN	0.485
149.171.126.16	TN	0.125	224.0.0.5	TN	0.467
149.171.126.19	TN	0.117	10.40.85.1	TN	0.378
149.171.126.11	TN	0.109	127.0.0.1	TN	0.197
149.171.126.13	TN	0.086	10.40.198.10	TN	0.182
10.40.182.3	TN	0.081	149.171.126.1	TN	0.135
10.40.170.2	TN	0.081	149.171.126.7	TN	0.106
149.171.126.12	TN	0.072	149.171.126.0	TN	0.088

Table 22

*Experiment 2, Run 9 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
192.168.241.243	TN	0.065
10.40.85.30	TN	0.054
127.0.0.1	TN	0.045
149.171.126.17	TN	0.041

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.4	TN	0.083
149.171.126.9	TN	0.074
149.171.126.5	TN	0.074
175.45.176.0	TN	0.069
149.171.126.2	TN	0.048
149.171.126.3	TN	0.021
149.171.126.8	TN	0.021
149.171.126.6	TN	0.000

Table 23

*Experiment 2, Run 10 Results*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>	<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
175.45.176.1	TP	1.000	10.40.170.2	FP	1.000
175.45.176.3	TP	0.914	149.171.126.12	TP	0.983
175.45.176.2	TP	0.887	149.171.126.14	TP	0.969
175.45.176.0	TP	0.839	149.171.126.13	TP	0.962
149.171.126.0	TN	0.703	149.171.126.15	TP	0.961
149.171.126.2	TN	0.695	149.171.126.17	TP	0.960
149.171.126.4	TN	0.694	149.171.126.19	TP	0.958
149.171.126.5	TN	0.688	149.171.126.11	TP	0.955
149.171.126.9	TN	0.687	149.171.126.18	TP	0.951
149.171.126.6	TN	0.684	224.0.0.1	FP	0.933
149.171.126.3	TN	0.684	149.171.126.16	TP	0.930
149.171.126.7	TN	0.682	149.171.126.10	TP	0.930
149.171.126.8	TN	0.681	10.40.182.3	FP	0.863
149.171.126.1	TN	0.671	32.50.32.66	FP	0.857
59.166.0.2	TN	0.593	10.40.85.30	TN	0.696
59.166.0.5	TN	0.582	59.166.0.9	TN	0.638
59.166.0.4	TN	0.581	59.166.0.6	TN	0.633
59.166.0.7	TN	0.577	59.166.0.2	TN	0.633
59.166.0.9	TN	0.569	59.166.0.3	TN	0.631
59.166.0.8	TN	0.568	59.166.0.8	TN	0.624
59.166.0.3	TN	0.566	59.166.0.0	TN	0.620
59.166.0.6	TN	0.559	59.166.0.4	TN	0.620
59.166.0.1	TN	0.556	175.45.176.0	TN	0.617
59.166.0.0	TN	0.498	175.45.176.2	TN	0.614
10.40.85.1	TN	0.282	59.166.0.1	TN	0.612
149.171.126.18	TN	0.195	59.166.0.5	TN	0.599
149.171.126.15	TN	0.183	59.166.0.7	TN	0.593
10.40.182.1	TN	0.178	224.0.0.5	TN	0.522
149.171.126.10	TN	0.162	10.40.85.1	TN	0.514
149.171.126.19	TN	0.107	149.171.126.0	TN	0.389
149.171.126.16	TN	0.104	149.171.126.2	TN	0.387
149.171.126.11	TN	0.096	149.171.126.5	TN	0.367
10.40.170.2	TN	0.085	149.171.126.3	TN	0.366
10.40.182.3	TN	0.085	149.171.126.9	TN	0.352
149.171.126.13	TN	0.077	149.171.126.1	TN	0.344
149.171.126.12	TN	0.071	149.171.126.4	TN	0.329

Table 23

*Experiment 2, Run 10 Results (cont.)*

<b>srcip</b>	<b>Result</b>	<b>P(A)</b>
10.40.85.30	TN	0.064
127.0.0.1	TN	0.051
149.171.126.17	TN	0.051
192.168.241.243	TN	0.051

<b>dstip</b>	<b>Result</b>	<b>P(A)</b>
149.171.126.6	TN	0.303
149.171.126.8	TN	0.300
149.171.126.7	TN	0.299
192.168.241.243	TN	0.226
127.0.0.1	TN	0.226
175.45.176.3	TN	0.207
175.45.176.1	TN	0.127
10.40.198.10	TN	0.000

## References

- Ahmad, I., Abdullah, A. B., & Alghamdi, A. S. (2009). Application of artificial neural network in detection of DOS attacks. *Proceedings of the 2nd International Conference on Security of Information and Networks*, 229-234.
- Ahrend, J. M., Jirotko, M., & Jones, K. (2016). On the collaborative practices of cyber threat intelligence analysts to develop and utilize tacit threat and defence knowledge on existing practices, shortcomings, system circumventions and implications for design. *2016 International Conference on Cyber Situational Awareness, Data Analytics and Assessment*, 1-10. IEEE.
- Al-Hamadi, H., & Chen, I. R. (2015). Integrated intrusion detection and tolerance in homogeneous clustered sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 11(3), 47.
- Ali, M. Q., & Al-Shaer, E. (2015). Randomization-based intrusion detection system for advanced metering infrastructure. *ACM Transactions on Information and System Security (TISSEC)*, 18(2), 7.
- Al-Mohannadi, H., Mirza, Q., Namanya, A., Awan, I., Cullen, A., & Disso, J. (2016). Cyber-attack modeling analysis techniques: An overview. *IEEE 4th International Conference on Future Internet of Things and Cloud Workshops*, 69-76.
- Analoui, M., & Sadighian, N. (2006). Solving cluster ensemble problems by correlation's matrix & GA. In: Shi Z., Shimohara, K., & Feng, D. (eds) *Intelligent Information Processing III. IIP 2006. IFIP International Federation for Information Processing*, 228, 227-231. Springer.
- Ayad, H. G., & Kamel, M. S. (2008). Cumulative voting consensus method for partitions with a variable number of clusters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(1), 160-173.
- Ayad, H. G., & Kamel, M. S. (2010). On voting-based consensus of cluster ensembles. *Pattern Recognition*, 43(5), 1943-1953.
- Azimi, J., & Fern, X. (2009). Adaptive cluster ensemble selection. *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, 992-997.

- Bakker, B., & Heskes, T. (2003). Clustering ensembles of neural network models. *Neural Networks, 16*(2), 261-269.
- Bamakan, S. M. H., Wang, H., & Shi, Y. (2017). Ramp loss K-Support Vector Classification-Regression; a robust and sparse multi-class approach to the intrusion detection problem. *Knowledge-Based Systems, 126*, 113-126.
- Bartnes, M., Moe, N. B., & Heegaard, P. E. (2016). The future of information security incident management training: A case study of electrical power companies. *Computers & Security, 61*, 32-45.
- Bhatt, P., Yano, E. T., Amorim, J., & Gustavsson, P. (2014). A cyber security situational awareness framework to track and project multistage cyber attacks. *Proceedings of the 9th International Conference on Cyber Warfare and Security*, 356-360.
- Brynielsson, J., Franke, U., & Varga, S. (2016). Cyber situational awareness testing. In B. Akhgar & B. Brewster (Eds.), *Combating Cybercrime and Cyberterrorism: Challenges, Trends and Priorities*, 209-233. Springer.
- Burroughs, D. J., Wilson, L. F., & Cybenko, G. V. (2002). Analysis of distributed intrusion detection systems using Bayesian methods. *21st IEEE International Performance, Computing, and Communications Conference*, 329-334.
- Cannady, J. (1998). Artificial neural networks for misuse detection. *National Information Systems Security Conference*, 368-381. NIST.
- Cao, V. L., Hoang, V. T., & Nguyen, Q. U. (2013). A scheme for building a dataset for intrusion detection systems. *2013 Third World Congress on Information and Communication Technologies*, 280-284.
- Chebrolu, S., Abraham, A., & Thomas, J. P. (2005). Feature deduction and ensemble design of intrusion detection systems. *Computers & Security, 24*(4), 295-307.
- Chen, W. H., Hsu, S. H., & Shen, H. P. (2005). Application of SVM and ANN for intrusion detection. *Computers & Operations Research, 32*(10), 2617-2634.
- Chivers, H., Clark, J. A., Nobles, P., Shaikh, S. A., & Chen, H. (2013). Knowing who to watch: Identifying attackers whose actions are hidden within false alarms and background noise. *Information Systems Frontiers, 15*(1), 17-34.
- Creech, G., & Hu, J. K. (2013). Generation of a new IDS test dataset: Time to retire the KDD collection. *IEEE Wireless Communications and Networking Conference*, 4487-4492.
- Daliran, M., Nassiri, R., & Latif-Shabgahi, G.-R. (2010). Using data analysis by deploying artificial neural networks to increase honeypot security. *Proceedings of the 6th International Conference on Networked Computing*, 1-4.

- Dasgupta, D., & González, F. (2002). An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6(3), 281-291.
- Dash, S. K., Reddy, K. S., & Pujari, A. K. (2011). Adaptive naive Bayes method for masquerade detection. *Security and Communication Networks*, 4(4), 410-417.
- Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8), 805-822.
- Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering*, 2, 222-232.
- Denning, D. E., & Neumann, P. G. (1985). *Requirements and model for IDES—a real-time intrusion detection expert system*. Document A005, SRI International, 333.
- Dimitriadou, E., Weingessel, A., & Hornik, K. (2001). Voting-merging: An ensemble method for clustering. *Proceedings of the Artificial Neural Networks-ICANN 2001*, 217-224.
- Domeniconi, C., & Al-Razgan, M. (2009). Weighted cluster ensembles: Methods and analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(4), 1-42.
- Domingos, P. (2015). *The master algorithm: How the quest for the ultimate learning machine will remake our world*. New York: Basic Books.
- Dubey, S., & Dubey, J. (2015). KBB: A hybrid method for intrusion detection. *IEEE International Conference on Computer, Communication, and Control*, 1-6.
- Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37(1), 32-64.
- Erbacher, R. F., Frincke, D. A., Wong, P. C., Moody, S., & Fink, G. (2010). A multi-phase network situational awareness cognitive task analysis. *Information Visualization*, 9(3), 204-219.
- Estivill-Castro, V. (2002). Why so many clustering algorithms: A position paper. *ACM SIGKDD Explorations Newsletter*, 4(1), 65-75.
- Feng, L., Guan, X. H., Guo, S. G., Gao, Y., & Liu, P. N. (2004). Predicting the intrusion intentions by observing system call sequences. *Computers & Security*, 23(3), 241-252.
- Fern, X. Z., & Brodley, C. E. (2004). Solving cluster ensemble problems by bipartite graph partitioning. *Proceedings of the Twenty-First International Conference on Machine Learning*, 36-42. ACM.

- Fontugne, R., Borgnat, P., Abry, P., & Fukuda, K. (2010). MAWILAB: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 1-12. ACM.
- Franke, U., & Brynielsson, J. (2014). Cyber situational awareness - A systematic review of the literature. *Computers & Security*, 46, 18-31.
- Fred, A. L. N., & Jain, A. K. (2005). Combining multiple clusterings using evidence accumulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6), 835-850.
- Frossyniotis, D., Likas, A., & Stafylopatis, A. (2004). A clustering method based on boosting. *Pattern Recognition Letters*, 25(6), 641-654.
- Gao, H. W., Zhu, D. J., & Wang, X. M. (2010). A parallel clustering ensemble algorithm for intrusion detection system. *Proceedings of the Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES 2010)*, 450-453.
- Gionis, A., Mannila, H., & Tsaparas, P. (2007). Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 341-352.
- Gogoi, P., Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2012). Packet and flow based network intrusion dataset. *Contemporary Computing*, 306, 322-334. Springer.
- Gowadia, V., Farkas, C., & Valtorta, M. (2005). Paid: A probabilistic agent-based intrusion detection system. *Computers & Security*, 24(7), 529-545.
- Gutzwiller, R. S., Hunt, S. M., & Lange, D. S. (2016). A task analysis toward characterizing cyber-cognitive situation awareness (CCSA) in cyber defense analysts. *2016 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, 14-20.
- Hadjitodorov, S. T., Kuncheva, L. I., & Todorova, L. P. (2006). Moderate diversity for better cluster ensembles. *Information Fusion*, 7(3), 264-275.
- Haider, W., Hu, J., Slay, J., Turnbull, B. P., & Xie, Y. (2017). Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*, 87, 185-192.
- Hajisalem, V., & Babaie, S. (2018). A hybrid intrusion detection system based on ABC-AFS algorithm for misuse and anomaly detection. *Computer Networks*, 136, 37-50.



- Han, S. J., & Cho, S. B. (2005). Evolutionary neural networks for anomaly detection based on the behavior of a program. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(3), 559-570.
- Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., & Wolber, D. (1990). A network security monitor. *1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 296-304.
- Helman, P., & Liepins, G. (1993). Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering*, 19(9), 886-901.
- Hou, S., Chen, L., Tas, E., Demihovskiy, I., & Ye, Y. (2015). Cluster-oriented ensemble classifiers for intelligent malware detection. *IEEE International Conference on Semantic Computing (ICSC)*, 189-196.
- Hu, W., Hu, W., Xie, N., & Maybank, S. (2009). Unsupervised active learning based on hierarchical graph-theoretic clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(5), 1147-1161.
- Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8), 651-666.
- Julisch, K. (2003). Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4), 443-471.
- Julisch, K., & Dacier, M. (2002). Mining intrusion detection alarms for actionable knowledge. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 366-375.
- Kamarudin, M. H., Maple, C., Watson, T., & Safa, N. S. (2017). A LogitBoost-based algorithm for detecting known and unknown web attacks. *IEEE Access*, 5, 26190-26200.
- Khan, L., Awad, M., & Thuraisingham, B. (2007). A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal – The International Journal on Very Large Data Bases*, 16(4), 507-521. ACM.
- Koc, L., Mazzuchi, T. A., & Sarkani, S. (2012). A network intrusion detection system based on a hidden naive Bayes multiclass classifier. *Expert Systems with Applications*, 39(18), 13492-13500.
- Kruegel, C., Mutz, D., Robertson, W., & Valeur, F. (2003). Bayesian event classification for intrusion detection. *Proceedings of the 19th Annual Computer Security Applications Conference*, 14-23. IEEE.
- Laskov, P., & Lippmann, R. (2010). Machine learning in adversarial environments. *Machine Learning*, 81(2), 115-119. Springer.

- Lazarevic, A., & Kumar, V. (2005). Feature bagging for outlier detection. *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 157-166.
- Li, J., Ou, X. M., & Rajagopalan, R. (2010). Uncertainty and risk management in cyber situational awareness. *Cyber Situational Awareness: Issues and Research*, 46, 51-68.
- Li, S. H., Kao, Y. C., Zhang, Z. C., Chuang, Y. P., & Yen, D. C. (2015). A network behavior-based botnet detection mechanism using PSO and k-means. *ACM Transactions on Management Information Systems (TMIS)*, 6(1), 3.
- Li, T., Ding, C., Jordon, M. I. (2007). Solving consensus and semi-supervised clustering problems using nonnegative matrix factorization. *Seventh IEEE International Conference on Data Mining, ICDM 2007*, 577-582.
- Lin, Y. D., Lin, P. C., Wang, S. H., Chen, I. W., & Lai, Y. C. (2016). PCAPLib: A system of extracting, classifying, and anonymizing real packet traces. *IEEE Systems Journal*, 10(2), 520-531.
- Lourenco, A., Buló, S. R., Rebagliati, N., Fred, A. L. N., Figueiredo, M. A. T., & Pelillo, M. (2015). Probabilistic consensus clustering using evidence accumulation. *Machine Learning*, 98(1-2), 331-357.
- Lunt, T. F. (1990). IDES: An intelligent system for detecting intruders. *Proceedings of the Symposium: Computer Security, Threat and Countermeasures*, 30-45.
- Luo, H., Jing, F., & Xie, X. (2006). Combining multiple clustering using information theory based genetic algorithm. *IEEE International Conference on Computational Intelligence and Security*, 84-89.
- McElwee, S. (2017). Active learning intrusion detection using k-means clustering selection. *IEEE SoutheastCon*, 1-7.
- McElwee, S., & Cannady, J. (2016). Improving the performance of self-organizing maps for intrusion detection. *IEEE SoutheastCon*, 1-6.
- McElwee, S., Heaton, J., Fraley, J., Cannady, J. (2017). Deep learning for prioritizing and responding to intrusion detection alerts. *MILCOM*, 1-5. IEEE.
- McGrayne, S. (2014). The theory that never died: How an eighteenth century mathematical idea transformed the twenty-first century. *Mètode Science Studies Journal - Annual Review*, 0(5), 159-165.
- Meilă, M. (2007). Comparing clusterings – an information based distance. *Journal of Multivariate Analysis*, 98, 873-895.

- Milenkoski, A., Vieira, M., Kounev, S., Avritzer, A., & Payne, B. D. (2015). Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys*, 48(1), 12.
- Miller, B., Kantchelian, A., Afroz, S., Bachwani, R., Dauber, E., Huang, L., Tschantz, M. C., Joseph, A. D., Tygar, J. D. (2014). Adversarial active learning. *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, 3-14. ACM.
- Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *Military Communications and Information Systems Conference (MilCIS)*, 1-6. IEEE.
- Mukherjee, B., Heberlein, L. T., & Levitt, K. N. (1994). Network intrusion detection. *IEEE Network*, 8(3), 26-41.
- Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. *Proceedings of the 2002 International Joint Conference on Neural Networks*, 2(1), 1702-1707. IEEE.
- Newcomb, E. A., Hammell, R. J., & Hutchinson, S. (2016). Effective prioritization of network intrusion alerts to enhance situational awareness. *IEEE International Conference on Intelligence and Security Informatics: Cybersecurity and Big Data*, 73-78.
- Orfila, A., Tapiador, J. M. E., & Ribagorda, A. (2009). Trends, problems and misconceptions in testing network intrusion detection systems' effectiveness. In R. D. Hopkins & W. P. Tokere (Eds.), *Computer Security: Intrusion, Detection and Prevention*, 51-62. Hauppauge: Nova Science Publishers, Inc.
- Pajouh, H. H., Dastghaibifard, G., & Hashemi, S. (2017). Two-tier network anomaly detection model: A machine learning approach. *Journal of Intelligent Information Systems*, 48(1), 61-74.
- Papamartzivanos, D., Mármol, F. G., & Kambourakis, G. (2018). Dendron: Genetic trees driven rule induction for network intrusion detection systems. *Future Generation Computer Systems*, 79, 558-574.
- Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., & Lee, W. (2009). MCPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6), 864-881.
- Perona, I., Gurrutxaga, I., Arbelaitz, O., Martín, J. I., Muguerza, J., & Pérez, J. M. (2008). Service-independent payload analysis to improve intrusion detection in network traffic. *Proceedings of the 7th Australasian Data Mining Conference*, 87, 171-178. Australian Computer Society.
- Ponemon Institute. (2016). *2016 Cost of Data Breach Study: Global Analysis*. IBM. Retrieved from <http://www.ibm.com/security/data-breach>

- Punera, K., & Ghosh, J. (2008). Consensus-based ensembles of soft clusterings. *Applied Artificial Intelligence*, 22(7-8), 780-810.
- Qian, J., Xu, C., & Shi, M. L. (2006). Redesign and implementation of evaluation dataset for intrusion detection system. *Emerging Trends in Information and Communication Security*, 3995, 451-465.
- Rajivan, P., & Cooke, N. (2017). Impact of team collaboration on cybersecurity situational awareness. In *Theory and Models for Cyber Situation Awareness*, 203-226. Springer.
- Rhodes, B. C., Mahaffey, J. A., & Cannady, J. D. (2000). Multiple self-organizing maps for intrusion detection. *Proceedings of the 23rd National Information Systems Security Conference*, 16-19. NIST.
- Saeed, A., Ahmadiania, A., Javed, A., & Larijani, H. (2016). Intelligent intrusion detection in low-power IoTs. *ACM Transactions on Internet Technology (TOIT)*, 16(4), 27.
- Salem, M., Reissmann, S., Buehler, U. (2014). Persistent dataset generation using real-time operative framework. *International Conference on Computing, Networking and Communications (ICNC)*, 1023-1027. IEEE.
- Sawyer, B., D., Finomore, V. S., Funke, G. J., Mancuso, V.F., Funke, M.E., Matthews, G., & Warm, J. S. (2014). Cyber vigilance: Effects of signal probability and event rate. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 58(1), 1771-1775. SAGE Publications.
- Scott, S. L. (2004). A bayesian paradigm for designing intrusion detection systems. *Computational Statistics & Data Analysis*, 45(1), 69-83.
- Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3), 357-374.
- Silva, J. D. A., & Hruschka, E. R. (2016). A support system for clustering data streams with a variable number of clusters. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 11(2), 11.
- Sinclair, C., Pierce, L., & Matzner, S. (1999). An application of machine learning to network intrusion detection. *Proceedings of the 15th Annual Computer Security Applications Conference, ACSAC'99*, 371-377. IEEE.
- Sindhu, S. S. S., Geetha, S., & Kannan, A. (2012). Decision tree based light weight intrusion detection using a wrapper approach. *Expert Systems with Applications*, 39(1), 129-141.

- Singh, R., Kumar, H., & Singla, R. K. (2015). A reference dataset for network traffic activity based intrusion detection system. *International Journal of Computers Communications & Control*, 10(3), 390-402.
- Sommer, R., & Paxson, V. (2003). Enhancing byte-level network intrusion detection signatures with context. *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 262-271.
- Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. *IEEE Symposium on Security and Privacy*, 305-316.
- Song, D., Heywood, M. I., & Zincir-Heywood, A. N. (2005). Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3), 225-239.
- Sqrrl Data. (2018). *A Framework for Cyber Threat Hunting*. Retrieved from: <https://sqrrl.com/media/Framework-for-Threat-Hunting-Whitepaper-web.pdf>.
- Šrndić, N., & Laskov, P. (2014). Practical evasion of a learning-based classifier: A case study. *IEEE Symposium on Security and Privacy (SP)*, 197-211.
- Stolfo, S. J., Fan, W., Lee, W., Prodromidis, A., & Chan, P. K. (2000). Cost-based modeling for fraud and intrusion detection: Results from the jam project. *Proceedings of the DARPA Information Survivability Conference and Exposition, 2*, 130-144. IEEE.
- Stopel, D., Boger, Z., Moskovitch, R., Shahar, Y., & Elovici, Y. (2006). Application of artificial neural networks techniques to computer worm detection. *International Joint Conference on Neural Networks, 2006*, 2362-2369.
- Strehl, A. & Ghosh, J. (2002). Cluster ensembles – A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3, 583-617.
- Swarnkar, M., & Hubballi, N. (2016). OCPAD: One class naive Bayes classifier for payload based anomaly detection. *Expert Systems with Applications*, 64, 330-339.
- Tadda, G. P., & Salerno, J. S. (2010). Overview of cyber situational awareness. *Cyber Situational Awareness: Issues and Research*, 46, 15-35.
- Tavallae, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD Cup 99 data set. *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 1-6.
- Toosi, A. N., & Kahani, M. (2007). A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Computer Communications*, 30(10), 2201-2212.

- Topchy, A. P., Law, M. H. C., Jain, A. K., & Fred, A. L. (2004). Analysis of consensus partition in cluster ensemble. *Proceedings of the Fourth IEEE International Conference on Data Mining*, 225-232.
- Topchy, A., Jain, A. K., & Punch, W. (2005). Clustering ensembles: Models of consensus and weak partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12), 1866-1881.
- Tsang, I. W., Kwok, J. T., & Cheung, P. M. (2005). Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6, 363-392.
- Tumer, K., & Agogino, A. K. (2008). Ensemble clustering with voting active clusters. *Pattern Recognition Letters*, 29(14), 1947-1953.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236), 433-460.
- Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, 185(4157), 1124-1131.
- Tylman, W. (2008). Anomaly-based intrusion detection using Bayesian networks. *3rd International Conference on Dependability of Computer Systems*, 211-218. IEEE.
- Valdes, A., & Skinner, K. (2000). Adaptive, model-based monitoring for cyber attack detection. *Recent Advances in Intrusion Detection*, 80-93. Springer.
- Vasilomanolakis, E., Cordero, C. G., Milanov, N., & Mühlhäuser, M. (2016). Towards the creation of synthetic, yet realistic, intrusion detection datasets. *IEEE/IFIP Network Operations and Management Symposium*, 1209-1214.
- Vasilomanolakis, E., Karuppayah, S., Mühlhäuser, M., & Fischer, M. (2015). Taxonomy and survey of collaborative intrusion detection. *ACM Computing Surveys*, 47(4), 55.
- Vega-Pons, S., & Ruiz-Shulcloper, J. (2011). A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(3), 337-372.
- Vega-Pons, S., Correa-Morris, J., & Ruiz-Shulcloper, J. (2008). Weighted cluster ensemble using a kernel consensus function. *Progress in Pattern Recognition, Image Analysis and Applications*, 195-202. Springer.
- Vega-Pons, S., Correa-Morris, J., & Ruiz-Shulcloper, J. (2010). Weighted partition consensus via kernels. *Pattern Recognition*, 43(8), 2712-2724.
- Wang, G., Wang, T., Zheng, H., & Zhao, B. Y. (2014). Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. *23rd USENIX Security Symposium*, 239-254.

- Wang, X., Yang, C., & Zhou, J. (2009). Clustering aggregation by probability accumulation. *Pattern Recognition*, 42(5), 668-675.
- Weng, F. F., Jiang, Q. S., Shi, L., & Wu, N. (2007). An intrusion detection system based on the clustering ensemble. *2007 International Workshop on Anti-Counterfeiting, Security, and Identification*, 121-124.
- Wheelus, C., Khoshgoftaar, T. M., Zuech, R., & Najafabadi, M. M. (2014). A session based approach for aggregating network traffic data - the SANTA dataset. *IEEE International Conference on Bioinformatics and Bioengineering*, 369-378.
- Xiao, L. Y., Chen, Y. T., & Chang, C. K. (2014). Bayesian model averaging of Bayesian network classifiers for intrusion detection. *38th Annual IEEE International Computer Software and Applications Conference Workshops*, 128-133.
- Xu, J., & Shelton, C. R. (2010). Intrusion detection using continuous time bayesian networks. *Journal of Artificial Intelligence Research*, 39, 745-774.
- Yassin, W., Udzir, N. I., Muda, Z., & Sulaiman, M. N. (2013). Anomaly-based intrusion detection through k-means clustering and naives Bayes classification. *4th International Conference on Computing and Informatics*, 298-303.
- Yoon, H. S., Ahn, S. Y., Lee, S. H., Cho, S. B., & Kim, J. H. (2006). Heterogeneous clustering ensemble method for combining different cluster results. *BioDM 2006, LNBI, 3916*, 82-92.
- Zhang, J., Zulkernine, M., & Haque, A. (2008). Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5), 649-659.
- Zhong, C., Yen, J., Liu, P., Erbacher, R. F., Garneau, C., & Chen, B. (2017). Studying analysts' data triage operations in cyber defense situational analysis. In *Theory and Models for Cyber Situation Awareness*, 128-169. Springer.
- Zhou, C. V., Leckie, C., & Karunasekera, S. (2010). A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security*, 29(1), 124-140.
- Zhou, Z. H., & Tang, W. (2006). Clusterer ensemble. *Knowledge-Based Systems*, 19(1), 77-83.