METHODS TO ADDRESS EXTREME CLASS IMBALANCE IN MACHINE LEARNING BASED NETWORK INTRUSION DETECTION SYSTEMS

THESIS

Russell W. Walter, MAJ, USA

AFIT-ENS-MS-16-M-131

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-16-M-131

METHODS TO ADDRESS EXTREME CLASS IMBALANCE IN MACHINE LEARNING BASED NETWORK INTRUSION DETECTION SYSTEMS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Russell W. Walter

MAJ, USA

March 2016

AFIT-ENS-MS-16-M-131

METHODS TO ADDRESS EXTREME CLASS IMBALANCE IN MACHINE LEARNING BASED NETWORK INTRUSION DETECTION SYSTEMS

Russell W. Walter

Major, USA

Committee Membership:

Dr. Kenneth W. Bauer, Jr.
Chair

Dr. Trevor J. Bihl
Member

Abstract

Despite the considerable academic interest in using machine learning methods to detect cyber attacks and malicious network traffic, there is little evidence that modern organizations employ such systems. Due to the targeted nature of attacks and cybercriminals' constantly changing behavior, valid observations of attack traffic suitable for training a classifier are extremely rare. Rare positive cases combined with the fact that the overwhelming majority of network traffic is benign create an extreme class imbalance problem. Using publically available datasets, this research examines the class imbalance problem by using small samples of the attack observations to create multiple training sets that reflect a realistic class imbalance. A variety of techniques to alleviate the imbalance are examined including under sampling the majority class and three techniques to over sample the minority attack observations by creating new synthetic observations. We test these methods on four of the most popular machine learning classifiers. We examine two single model classifiers, artificial neural networks and support vector machines, and two ensemble methods, gradient boosting and random forests. We find that under sampling generally outperforms oversampling techniques and that the ensemble methods both outperform single models. We show that the apparent superiority of the ensemble methods may be illusory due to the "laboratory conditions" of using well-crafted public datasets. By introducing an element of noise into the training data, we show that neural networks' robustness to noise make it the preferred approach in real world settings where the more sophisticated ensemble methods fail. We also present a technique where neural networks are used to select features from the noisy dataset that improve the performance of random forests and gradient boosting allowing for the creation of an improved ensemble classifier.

This work is dedicated to my wife, daughter, and mother.

Acknowledgments

I would like to express my sincere thanks to my advisor, Dr. Bauer, for his wise counsel and for allowing me to pursue a non-traditional topic. I'd also like thank Dr. Bihl for his exceptional attention to detail. I also wish to thank Lieutenant Colonel Brian Lunday and Colonel Lynette Arnhart for their endorsements of my ability to succeed at AFIT.

Russell W. Walter

Table of Contents

List of Figures

List of Tables

METHODS TO ADDRESS EXTREME CLASS IMBALANCE IN MACHINE LEARNING BASED NETWORK INTRUSION DETECTION SYSTEMS

I. Introduction

1.1 Motivation

After years of increased cyber-attacks against US Department of Defense (DoD) networks, the DoD created US Cyber Command (CYBERCOM) in 2010 to defend the department's cyber infrastructure which exists in over 15,000 computer networks across 400 military bases in 88 countries [1]. The DoD and CYBERCOM, like most private organizations, rely on Intrusion Detection Systems (IDS) to detect the presence of adversaries attacking a network or attempting to steal sensitive information. Historically, IDSs have relied on signature based methods where network traffic is compared against a database of known threats. This requires a non-trivial amount of human involvement in the system to constantly update signatures and inspect traffic. As the amount of data flowing over networks grew in the late 1990s, interest in using statistical anomaly based detection systems that rely on machine learning algorithms to detect traffic autonomously grew in popularity [2]. In the early 2000s, extensive academic research examined a variety of machine learning techniques to improve IDSs but currently there is little evidence that many organizations actually deploy such systems [3]. One of the principal shortcomings of this research has been the use of large samples of synthetic data to train and evaluate classifiers.

Due to concerns about revealing network vulnerabilities and privacy concerns from sharing sensitive information, real world data on attacks and malicious network traffic has been largely unavailable to academic researchers. Researchers typically used synthetic data created in cyber labs that could be made publically available. While a classifier built with such data could

not be employed by any individual organization, the data provided a much needed way for researchers to compare methods and benchmark results. Researchers have pointed out the many shortcoming of the recent academic research [3, 4], one of which is the unrealistic sample sizes often used in the data. Actual network security professionals are not able to create newer examples of cyber-attacks or malware infections to use as training data. Due to the targeted nature of most attacks and hackers' constantly changing behavior to avoid detection, real world samples are extremely rare [5]. This shortage and the fact that the overwhelming majority of network traffic is benign mean that security professionals using machine learning techniques are faced with a problem of extreme class imbalance. Classifiers will tend to misclassify minority observations if they are overwhelmed in number by the majority class. In their Ziegel Award winning book[6], *Applied Predictive Modeling* [7], Kuhn and Johnson devote an entire chapter to the class imbalance problem calling it, "one of the toughest problems in predictive modeling."

1.2 Contributions

The first contribution of this research is an examination of the effects of extreme class imbalance common in IDS data on popular machine learning classifiers using improved datasets that reflect modern cyberattack traffic. The second contribution is an assessment of some of the common techniques used to address class imbalance, namely under sampling the majority class and various techniques to over sample the minority class to create an artificial balance. Two public datasets are used but instead of large balanced samples, we randomly sample from the larger datasets to create multiple datasets with realistic imbalances among normal and malicious traffic observations. These samples allow for the use of common statistical tests to measure performance between both sampling techniques and between classifiers. The third contribution is

an examination of the most popular classifiers to identify the most promising classifiers for implementation in an intrusion detection system.

1.3 Organization

In the second chapter, we present a literature review of published research related to network intrusion detection, machine learning techniques for handling unbalanced data, and a review of methods that artificially increase the number of minority observations. Chapter 3 details the overall methodology including a description of the datasets, the classification algorithms, and the methodology for the experiment. Chapter 4 presents the analysis of the results. Chapter 5 offers conclusions and recommendations for further research on class imbalanced training in the cyber domain.

II. Literature Review

2.1. Chapter Overview

A review of literature on machine learning in intrusion detection systems (IDS) reveals an extensive body of literature but also a number of scathing reviews of much of that literature. Academic researchers and network security professionals alike have questioned the utility of many of the published articles questioning the data, methodology, and the veracity of the results. Section 2.2 will outline the published literature as well as its criticism. Much of the recent research on intrusion detection systems occurs within industry and due to proprietary trade secrets, is largely unavailable to academic research. We briefly touch on some current industry trends in chapter 4 when we discuss ensembles and feature selection.

A class imbalance in the training data tends to adversely affect accuracy as most models become biased towards the majority class. Section 2.3 will present the most common methods to address class imbalance. Sections 2.4 deals with research directly related to experiments measuring the performance of the methods for addressing class imbalance. Several published articles have compared and contrasted techniques to address class imbalance [51, 53, 55, 57:59] not unlike this research. Section 2.4 will review that research with an emphasis on the differences from this thesis.

2.2. Intrusion Detection Systems

Concern for network intrusion first surfaced in the Department of Defense (DoD) in the 1970s when network security pioneer James Anderson published a study commissioned by the United States Air Force (USAF) titled, “Computer Security Threat Monitoring and Surveillance” [8]. Many credit the study with creating the foundation for an automated IDS. In 1987, Denning created the Intrusion Detection Expert System (IDES) for the USAF that was one of the first

signature based IDSs [9]. A signature based IDS constantly scans networks and compares network traffic against a database of known threats. Such systems were the standard for years but the explosion in network traffic in the 1990s made scanning and updating known threat signatures increasingly cumbersome. The variety of attacks also presented a challenge as the IDS could only detect known threats making them vulnerable to novel attacks. This brought about interest in using machine learning techniques to create an anomaly based IDS. An anomaly based IDS classifies network traffic as normal or suspicious by comparing the traffic to an established baseline. These techniques showed much promise but were initially problematic due to the large number of false positives they generated. This led to much academic interest and an explosion of publications using machine learning to identify malicious network traffic.

Much of the popularity of machine learning in IDS research can be traced to the creation of popular datasets made publically available in 1999 [10]. In 1998, researchers from Massachusetts Institute of Technology's (MIT) Lincoln Labs, commissioned and funded by the Defense Advanced Research Project Agency (DARPA) and in cooperation with the Air Force Research Laboratory (AFRL), created a network of workstations intended to simulate the traffic at a medium sized Air Force base [11]. Researchers then attacked the notional network using common attack techniques at the time. The team collected data from the network traffic using log files and packet capture software. These files contained raw measurements of packet sizes, timing of traffic flows, and other significant information about the traffic. Using these files, researchers processed the data into four million network traffic observations with 41 features. DARPA made the data available to the public in the 1999 for the annual Knowledge Discovery and Datamining (KDD) Cup, a machine learning competition organized by the Association for

Computing Machinery's (ACM) Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD).

The 1999 KDD dataset, as it would come to be known, quickly became the standard benchmark for researchers in the field of anomaly based IDS. A survey on anomaly based IDS publications using Google Scholar and the Digital Bibliography and Library Project (DBLP) found that of the 163 publications on the subject between 2000 and 2008, 52% of them used either the 1999 KDD dataset or datasets created with the raw files from the 1998 DARPA experiment [4]. The published works used a wide variety of classification algorithms and portions of the available data. Table 1 shows sixteen popularly cited journal articles using the 1999 KDD data during that period.

Table 1: Summary of machine learning IDS research using the 1999 KDD Dataset

Author(s)	Year	Classifier(s)
Jirapummin [12]	2002	Hybrid Neural Network
Zhang et al. [13]	2003	C4.5, Neural Networks
Moradi and Zulkernine [14]	2004	Neural Networks
Xu and Wang [15]	2005	SVM with PCA
Pan et al. [16]	2005	Neural Networks, Rule Based
Peddabachigari [17]	2005	SVM, Decision Trees
Yao [18]	2005	Fuzzy SVM
Depren [19]	2005	Self-Organizing Maps
Zhang and Zulkerine [20]	2005	Random Forest
Yao [21]	2006	SVM
Bouzida [22]	2006	Decision trees
Panda and Patra [23]	2007	Naïve Bayes
Katos [24]	2007	Discriminant analysis, clustering
Hu et al. [25]	2008	AdaBoost
Nguyen [26]	2008	Multiple
Panda and Patra [27]	2009	Semi-Naïve Bayes

Despite the immense popularity of the dataset, it has suffered from much criticism since its creation. The year after its release McHugh [28] pointed out that no validation had been done to ensure the simulated traffic looked at all like real traffic noting the lack of variation in the normal traffic one would expect in a medium sized network. McHugh also questioned the distribution of attack types noting that reconnaissance and probing traffic would be much more common than the less subtle Denial of Service attacks (DoS), yet the attack types are equally

proportional in the data. In 2003, Mahoney and Chan noted that in one feature, the time-to live or TTL, the observations in the training set always had a value of 126 or 253 while the test set only had values of 127 or 254 creating a completely useless feature [29]. Others found redundancies in the original KDD data where observations were duplicated in the training and testing sets [30]. By 2007, the data had fallen so far from grace that Brugger and Chow noted that many of their peers confessed that if asked to peer review a paper using the dataset, they would reject it out of hand [31].

Despite the well-known problems of the data, journals continued publishing articles from experiments using the dataset as late as 2010 [32]. A second version attempting to correct some of the redundancies of the original appeared in 2009 titled the NSL-KDD dataset but had some of the same problems as the original namely an unrealistic balance between the normal and malicious traffic in the training set [33]. In 2009, Tavallaee et al. published a review of the various publications on anomaly based IDS research from 2000 to 2009 [4]. The reader is referred to that publication for details on their impressively thorough critique of the research conducted during that period but point out two of their criticisms that this research addresses. The first is the lack of pre-processing of the training data. Datasets in this domain will typically have features that vary greatly in scale. Features measured in the number of packets sent or received in a period may have ranges in the hundreds and thousands while timing variables such as average inter-arrival times will be in small fractions of a second. To prevent classifiers from bias towards features with larger units some transformation is typically required such as normalization or the use of principal components. The researchers show that only 21% of the published studies indicated that the researchers conducted any kind of transformation.

The other main shortcoming that this research attempts to address is the number of observations used and the balance of classes used in the training sets. The training data from 1999 KDD data, for example, has 494,000 observations of which 81% are malicious or attack traffic. While the ratio between the normal and attack traffic is questionable, the most unrealistic feature is the sheer number of observations of attack traffic. Writing in *Computer World,* Pegna describes the challenge of finding relevant observations of attack or malware traffic,

> "Despite the increase of cyberattacks reports in the news that have affected organizations across a broad set of industries, positive data from real cyberattacks and malware infections are not easily accessible. And this is particularly true for "targeted" attacks where the attack is highly customized for a particular target... It should also be clear that, no matter how many positive samples are available, the training data for the machine-learning model will be highly unbalanced, as the amount of negative samples (e.g., benign network traffic) will always be many orders-of-magnitude more abundant than the positive (e.g., cyberattack, malware infection) data samples" [5].

Tavallaae et al. [4] point out that of the 163 publications on an anomaly based IDS, only 34 bothered to specify the size and balance of the training data. Of the 34 that did, 24 featured a training set where the attack observations made up between 50-80% of the data. Of the total 163, only eight publications specified the use of a more realistic balance of 1-2% in the training data. Echoing much of the same criticism as Tavallaae et al., Sommer and Paxon sum up the frustration of many in the network security industry with the academic research community stating,

> "The intrusion detection community does not benefit any further from yet another study measuring the performance of some previously untried combination of a machine learning scheme with a particular feature set, applied to something like the DARPA dataset" [3].

Since Sommer and Paxon's article there have been only a limited number of articles on machine learning based IDS. Dube et al. [34] proposed a technique to

categorize known malware infections by type using static heuristics. Moore et al. [35] examined intrusion detection data from several years of Cyber Defense Exercises (CDX) conducted annually by the military service academies. The research focused on feature selection for artificial neural networks but primarily trained classifiers on large balanced samples. Recently, commercial firms [36] have advertised using large deep belief networks for intrusion detection with some interest in these topic mirrored in academic journals. Gao et al. [37] examined the efficacy of Restricted Boltzman Machines (RBM) [38], a type of deep belief network, in 2014 but used the 1999 KDD dataset. The next year Li et al. [39] proposed a hybrid of an RBM with a traditional back propagation neural network but also used the much maligned 1999 KDD dataset.

2.3. Methods to address class imbalance

When a classifier is trained on a dataset where one class makes up the preponderance of the data, the classifier will often bias towards the majority class leading to poor predictions of unseen data. This situation is referred to as a class imbalance problem. Methods to the class imbalance problem generally fall into two approaches. The first approach entails either modifying existing algorithms or using weighting schemes to accommodate the class imbalance. The second involves sampling strategies to alleviate the imbalance or ensure that minority classes are represented well enough in the training set for the classifier to learn the appropriate features. We will examine the second in more detail since that is the approach of this research. In 1999, Choe and Ersoy [40] proposed combining neural nets with a rule extraction algorithm based on decision trees. The authors admit that stratified sampling of the classes is ideal for neural networks but posit that in many cases there are not enough examples of the minority class

for this to be practical. Their method uses multiple bootstrapped examples of the minority classes and subsampled examples of majority classes for the training data. The authors point out their rule extraction technique helps add explanatory power to the black-box nature of traditional neural nets. The authors tested their methods on a variety of real and synthetic datasets with promising results.

Support vector machines (SVM) are one of the most common techniques used in weighting schemes due to the presence of a cost parameter that can be assigned to different classes in the data. While other algorithms can specify class weights, the majority of the literature examines SVMs. Verpoulous et al. explored methods to change the sensitivity of SVM to handle class imbalance [41]. Their technique involves assigning penalty costs to particular classes that induces a decision boundary to be more distant from the critical class than from the others. The authors show that their method is able to balance both sensitivity and specificity balancing between the off-diagonal terms of the confusion matrix. Several other researchers have modified SVMs using similar techniques. Lin and Wang [42] introduced the fuzzy SVM in 2002 where training examples are assigned different fuzzy-membership values based on their importance, and these membership values are incorporated into the SVM learning algorithm to make it less sensitive to outliers but did not directly address the class imbalance problem. Batuwita and Palade [43] introduced a modification of the FSVM in 2010 called the fuzzy support vector machine for class imbalanced learning (FSVM-CIL) to address this issue. FSVM-CIL addresses class imbalance by assigning misclassification costs and uses quadratic optimization to account for both outlier and class imbalance. The following year, Lakshmanan et al. [44] produced a very similar FSVM method tested on medical datasets with similar results.

Using modified sampling approaches has been the more common method for the class imbalance problem, principally over sampling the minority class or under sampling the majority. In 2002, Chawla et al. introduced an alternate method called the synthetic minority oversampling technique (SMOTE) [45]. SMOTE uses k-nearest neighbors (k-NN) to create new artificial observations that are similar to, but not identical to existing observations. Figure 1 shows the pseudo-code for SMOTE.

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples
1. (∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
2. **if** $N < 100$
3. **then** Randomize the T minority class samples
4. $T = (N/100) * T$
5. $N = 100$
6. **endif**
7. $N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
8. k = Number of nearest neighbors
9. $numattrs$ = Number of attributes
10. $Sample[\,][\,]$: array for original minority class samples
11. $newindex$: keeps a count of number of synthetic samples generated, initialized to 0
12. $Synthetic[\,][\,]$: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
13. **for** $i \leftarrow 1$ **to** T
14. Compute k nearest neighbors for i, and save the indices in the $nnarray$
15. Populate(N, i, $nnarray$)
16. **endfor**

Populate(N, i, $nnarray$) (∗ *Function to generate the synthetic samples.* ∗)
17. **while** $N \neq 0$
18. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
19. **for** $attr \leftarrow 1$ **to** $numattrs$
20. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
21. Compute: gap = random number between 0 and 1
22. $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23. **endfor**
24. $newindex{+}{+}$
25. $N = N - 1$
26. **endwhile**
27. **return** (∗ *End of Populate.* ∗)
End of Pseudo-Code.

Figure 1: Chawla's SMOTE Pseudo-code, from [45]

In simpler terms, SMOTE samples a minority class observation (line 10) then randomly selects one of its k-nearest neighbors in each dimension (lines 13-18) and assigns the new

observation to a random point between them (lines 19-23). This has the effect of creating new data points that are bounded within the feature space of the original data. Consider the simple two-dimensional example shown in figure 2. The top figure shows the original data with only ten minority observations represented by the blue dots and fifty majority observations in red dots. We create 40 additional minority observations with SMOTE in the bottom figure where k = 5 using the "DMwR" package in R [46]. Notice that none of the new points exceeds the boundaries of the original data.

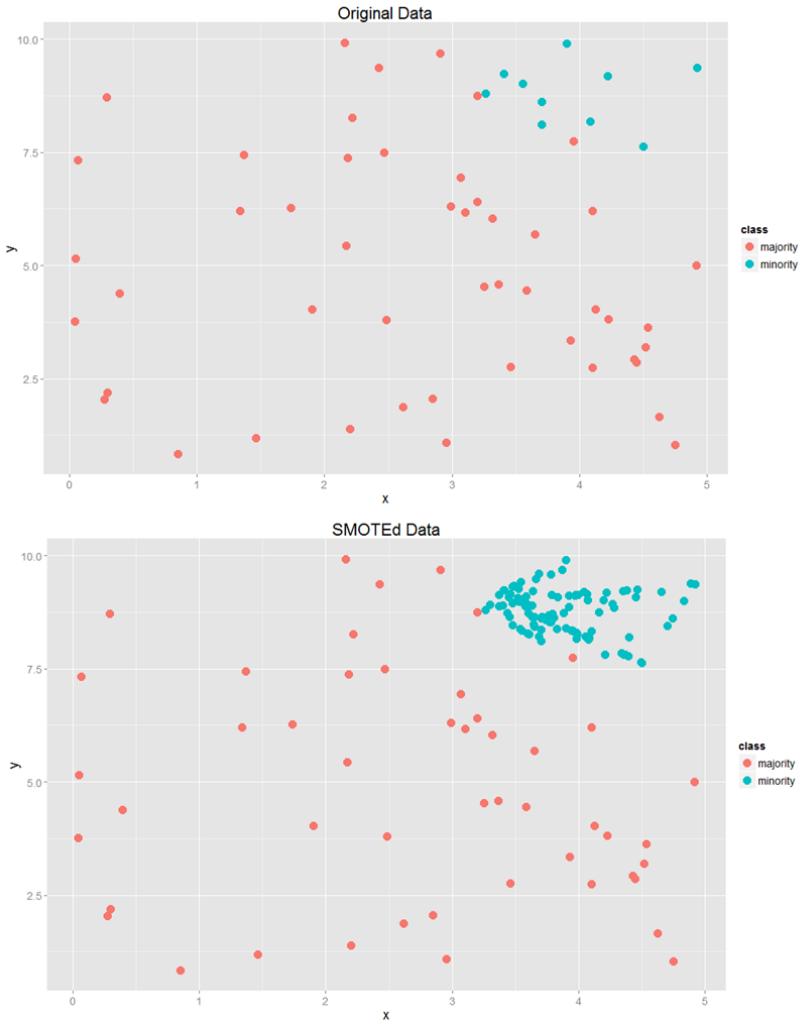

Figure 2: SMOTE data creation example

There are benefits and drawbacks to such an approach. SMOTE is cautious in that it will not create a new data point that is more extreme than a known observation. If the number of samples is small, however, this could be limiting. The small sample size may not be representative of the feature space that the class truly inhabits. The SMOTE data will then be biased towards the problematic original data sample. Researchers have proposed several modifications to improve the original method.

Chawla introduced a modification of SMOTE called SMOTEboost in 2003 that improves on the original technique using a combination of SMOTE with gradient boosting methods [47]. In 2009, Hu et al. proposed another improvement on SMOTE called modified SMOTE (MSMOTE) and modified SMOTEboost (MSMOTEboost) [48]. The authors' principal criticism of the original SMOTE was that it did not consider the distribution of minority classes and latent noises in data set when it generates synthetic samples. To address this, the authors group the minority class data into three groups based on their difficulty for the SVM to classify them and uses a k-nearest neighbor algorithm to improve the chance of correct classification. Using F-ratios as a metric, the authors showed their modifications slightly improved the performance of the original algorithms on three publicly available datasets from the UCI data repository. Seiffert et al. introduced another modification to the original SMOTE called random under sampling boost (RUSBoost) [49]. Their technique under samples the majority class randomly before incorporating the Ada boost algorithm [50]. The authors show that their method achieves similar accuracy results to SMOTEBoost but is far less time consuming and less computationally expensive.

The most recent contribution to synthetic data creation comes from Menardi and Torreli in 2010 with the Random Over Sampling Examples (ROSE) method [51]. ROSE creates kernel

density estimates for each feature in the dataset and then samples from the esitmated densities to create new observations. Unique to ROSE among the methods for synthetic data creation, synthetic examples of the majority class are also created with the same kernel density sampling technique. The authors point out several advantages to this principally the benefit of using the original dataset for testing while training exclusively on the synthetic observations. Kernel desnity estimates are a non-parametric method of estimating the probability density function of a random variable first introduced by Rosenblatt in 1956 [52]. A kernel function, typically a symetric unimodal function such as the normal distribution, is fit over each data point and then summed and divided by the number of samples to ensure the distirubtion integrates to 1. The below equation shows the basic kernel density estimate where x_i is a sampled observation, K is the kernel function, and h is a smoothing parameter.

$$\widehat{f_h}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) \quad (1)$$

The ROSE authors refer to the vast literature on kernels that shows the choice of kernel is largely immaterial [47,48] but the choice of the h parameter is crucial to the accurate estimation of the density function. We demonstrate this by performing a kernel density estimate on 100 randomly generated points from a standard normal distribution using a gaussian kernel. In figure 3, the thick gray line shows the true density while the black line uses an h paramter of 1 and the red line uses an h parameter of 0.2. The black ticks at the bottom show the actual generated values. We see that larger values of h tend to oversmooth and may mask the true structure of the data while smaller values of h will be spiky and harder to intrpret.

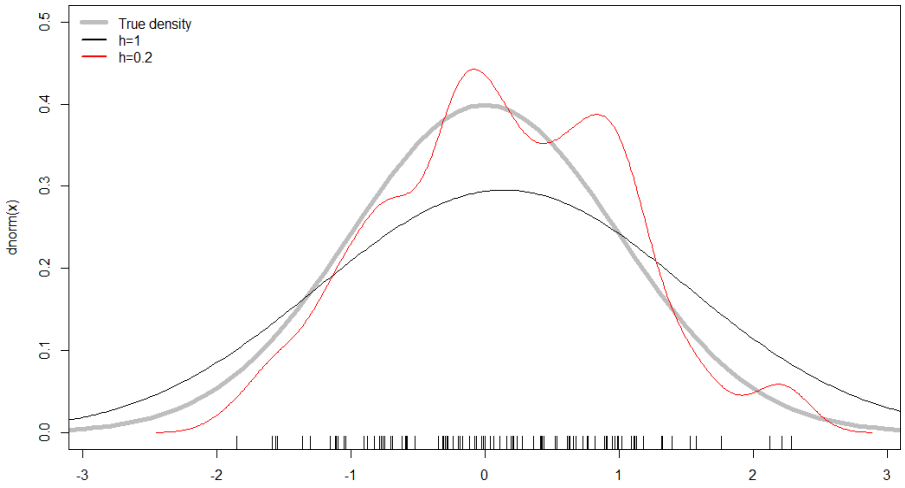

Figure 3: Effect of bandwidth size on kernel density estimates

In 2014, Menardi and Torreli collaborated with Lunardon to create a package in the R statistical programming environment with functions to create minority observations using the ROSE method [55]. The package uses the gaussian kernel and the standard deviation of each feature for the *h* paramter as a default. The code allows the user to adjust the bandwith for each class with a scaling multiplier on *h*. Using a tree based classifier on a small two dimensional dataset, the authors show that the synthetic ROSE data performs better than traditional over and under sampling methods as measured by computing the ROC area under the curve (AUC). For comparison with SMOTE, we peform a similar experiment on the notional two dimensional data from figure 2, using the authors' R code. We vary the levels of the *h* scaling parameter to see how the synthetic data behaves. Results are shown in figure 4.

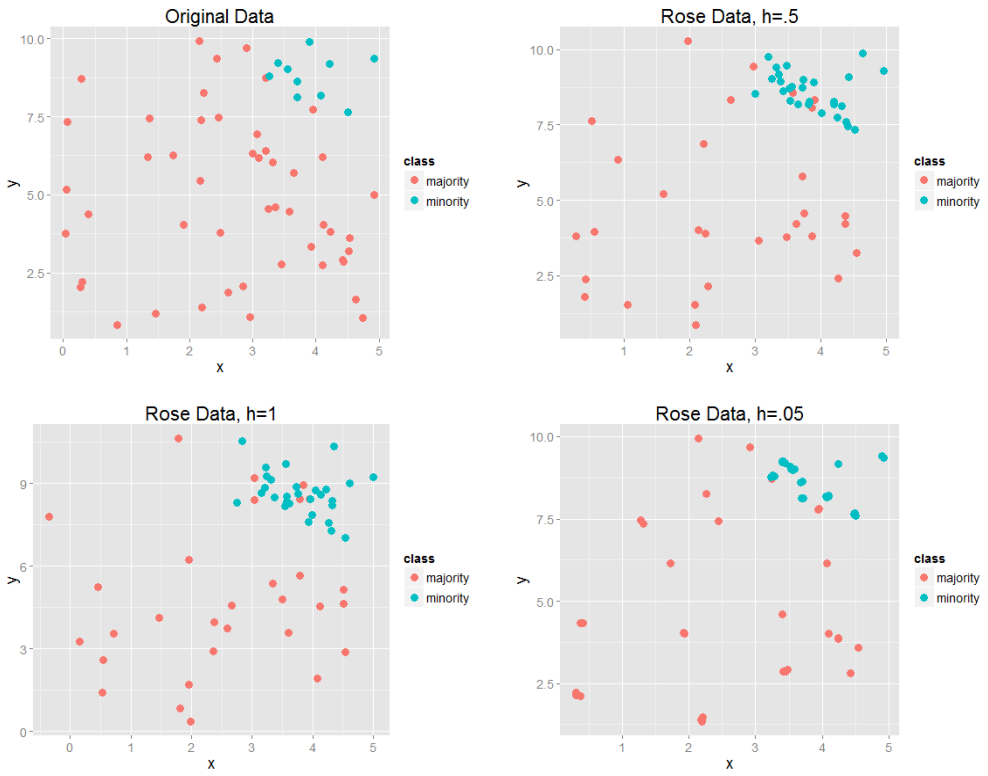

Figure 4: Bandwidth selection with ROSE

Notice that the classes in the original data in the top left are nearly completely separable. If we use a value of *h* that is too large like in the bottom left figure, we can blur the sepearation between the classes. A value of *h* that is too small, however, may simply return nearly identical points to the original data as if the points were drawn by simpling sampling the original data with replacement. In this case, the method will behave similar to SMOTE where the new data does not inhabit areas outside the feature space of the original sample.

2.4. Experiments on class imbalance methods

Weiss and Provost conducted one of the first published experiments on class imbalance in 2001 using 25 datasets from the UCI Machine Learning Repository [56] with a wide range of class balance [57]. The researchers only used a single classifier, the decision tree based C4.5

algorithm developed by Quinlan [58]. The authors verified the problematic nature of unbalanced training data and concluded that the optimal balance in the training set is between 50-80% for the class of interest. In 2003, Drummond and Holte [59] followed up with a similar study using four datasets and the C4.5 algorithm but looked at the performance of over and under sampling. The authors concluded that under sampling is superior to traditional over sampling of the minority class for C4.5. Batista et al. [60]conducted a similar experiment in 2004 but incorporated different over sampling techniques including SMOTE, tomek links, and novel nearest neighbor techniques. Using 14 datasets, they concluded that over sampling techniques in general provided improvements in performance as measured by AUC for C4.5. They also noted that traditional over sampling by sampling with replacement was competitive with more sophisticated methods such as SMOTE.

In 2007, Van Hulse et al. [61] conducted the most comprehensive experiment currently found in literature. The authors used 35 datasets from the UCI repository whose level of class imbalance ranged from 1-35% minority class observations. Using 11 classification methods available in the WEKA software suite [62], the authors trained classifiers using seven over sampling techniques, including SMOTE and RUS. Using AUC and Tukey's Honestly Significant Differences (HSD) test, the authors concluded that there is no uniform superior choice of sampling technique with performance varying between data type and classification method. Van den Poel and Burez [63] conducted the first domain specific experiment using six proprietary datasets of customer turnover known as churn. They examined the data using gradient boosting and random forests. For over sampling, the authors used the cube method developed by Deville [64] and concluded that sampling methods showed no improvement over using the data at the natural levels of class imbalance. Jeatrakul [65] used a hybrid-SMOTE method incorporating

artificial neural networks for oversampling and found favorable results on four moderately unbalanced datasets.

This research has some similarities with the studies previously mentioned but has some key differences. The most salient difference is the level of class imbalance examined. None of the studies examined an imbalance at or below 1%. See table 2 below. The other studies, with the singular exception of Burez and Van den Poel, examined data from a variety of disciplines and were only able to make broad generalizable recommendations on dealing with class imbalance. This research is domain specific and aims to determine the most appropriate methods for handling class imbalance within the context of anomaly based intrusion detection. Lastly, to the author's knowledge, no other studies currently in literature have examined the ROSE sampling technique other than the ROSE creators.

Table 2: Summary of Class Imbalance Experiments

Author	Year	Classifier(s)	Smallest portion of minority class examined	Over sampling methods employed
Weiss and Provost [57]	2001	C4.5 Only	3.90%	none
Drummond and Holte [59]	2003	C4.5 Only	2.55%	traditional
Batista et al. [60]	2004	C4.5 Only	2.55%	multiple
Van Hulse et al. [61]	2007	multiple (11)	1.33%	multiple
Burez and Van den Poel [63]	2009	Gradient Boosting, Random Forests	2.98%	cube
Jeatrakul et al. [65]	2010	Neural Networks, k-NN, SVM	20.60%	Hybrid-SMOTE

III. Methodology

3.1 Chapter Overview

Chapter 3 begins with a description of the two datasets used in the research in section 3.2. The section provides a brief description of the methodology used to create the datasets, the types of attack or malicious traffic, and summary statistics. Section 3.3 presents an explanation of the four classification methods used, artificial neural networks, random forests, gradient boosting, and support vector machines (SVM). Section 3.4 explains the basic methodology of the experiment detailing how the datasets were sampled, divided between training and test sets, and the method to choose model parameters.

3.2 Dataset Description

The first dataset under analysis comes from the Australian Centre for Cyber Security (ACCS), a research center affiliated with the University of New South Wales (UNSW) and the Australian Defence Force Academy (ADFA). The dataset is titled the UNSW-NB15 dataset [66] but for brevity's sake will be referred to as the UNSW dataset. ACCS researchers sought to create a dataset that could fulfill the role of a standard benchmark dataset for IDS researchers like the 1999 KDD dataset but with traffic that reflected more modern attack behaviors, especially low profile attack traffic. Researchers generated the internet traffic through a combination of real and simulated network traffic at the ACCS' cyber range in January and February of 2015.

Similar to the 1999 KDD team, the ACCS researchers simulated the attack traffic which was then saved in packet capture (PCAP) files which could be processed to create features for each observation in a tabular format. The ACCS team made several notable improvements. A network traffic generator from IXIA [67] created both the normal and attack traffic but sampled

from the common vulnerability and exposure (CVE) site to ensure the simulated traffic reflects modern malicious network traffic. The CVE [68] is a repository of reported cyber-attacks and malicious traffic maintained by the MITRE Corporation on behalf of the National Cyber Security Division of the US Department of Homeland Security (DHS). Using data from the CVE site, the researchers created attack traffic from nine distinct categories. The CVE resource also helped the researchers ensure a realistic distribution of attack types. Table 3 outlines the attack types with a short description of each.

Table 3: Attack Traffic Summary for UNSW dataset [66]

Type	No. of Records	Description
Normal	2,218,761	Normal network traffic.
Fuzzers	24,246	Attempting to crash a program or network by feeding it massive amounts randomly generated data, or fuzz.
Analysis	2,677	Different attacks of port scan, spam and html file penetrations.
Backdoors	2,329	A technique in which a system security mechanism is bypassed stealthily to access a computer or data.
Denial of Service (DoS)	16,353	An attempt to make a server or network resource unavailable to users by interrupting or suspending the services of a host connected to the internet.
Exploits	44,525	An attacker exploiting a known security problem within an operating system or software.
Generic	215,481	A technique that works against all block ciphers without consideration of the structure of the block-cipher.
Reconnaissance	13,987	Strikes that can simulate attacks that gather information.
Shellcode	1,511	A small piece of code used as the payload in the exploitation of software vulnerability.
Worms	174	An attack that replicates itself in order to spread to other computers relying on security failures of the target computer.

After capturing the network traffic, the researchers processed the data to extract 49 total features for release to the public as a labeled dataset in comma separated value (CSV) files. The 49 features are divided into six categories: flow features, basic features, content features,

time features, connection features, and an “additional generated features” category. Tables 4 and 5 outline the features with a brief description of each.

Table 4: UNSW Dataset Features

	#	Name	Type	Description
Flow based features	1	srcip	nominal	Source IP address
	2	sport	integer	Source port number
	3	dstip	nominal	Destination IP address
	4	dsport	integer	Destination port number
	5	proto	nominal	Transaction protocol
Basic Features	6	state	nominal	State and its dependent protocol, e.g. ACC, CLO
	7	dur	float	Total record duration
	8	sbytes	integer	Source to destination bytes
	9	dbytes	integer	Destination to source bytes
	10	sttl	integer	Source to destination time to live
	11	dttl	integer	Destination to source time to live
	12	sloss	integer	Source packets retransmitted or dropped
	13	dloss	integer	Destination packets retransmitted or dropped
	14	service	nominal	Service type, e.g. http, ftp, dns, etc.
	15	sload	float	Source bits per second
	16	dload	float	Destination bits per second
	17	spkts	integer	Source to destination packet count
	18	dpkts	integer	Destination to source packet count
Content Features	19	swin	integer	Source TCP window advertisement
	20	dwin	integer	Destination TCP window advertisement
	21	stcpb	integer	Source TCP sequence number
	22	dtcpb	integer	Destination TCP sequence number
	23	smeansz	integer	Mean of the source's flow packet size
	24	dmeansz	integer	Mean of the destination's flow packet size
	25	trans_depth	integer	Represents the pipelined depth into the connection of http request/response transaction
	26	res_bdy_len	integer	Actual uncompressed content size of the data transferred from the server’s http service.

Table 5: UNSW Dataset Features Continued

	#	Name	Type	Description
Time Features	27	sjit	float	Source jitter (mSec)
	28	djit	float	Destination jitter (mSec)
	29	stime	float	Record start time
	30	ltime	float	Record last time
	31	sintpkt	float	Source interpacket arrival time (mSec)
	32	dintpkt	float	Destination interpacket arrival time (mSec)
	33	tcprtt	float	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
	34	synack	float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
	35	ackdat	float	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
Additional Generated Features	36	is_sm_ips_sports	binary	If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0
	37	ct_state_ttl	integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
	38	ct_flw_mthd	integer	No. of flows that has methods such as Get and Post in http service.
	39	is_ftp_login	binary	If the ftp session is accessed by user and password then 1 else 0.
	40	ctftp_cmd	integer	No of flows that has a command in ftp session.
Connection Features	41	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).
	42	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).
	43	ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
	44	ct_src_ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
	45	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).
	46	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).
	47	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).
Labels	48	attack_category	nominal	The name of each attack category.
	49	Label	nominal	0 for normal and 1 for attack records

Of the 49 features in the dataset, eight were removed prior to training leaving 39 numeric features and two label features. The features found in one through six are problematic for two reasons. While port numbers are integers, in reality they are categorical variables that appear numeric since they are indexed with numbers. If left in this form, an observation with a port number of 100 would be incorrectly viewed as 20% greater than an observation from port 80. While tree based classifiers can handle the other non-numeric features, neural networks and support vector machines must have numeric data. Nominal features like IP addresses and transaction protocols can be converted to numeric by adding binary features. This would be practical if there were only a handful of unique observations. For most of the nominal features there are, however, dozens of unique values, which would require drastically increasing the number of features in the overall dataset. Additionally, many of these features like IP addresses are easily disguised by even the most rudimentary hackers [69] and are unlikely to provide meaningful information to an IDS. For these reasons and the desire to measure each classifier on identical datasets, we only retain the numeric features. The size of the training and test sets for the UNSW data are covered later in section 3.4.

The second dataset under examination was created in 2014 at the Air Force institute of Technology (AFIT) by Moore [70] using files from the US Military Service Academies' annual Cyber Defense Exercise (CDX) competitions [71] from 2003 to 2009. We refer to this dataset as simply the CDX dataset. Using packet capture files from six years of the CDX exercises, Moore conducted extensive processing of the raw data files to create a labeled dataset with 204,371 observations and 212 features. Table 6 summarizes the dataset showing the number of attack observations by year.

Table 6: CDX Dataset Overview [70]

	2003	2004	2005	2006	2007	2009	Total
Observations	30829	73507	1301	9535	51777	37422	204371
Threat observations	3357	8628	39	214	41	4037	16316
Percentage	10.89%	11.74%	3.00%	2.24%	0.08%	10.79%	7.98%

Moore's research [70] focused on feature selection and concluded that 21 of the original 212 features were relevant to intrusion detection. This research only uses the 21 features recommended by Moore. The majority of the features are similar to the basic features category from the USNW dataset but with some subtle differences. The features in the reduced CDX dataset include summary statistics such as the mean, variance, and quartiles of packet flows and timing variables while the USNW dataset contains only the mean or totals of these features. Table 7 shows a summary of the CDX dataset features. While not as current as the UNSW dataset, the CDX dataset provides a meaningful contrast. The CDX dataset having fewer and distinct features from the USNW set may illuminate differences in the performance of either sampling methods or classification models.

Table 7: Summary of CDX Dataset Features [70]

Original Feature Number	Feature description
26	Median of control bytes in packet
80	Maximum segment size requested as a TCP option in the SYN packet opening the connection (server to client)
81	Maximum segment size observed during the life of the connection (client to server)
83	Minimum segment size observed during the life of the connection (client to server)
84	Minimum segment size observed during the lifetime of the connection (server to client)
86	Average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data packets reported (server to client)
90	Minimum window advertisement seen (if both sides negotiated window scaling)(server to client)
171	Third quartile of control bytes in packet (client to server)
173	Variance of control bytes in packet(client to server)
193	Maximum of control bytes in packet (server to client)
96	Total number of bytes sent in the initial window (i.e., the number of bytes seen in the initial flight of data before receiving the first ACK packet from the other endpoint acknowledging some data – not the 3-way handshake) (server to client)
97	Total number of segments (packets) sent in the initial window (client to server)
98	Total number of segments (packets) sent in the initial window (server to client)
10	Minimum of bytes in (Ethernet) packet, using the size of the packet on the wire
158	Maximum of bytes in (Ethernet) packet(client to server)
178	Third quartile of bytes in (Ethernet) packet(server to client)
179	Maximum of bytes in (Ethernet) packet (server to client)
180	Variance of bytes in Ethernet packet (server to client)
17	Minimum of total bytes in IP Packet, using the size of the payload declared by the IP Packet
186	Maximum of total bytes in IP packet (server to client)
187	Variance of total bytes in IP packet (server to client)

3.3 Classification Techniques

3.3.1 Artificial Neural Networks

The first classification technique examined is artificial neural networks that we will refer to by the common shorthand term, neural networks. Neural networks are a type of computational model inspired by the biological nervous system. In the nervous system, neurons receive signals through synapses on the dendrites of the neuron. If a signal is sent that exceeds a threshold, the neuron activates and sends a signal through the axon. Figure 5 shows a simple sketch of the process.

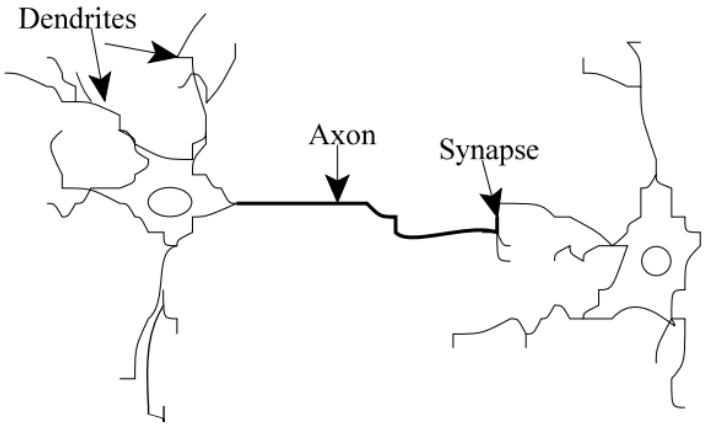

Figure 5: Neural Diagram from [72]

Neural network models were so named due to a similar relationship between inputs and activation. Data points are inputs that act as the synapse. These inputs are multiplied by weights, which represent the signal strength along the axon in the biological system. A mathematical function determines which neurons activate. A separate function determines the output of the synapse, which in classification indicates the predicted class or outcome given the input data and the weights. Before we explore the mathematics behind this process, we first show a brief example to illustrate the relationship between the inputs, weights, and outputs. We'll use the

Fisher iris dataset [73] and the "nnet" package developed by Venables and Ripley [74] for the R statistical computing environment. The authors first developed the package in 2002 but for this example and in the remainder of the thesis, we show output from the most recent version, 7-3.11. The iris dataset contains 150 observations of four flower measurements from three different species of iris. The features are the petal width, petal length, sepal width, and sepal length. The three species are setosa, versicolor, and virginica. The flower measurements are the inputs, which feed a single hidden layer of two neurons, which then activate to predict the output or species. Figure 6 demonstrates the process where the thickness of the lines represents larger weights. The B terms are the bias terms that feed each non-input layer, typically set to one, needed to establish the activation threshold. We see that petal length and width have small weights feeding the first neuron predicting setosa but large weights towards the second neuron that activate to predict virginica.

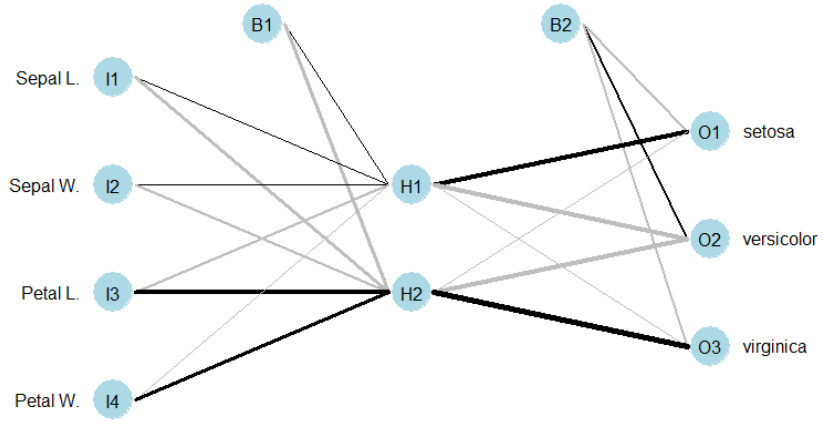

Figure 6: Neural network Example

Scatter plots of the flower measurements help illustrate the wisdom of the example neural network. Notice in the first chart in Figure 7 the large values of petal length and width associated with the virginica and the small values associated with setosa. This neural network found a

simple relationship between two features but neural networks are able to discover more complicated non-linear relationships. Large datasets with many features often have complex nonlinear relationships among inputs and outputs that are not evident in simple scatterplots.

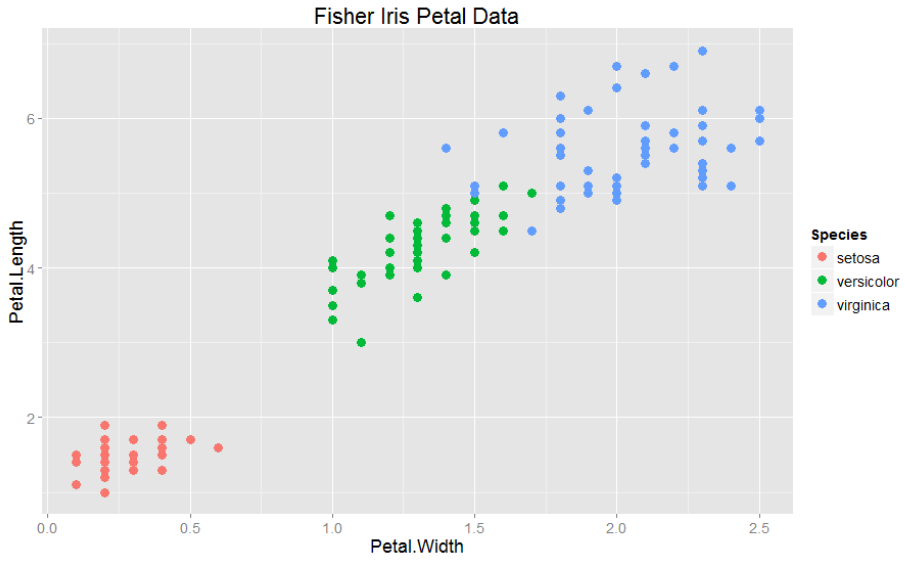

Figure 7: Fisher Iris Petal Data

We now examine the mathematics behind the activation functions and how the weights are chosen using the backpropagation algorithm first developed by Rumelhart and McClellan in 1986 [75]. Since then neural networks have become one of the most popular classification algorithms applied in many academic and commercial settings [76]. The most common form of a neural network is the feed forward, back propagation neural network. In back propagation, we send the signals forward, and then calculate the errors by moving backward through the network in order to adjust the weights to improve the prediction, typically done iteratively until a tolerance is reached or the specified number of iterations occurs. The first function is the activation algorithm which is simply the sum of the inputs (x_i) multiplied by the weights (w_{ji}) shown in (2)

$$A_j(\bar{x}, \bar{w}) = \sum_{i=1}^{n} x_i w_{ji}. \quad (2)$$

The output function can be identical to the activation function called an identity function. Typically, in neural networks we see the sigmoidal function shown in (3) because it introduces non-linearity to the model, is continuous, and relatively easy to differentiate although others are used. The sigmoidal function is near zero for large positive numbers, 0.5 at zero, and near zero for large negative numbers allowing a smooth transition between the low and high output of the neuron

$$O_j(\bar{x}, \bar{w}) = \frac{1}{1+e^{A_j(\bar{x},\bar{w})}}. \quad (3)$$

Because the goal of the prediction is to minimize error, we define an error function as the sum of the squared differences between the output and the true output shown in

$$E_j(\bar{x}, \bar{w}, d) = \sum_j \left(O_j(\bar{x}, \bar{w}) - {d_j}^2\right). \quad (4)$$

The backpropagation algorithm [75] calculates the relationship between the error and the inputs, outputs, and weights using the method of gradient descent (4). This formula can be interpreted in the following way: the adjustment of each weight w_{ji} will be the negative of a constant, *eta*, multiplied by the dependence of the previous weight on the error of the network, which is the derivative of E with respect to w_{ji}. The size of the adjustment will depend on eta, the learning rate, and on the contribution of the weight to the error of the function. If the weight contributes a lot to the error, the adjustment will be greater than if it contributes in a smaller amount. (5) is used until we find better weights to minimize the error.

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} \quad (5)$$

From this, we see that the adjustment to the weights requires that we simply find the derivative of E with respect to $w_{ji.}$ Left unchecked, this could allow a neural network to minimize the error to near zero for the training data after many iterations, which could cause a problem of overfitting where the trained net does not generalize well to new data. Practitioners prevent this in several ways. The first is to limit the number of iterations for training a net. The second is to introduce a decay parameter, λ [77]. In practice, this penalizes large weights and effectively limits the freedom in your model. The regularization parameter λ determines how you trade off the original cost E with the large weights penalization shown in

$$\Delta w_{ji} = w_{ji} - \eta \frac{\delta E}{\delta w_{ji}} - \eta \lambda w_{ji}. \qquad (6)$$

The $-\eta\lambda w_{ji}$ term causes the weight to decay in proportion to its size. The "nnet" package comes with a default decay parameter of zero but can be adjusted if there is a stronger possibility of overfitting. This often occurs when there are many features in the dataset.

To this point we have not mentioned the η term from (5) known as the learning rate. The backpropagation algorithm is guaranteed to find a minimum in the error function from (4) but not necessarily a global minimum. The term η tells the neural network how far to go while the rest of (4) is the gradient showing which direction will improve the function. Because we seek to minimize the error, the η term is negative. A traditional back propagation algorithm requires that we set η, typically to a very small number to avoid bounding over a minimum in the function. This can make training a neural network incredibly slow [78]. Most software packages, including nnet in R, use some time of non-linear optimization to calculate η dynamically at each iteration. If the gradient is very shallow, we want to move farther since we are likely far from the minimum. As the gradient gets steeper, we are likely approaching a potential minimum and want

to use a more conservative step. To do this, the techniques rely on the second order derivatives found in the Hessian matrix. Computing the Hessian at each step is memory and computationally expensive [79]. Quasi-Newton methods use an approximation of the Hessian that make them practical for all but very large problems. The nnet package used in the research employs one of the more popular quasi-Newton methods, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [80].

3.3.2 Random Forests

Breiman and Cutler created the Random Forest algorithm for regression and classification in 2001 [81]. Random Forests use modified bootstrap aggregating (bagging) of simpler decision tree models. Bagged models sample the data with replacement multiple times and build classifiers for each sample. This ensemble of classifiers then vote, in a sense, to determine the overall prediction. Random forests introduced a slight modification to bagged trees that improved accuracy. Random forests quickly became popular due to their relative simplicity and excellent performance compared to other classification models. Before we examine Random forests, we first look at the mechanics of the basic decision tree classifier at the heart of random forests, the Classification and Regression Trees (CART) first introduced by Breiman in 1983 [82].

The classification tree begins at the root node with the entire dataset. From the root, the tree splits into two or more child nodes sometimes called leaves. In a binary example, the algorithm searches for a feature that makes the largest distinction between the two classes. The algorithm takes the resulting new nodes and repeats the process, continuing the recursion until some stopping criterion is reached. Using the Fisher iris data as an example, we see the splits in

Figure 8 show that the first split based on a petal length of 2.4, separated the setosa from virginica and versicolor perfectly. A second split on petal width separates virginica from versicolor but includes five virginicas misclassified as versicolor.

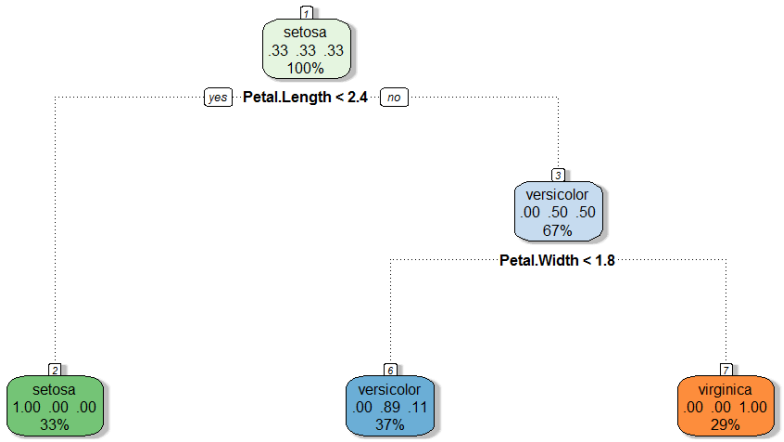

Figure 8: Basic Classification Tree Example

Figure 9 shows the graphical example of the splits with five blue dots on the incorrect side of the petal width partition. It is possible the tree could continue to make further splits to create more partitions to separate those five points from the versicolor observations but such extensive partitioning generally leads to overfitting, a common problem with tree-based classification.

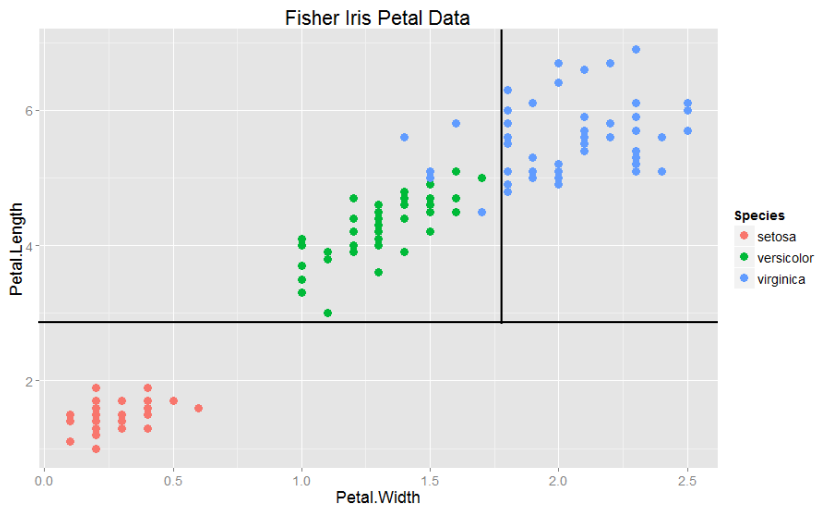

Figure 9: Fisher Iris Classification Partitions

A common way to prevent this overfitting is a technique called pruning where we remove sections of the tree that provide little or no information. A variety of methods to determine information gained and where to prune exist in literature [83]. Random forests prevent overfitting with a different technique eschewing pruning altogether. The following, adapted from [81], shows the basic procedure for creating a Random Forest:

1. Let N be the number of observations in the training set.
2. Take a random sample of size N with replacement from the data (bootstrap sample).
3. Take a random sample without replacement of the features of size, m_{try}.
4. Construct a split by using the m_{try} features selected in Step 2.
5. Repeat steps 2 and 3 for each subsequent split until the tree is as large as desired.
6. Repeat steps 1-5 a large number of times, creating typically 500-1000 trees.
7. Apply the validation or test set data to the forest of trees.
8. Assign each observation to a final category by a majority vote over the set of trees.

While traditional trees make splits by selecting the feature with the largest class distinction in the entire dataset, Radom Forests choose the best feature from a random sample of

features at each node. The method does not prune but typically, splits the data until the classes are perfectly separated which would tend to over fit as a single tree model. This may seem counterintuitive. How can a method that uses less data, over fit trees, and randomly ignores potentially significant features be an improvement? The answer lies in the aggregating. We can consider each tree as a weak learner in that it provides only a small amount of information. When we combine hundreds or thousands of trees each with a small amount of information, we improve over a single tree but with less variance due to the random feature selection. Had we simply bagged a series of traditional CART trees, the trees would look very similar. A few dominant features would partition the data in much the same way. By using a small sample of features, we de-correlate the trees making the fitted values across the trees more independent, allowing local features to influence prediction. Traditional bagged trees are identically distributed (i.d) while the small tweak to random forests makes them independent and identically distributed (i.i.d.). Consequently, the gains from averaging over a large number of trees reduce model variance.

The randomness introduced in the model has other benefits as well. The bootstrapped samples allows each tree to test on the unsampled points called the out-of-bag (OOB) samples. This allows for the use of the entire dataset for training if the few observations are available. Another positive feature of random forests are that they minimize the influence of potentially noisy features by ignoring them randomly based on the variables chosen unlike other classifiers such as Adaboost [50], pointed out by Breiman [81]. Lastly, the model is simple in that it only requires the choice of two parameters, the number of trees and the number of features chosen per split. Literature suggest that there is no optimal number of trees and that growing trees into the thousands provides no practical gain in accuracy [84]. The authors of the R package used in

this thesis, "RandomForest" [85] version 4.6-12, show that 500 trees is sufficient for even large datasets and that the square root of the number of features is a reliable choice for m_{try}.

3.3.3 Support Vector Machines

Support vector machines (SVM) can be traced to their original formulation by Vapnik [86] in 1963 but it was not until 1998 when Vapnik et al. [87] were able to create nonlinear classifiers using kernel methods that the models became very popular. Machine learning practitioners were so effusive with praise for SVMs in the years that followed that Bennet and Campbell penned an article reviewing recent SVM literature titled, "Support Vector Machines: Hype or Hallelujah?," [88] in 2001. Before we look at the inner workings of the SVM and the variety of kernels available, we must first examine the most basic SVM, the maximum margin classifier.

According to Vapnik, the maximum margin classifier is the optimal separating hyperplane between two classes. We use the Fisher Iris petal data as a simple example. Notice in figure 10 we have renamed virginica and versicolor as simply "other" so that we have only two perfectly separable classes. The three black lines are all examples of separating hyperplanes, which are simply lines in $\mathbb{R}^2$. There are an infinite number of possible hyperplanes but we should seek to find an optimal separating hyperplane that is farthest from the boundaries of each class in the training data in hopes that the classifier will perform well on unseen data.

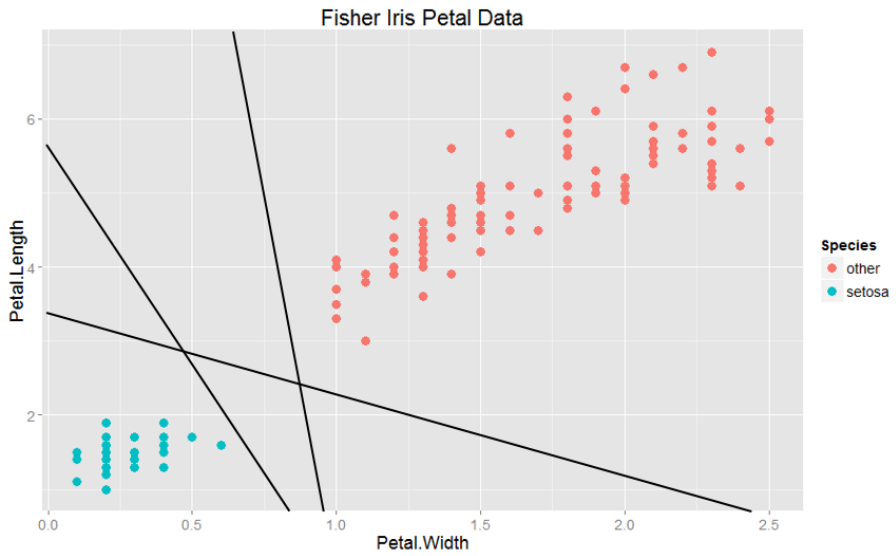

Figure 10: Fisher Iris Petals with Separating Hyperplanes

We can compute the perpendicular distance from each training observation to a given separating hyperplane. The smallest of these distances is known as the margin or M in the notation below. The maximal margin hyperplane is the hyperplane that has the farthest minimum distance to the training observations. To find this hyperplane, we can formulate the model as an optimization problem. Below we borrow the notation for the optimization problem from Hastie and Tibshirani's, *Elements of Statistical Learning* [89]. If we let $\beta_0, \beta_1 \ldots \beta_p$ be the coefficients of the hyperplane and class labels are simply (-1, 1), then we can classify based on the sign of (7).

$$f(x) = \beta_0 + \beta_1 x_{i1} \ldots \beta_p x_{ip} \tag{7}$$

The goal is then to find a hyperplane that maximizes the margin, M, by solving the optimization problem below in (8) through (10) with $\beta_0, \beta_1 \ldots \beta_p$ as the decision variables.

$$maximize\ M \tag{8}$$

Subject to:

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{9}$$

$$y_i\left(\beta_0 x_{i1} + \beta_1 x_{i2} + \cdots \beta_p x_{ip}\right) \geq M \ \forall\ i = 1 \ldots . . n \tag{10}$$

Constraint (9) ensures that each point will be on the correct side of the hyperplane provided that $M > 0$. The third constraint (9) is not really a constraint on the hyperplane but when combined with (10), $y_i\left(\beta_0 + \beta_1 x_{i1} + \cdots \beta_p x_{ip}\right)$ represents the perpendicular distance from the i[th] observation to the hyperplane. Thus, the two combined ensure that each observation is on the correct side of the hyperplane and at least M distance from the hyperplane. Hence, M represents the margin of the hyperplane and the optimization chooses the appropriate β terms to maximize M. The result gives us the maximum margin classifier. We show a graphical representation on the Iris petal chart in Figure 11. The thick black line is the maximal margin classifier while the dotted lines represent the margins.

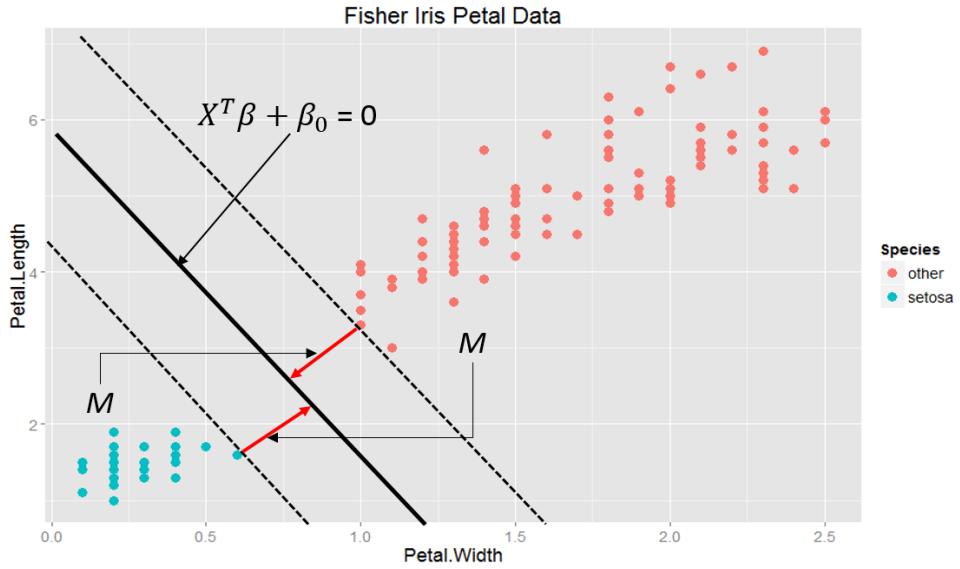

Figure 11: Maximal Margin Classifier Example

Note that only the two points on the margin, whose distance defines *M*, define the hyperplane. We call these points the support vectors. We could remove or slightly adjust any of the other observations and not change the results of the optimization. This has several convenient mathematical properties. For a linear classifier, like the one above, it can be shown that the hyperplane is defined by the dot product of the support vector observations [89]. If we had two observations, x_i and x_i', their dot product could be replaced by a generalization with a kernel function, $K(x_i, x_i')$. When a support vector classifier is combined with a nonlinear kernel, the resulting classifier is known as a support vector machine.

One question exists, "Why would we want to use a kernel?" In most cases, the true class boundary will be nonlinear. Kernels allow us to fit flexible decision boundaries by working in higher dimensional space rather than the original feature space. Some examples of common kernels for SVMs include:

- d-degree polynomial: $K(x_i, x_i') = (1 + \langle x, x' \rangle^d)$. (11)
- Radial: $K(x_i, x_i') = exp(-\gamma \|x - x'\|^2)$. (12)

Another advantage to kernels is computational. When using kernels, we only need to compute $K(x_i, x_i')$ for all distinct pairs of i and i'. This can be done without explicitly working in the enlarged feature space. This is important because in many applications of SVMs, the enlarged feature space is so large that computations are intractable. The radial kernel, for example, has an implicit, infinite-dimensional feature space [90].

In this thesis, we will use a radial kernel from the "kernlab" package, version 0.9-22, in R. The package was developed by Karatzoglou et al. [91] and relies on Platt's [92] sequential minimal optimization (SMO) technique as the optimization engine. The literature on SVMs shows that radial kernels are among the most popular and generally robust to a variety of data.

Radial kernels also have fewer tuning parameters. The polynomial kernel, for example, includes the choice of the polynomial degree, *d*. To determine the optimal choice for *d*, we must employ some kind of resampling technique and cross validation using the training data.

The γ term in (12) is chosen heuristically by the package. For research on techniques to tune this parameter, we refer the reader to Han et al. [93]. This means that we only need to adjust for the cost parameter, *C*. Because most data do not have completely separable classes like the iris dataset, we need the *C* paramter to allow for observations to exist inside the margin or on the wrong side of the classifying hyperplane. This involves slightly altering the notation from (8-10) to include slack variables, $\epsilon_1 \ldots \epsilon_n$, the *C* parameter, and another constraint to accommodate them. We show the new version in (13-16) below:

$$maximize\ M \tag{13}$$

Subject to:

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{14}$$

$$y_i\left(\beta_0 x_{i1} + \beta_1 x_{i2} + \cdots \beta_p x_{ip}\right) \geq M(1 - \epsilon_i)\ \forall\ i = 1 \ldots\ldots n \tag{15}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \tag{16}$$

A large value of *C* allows many observations to exist within the boundary and thus many observations to act as support vectors leading to a wiggly potentially over fit boundary in the original feature space. A small value of *C* will encourage a small number of observations within the margin and fewer support vectors leading to a smoother, perhaps more generalizable boundary. The method to choose the value of *C* is described in the next section.

3.3.4 Gradient Boosting Methods

Gradient boosting methods, like bootstrap aggregating, have garnered much attention and praise in recent years. The winners of recent machine learning competitions such as Kaggle or Yahoo's Learning to Rank Challenge nearly always use some type of boosting algorithm either alone or as a part of an ensemble of classifiers [94]. The term boosting was first introduced by Kearns [95] who loosely defined the term as a method to turn a weak learner that is only slightly better than random guessing into a strong learner, a classifier with near 100% accuracy. Schapire and Freund each proposed the conceptual formulation for boosting in response to Kearns' question in the early 1990s but it was not until 1996 that boosting became practical when the two collaborated to create the first useable algorithm, AdaBoost [96]. Leo Breiman, co-creator of the random forest, at the time called Adaboost, "the best off-the-shelf classifier in the world" [97]. While boosting can be applied to a variety of model types, they are most often associated with tree-based models. For this research, we use a tree based classifier found in the xgboost package in R version 0.4-2, developed by Chen et al. [98].

We discuss the mathematics and intuition behind boosting using trees below. Boosting and bagging are frequently mentioned together due to their recent popularity and their similarity. Both are ensembles of many classification trees. While both use trees, there is one significant difference. In bagging and its variants like random forests, each tree is grown independent of the others. Conversely, the boosting approach grows each new tree sequentially based on the performance of the previous tree. We weight the importance of correctly classifying each observation with an error function defined by the previous tree. The basic idea is to create short trees or stumps and to re-weight each new tree based on the misclassified observations in the previous tree. This has the effect of forcing the next tree to learn a different aspect of the

data. The early errors identify the difficult observations and the remaining trees are built to aid in learning those observations. We then combine each weak learning tree into a single stronger ensemble classifier similar to what is done in bagged trees and random forests. We show the pseudocode for a basic boosting procedure below borrowing much of the notation from Freund and Schapire's, "*A short introduction to boosting"* [99]:

1. Initialize the starting weights $w_n^{(1)}$ with a naïve 1/N for each training observation.

   For k=1 to *M*, where *M* is the number of iterations.

2. Fit a weak classification tree, f_k, in the form of a shallow tree or stump using *d* number of splits in the data to minimize the error function defined by :

   a. $\sum_{n=1}^{N} w_n^{(k)} * I(y_n \neq t_n)$ where *I* is an indicator function that returns 1 if the observation is incorrectly classified and 0 if correctly classified.

3. Calculate the error of the k[th] stump, $error_k$, by:

   a. $\text{error}_k = \frac{\sum_{n=1}^{N} w_n^{(k)} * I(Y_n \neq t_n)}{\sum_{n=1}^{N} w_n^{(k)}}$ where t_n is the true class membership of observation n.

4. Update the weights for f_{k+1} with:

   a. $\sum_{n=1}^{N} w_n^{(k)} * \exp(\alpha_k * I(y_n \neq t_n)$ where α_k is defined by:

   b. $\alpha_k = \frac{1}{2} ln\left[\frac{1-error_k}{error_k}\right]$

   END

5. Calculate final predictions with:

   a. $sign\left[\sum_{n=1}^{N} w_n^{(k)} * I(Y_n \neq t_n)\right]$

Lines 2.a and 3.a identify the incorrectly labeled observations, which get increased weights on the following tree according to lines 4.a and 4.b. The indicator function returns zeros for the correct observations which keeps their weights constant. At each new tree, however, the incorrect observations are increased meaning the weight on the correct observations grow relatively smaller as the process continues. This implies that we can choose a number of iterations that may pathologically overweight difficult or mislabeled points at the expense of the majority of the data. Servedio and Long [100] showed in a 2009 paper that most boosting procedures fail when confronted by even a small amount of classification noise as the model will be drawn toward the noisy features in a futile attempt to learn them. This implies that unlike random forests, we must carefully choose the number of iterations, *M*, to avoid overfitting. Another way to combat overfitting is the addition of a shrinkage parameter, λ. In this case, we don't update with α_k but $1 - \lambda * \alpha_k$. Shrinkage parameters are typically small, 0.01 or 0.001. The other tuning parameter is the number of splits, d. The d parameter is typically 1-3 representing a stump rather than a tree. If we choose small levels of d we must choose a large enough *M* to ensure a good fit. We describe the tuning methods and results in section 3.4.4.

3.4 Overall Methodology

3.4.1 Data Partitioning

We begin the analysis with a basic partition of the two datasets into training sets with a second portion of the datasets set aside for testing. We see in figure 12 that a 75/25 split leaves training sets that are both unrealistically large and unrealistically unbalanced. We show later in section 4 that our classifiers can predict with near perfect accuracy when trained on such large datasets.

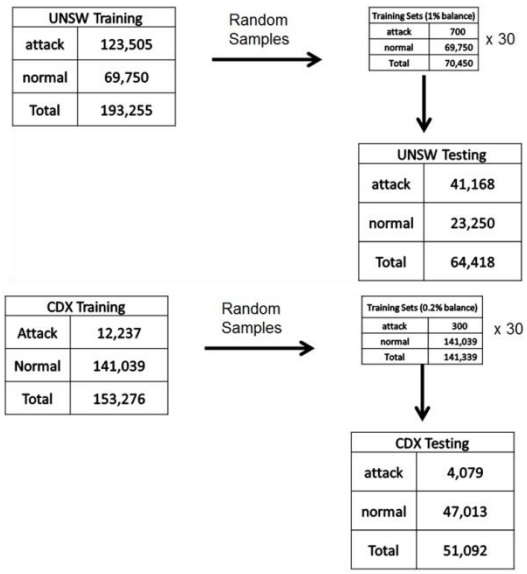

Figure 12: Traditional Dataset Partitions

To create a realistic imbalance in a training set, we can simply discard the majority of the attack observations and train on the resulting unbalanced data. In the USNW dataset, we want to create a 1% balance and only use 700 of the original attack observations for training. For contrast, we create a more unbalanced dataset in the CDX data and retain only 300 observations for an imbalance of 0.2% in the training set. This creates a dilemma in trying to figure out which data to discard. If we randomly choose a sample of 700 that are representative of the UNSW data as a whole, this might yield overly optimistic results. The opposite could happen if we choose a set that poorly represents typical attack traffic. In the UNSW dataset alone, there are $\binom{123{,}505}{700}$ combinations of 700 attack observations. We are unable to test them all so we resort to random sampling. We choose a sample size of 30 that will allow us to average results across the samples and use traditional testing methods to determine significant differences among sampling techniques and classification models.

The next sampling question considered was the size and composition of the testing set. It might seem intuitive that the most realistic measure would come from testing on a set that reflects the true class imbalance among the classes. If we choose this route, we have the same problem as choosing how to unbalance the training data. We could also create multiple test sets through random sampling but this begins to become burdensome computationally. Each classifier from the thirty training samples would need to be tested on the thirty testing sets equating to 900 unique values per dataset. We would then need to compute these values for each unique combination of over and under sampling used in the training, which quickly becomes impractical. We choose to use the entire test set as shown in figure 12. While unrealistically large and imbalanced for both datasets, each gives us a large number of observations with which to test the various classifiers and sampling methods. This unique combination of imbalanced training and relatively balanced testing sets creates the need for careful consideration of accuracy measure, discussed in section 3.4.3. Figure 13 shows the sizes of the sampled training sets and the testing sets for each dataset.

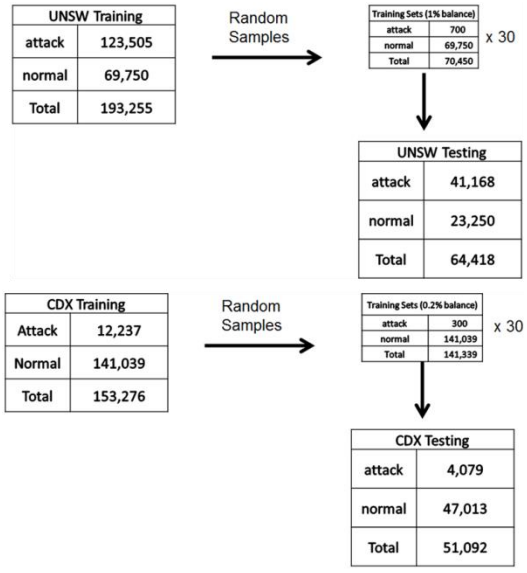

Figure 13: Sampled Dataset Partition

3.4.2 Under and Over Sampling

The simplest and most straightforward way to balance the training set is to under sample the majority class observations to create a more balanced training set. This has the added benefit of reducing the total size of the data saving computation time. The drawback, however is that we are throwing away perfectly good observations that our classifier could use to learn the structure of the majority class, the normal traffic in this case. We examine various levels of under sampling to create levels of balance between the original levels near 1% to a perfectly balanced training set of 50%. Using our original 30 unbalanced samples, we create an additional thirty samples for each level of imbalance for 240 new unique datasets. Figure 14 shows the various sizes. Note that as we increase the balance we are discarding, in each case, tens of thousands of perfectly good samples to achieve the desired level of balance.

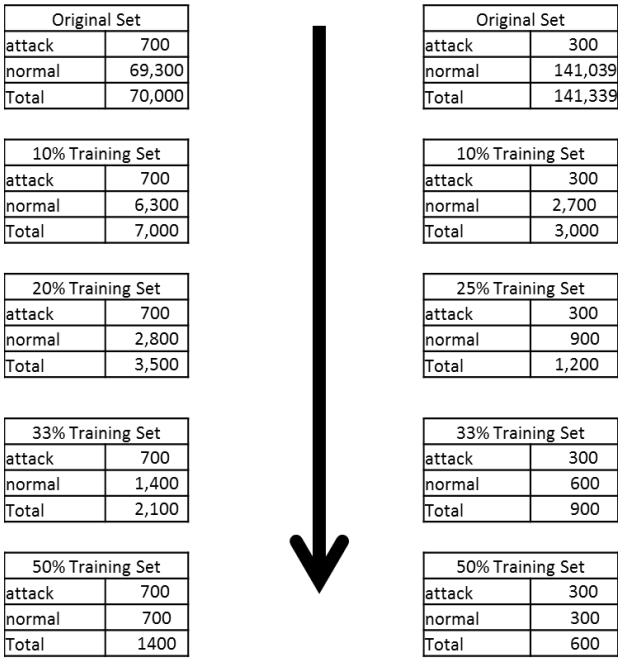

Figure 14: Under Sampled Training Sets

We next examine over sampling where we create an artificial balance in the data by creating new observations of the minority class, the attack traffic observations. There are several ways in literature to create new data, which were covered in chapter two. For this research, we examine three methods. The first is simply sampling the minority class with replacement until we reach the desired balance in the training set. We refer to this method by its more common name, bootstrapping. Bootstrapping creates a numeric balance by adding observations but introduces no new additional information, as the new observations are just replicas of the original data. This complicates using cross validation for parameter selection since we cannot partition this data using any resampling technique as identical points will inevitably exist in the training and testing sets biasing the results. We cover parameter selection and cross validation in section 3.4.4.

The two remaining techniques for over sampling are SMOTE and ROSE. Using each of the three over sampling methods, we over sample the attack traffic to remedy the imbalance to varying degrees. We begin at the original levels and create more and more observations until we reach more reasonably balanced samples. Figures 15 and 16 show the levels of oversampling for the two datasets. Because the data begins with so few attack observations, this means that the overwhelming majority of the attack data is either synthetic or copies of the original data. At three levels of oversampling with three separate oversampling techniques, we have created over 540 unique datasets from the original 60 samples.

Bootstrap

Original Set	
attack	700
normal	69,300
Total	70,000

10% Training Set	
attack	7800
normal	69,300
Total	77,100

25% Training Set	
attack	23,300
normal	69,300
Total	92,600

50% Training Set	
attack	69,300
normal	69,300
Total	138,600

SMOTE

Original Set	
attack	700
normal	69,300
Total	70,000

10% Training Set	
attack	7800
normal	69,300
Total	77,100

25% Training Set	
attack	23,300
normal	69,300
Total	92,600

50% Training Set	
attack	69,300
normal	69,300
Total	138,600

ROSE

Original Set	
attack	700
normal	69,300
Total	70,000

10% Training Set	
attack	7800
normal	69,300
Total	77,100

25% Training Set	
attack	23,300
normal	69,300
Total	92,600

50% Training Set	
attack	69,300
normal	69,300
Total	138,600

Figure 15: Over Sampled Training Sets-UNSW

Bootstrap

Original Set	
attack	300
normal	141,039
Total	141,339

10% Training Set	
attack	15,300
normal	141,039
Total	156,339

20% Training Set	
attack	35,000
normal	141,039
Total	176,039

33% Training Set	
attack	70,000
normal	141,039
Total	211,039

50% Training Set	
attack	141,039
normal	141,039
Total	282, 078

SMOTE

Original Set	
attack	300
normal	141,039
Total	141,339

10% Training Set	
attack	15,300
normal	141,039
Total	156,339

20% Training Set	
attack	35,000
normal	141,039
Total	176,039

33% Training Set	
attack	70,000
normal	141,039
Total	211,039

50% Training Set	
attack	141,039
normal	141,039
Total	282, 078

ROSE

Original Set	
attack	300
normal	141,039
Total	141,339

10% Training Set	
attack	15,300
normal	141,039
Total	156,339

20% Training Set	
attack	35,000
normal	141,039
Total	176,039

33% Training Set	
attack	70,000
normal	141,039
Total	211,039

50% Training Set	
attack	141,039
normal	141,039
Total	282, 078

Figure 16: Over Sampled Training Sets-CDX

Initially, we strategically examine over and under sampling separately although the two are not mutually exclusive. In fact, it is common to combine the two strategies. We will examine

promising combinations of over and under sampling based on the results of the over and under sampling. We discuss the choice for these sampling levels in detail in chapter 4. The table below shows a rollup of the unique datasets created by varying the sampling technique and level of class balance. For the majority of the datasets we train and tested on 30 unique samples created from the original data but reduced the number to twenty for several of the larger over sampled datasets in the interest of time. We note that training the four classifiers on each of these 1,245 datasets creates over 4,980 unique classifiers.

Table 8: Rollup of Datasets and Classifiers Created

Data Set	USNW	CDX
Over/Under Sampling Combinations	22	25
Samples per Combination	20-30	20-30
Unique Datasets	665	580
Total Unique Datasets	1,245	
Classifiers	4	
Total Unique Classifiers	**4,980**	

3.4.3 Accuracy Measures

Before we begin the discussion on accuracy measures, we first define some notation borrowed from Fawcett [101]. For binary classification problems, the most common way to present the results of prediction is in the form of a confusion matrix also called an error matrix or a contingency table. Traditionally, the rows represent the actual class membership and the columns represent the predicted class membership. Using Table 9 as an example, we see that the diagonals of the matrix, a and d, represent accurate predictions. The off-diagonals represent incorrect predictions. A negative observation classified as positive is a false positive (FP),

represented by b. A positive observation misclassified as a negative observation is a false negative (FP), represented by c. We further define some common terms below based on these four variables in the confusion matrix:

Table 9: Confusion Matrix Example

		Actual	
		Negative	Positive
Predicted	Negative	a	c
	Positive	b	d

- Accuracy (*AC*) is the proportion of the total number of predictions that are correct, calculated with $AC = \frac{a+d}{a+b+c+d}$
- Sensitivity or true positive rate (TPR) is the proportion of positive cases that are correctly identified, calculated with $TPR = \frac{d}{c+d}$
- False positive rate (FPR) is the proportion of negatives cases that are incorrectly classified as positive. One minus the FPR is often called the model specificity. FPR is calculated with $FPR = \frac{b}{a+b}$

The choice of accuracy measure is not trivial as the composition of the test data can make common measures biased or down right misleading [102]. Consider the traditional overall accuracy measure. If we were to use a test set with a balance reflective of typical network traffic where malicious traffic was 1% of the total, we could achieve 99% accuracy by simply declaring all the observations to be normal but this would hardly be useful to a network security professional. Sensitivity is also of little use on its own. Sensitivity near 100% is ideal but if this also yields a large number of false positives, it may provide little benefit. Consider an example where 100,000 observations only contain 100 positive cases. A 100% sensitivity rate coupled with a 30% FPR implies that we classified 30,100 observations as suspicious but over 99% of these are, in fact, benign. We have essentially taken a needle-in-a-haystack problem and reduced

the size of the haystack only slightly. A single measure of accuracy that incorporates elements of sensitivity and FPR is often desired for purposes of model comparison. Such measures are numerous in literature [103] but we first look at the most common technique, receiver operating characteristic (ROC) curves.

First developed by radar technicians in the 1940s to contrast a system's ability to correctly identify objects of interest, ROC curves became popular in the machine learning community to contrast sensitivity with FPR [104]. The sensitivity is plotted on the y-axis and the FPR is on the x-axis. Using a model's predictions for a given cutoff score, we can plot a continuous line in ROC space contrasting the tradeoffs between sensitivity and FPR. This can be used to determine a preferred cutoff score or as an overall value for a model's predictive accuracy by taking the area under the ROC curve (AUC). Figure 17 shows an example of a ROC curve plot with the AUC shown in the blue shaded region. The dashed line represents the equivalent of a random guess sometimes called the line of no discrimination. We note that the ideal performance is in the upper left hand corner (0, 1) where we have perfect sensitivity with zero false positives.

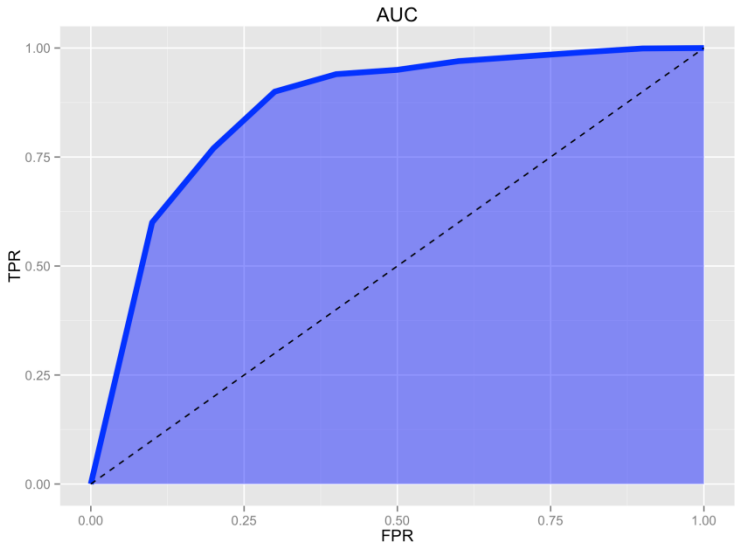

Figure 17: ROC Curve Example

While AUC is an excellent measure for comparison between two models, this research seeks to compare the average performance of models using multiple samples. Instead of attempting to aggregate the AUC across samples into a single line, we look at two computationally simpler measures that provide similar summary information of model performance in ROC space. The first is Youden's J-Statistic sometimes referred to as informedness [105]. The J statistic is calculated as Sensitivity + Specificity -1. Geometrically the J statistic is the distance from a point on the ROC curve to the line of no discrimination on the diagonal. First developed to aid in medical testing, the intuition behind the J statistic is to give equal weight to false positives and false negatives while bounding the scale between zero and one for simplicity. The second measure is also a measurement of distance in ROC space. The ROC Curve Euclidian Distance (ROCCED) [106] is simply the Euclidean distance from a point in ROC space to the upper left corner representing perfect classification. Figure 18 shows a graphical example of ROCCED and the J statistic using the ROC curve example from figure 17.

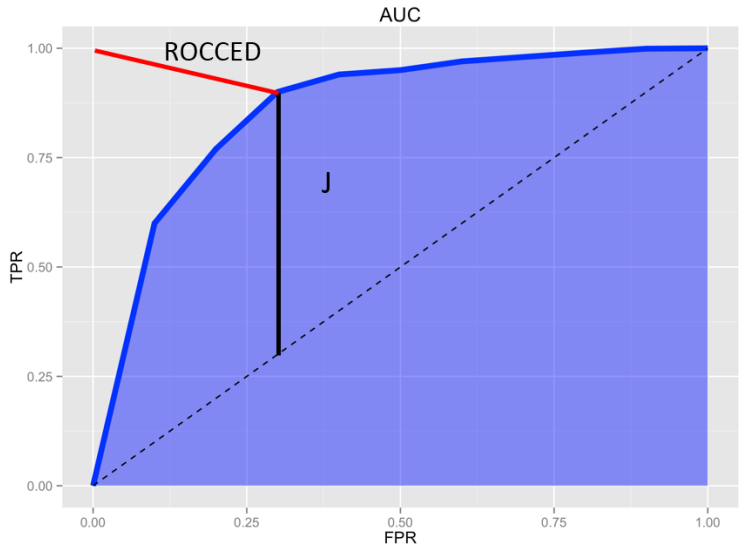

Figure 18: ROCCED and J Statistic example

When comparing two models, the J statistic and ROCCED will usually agree but are not guaranteed to reach the same conclusions. We calculate both measures but will primarily show the ROCCED in chapter 4.

3.4.4 Parameter Selection

While sophisticated classification methods like those discussed in section 3.3.1 frequently outperform simpler methods like logistic regression and discriminant analysis, they require us to set several model parameters. The choice of parameter levels can have a significant impact on overall performance and must be done with careful consideration. The most common method for parameter tuning is through cross validation using some kind of resampling and testing within the training dataset. There are in existence a number of resampling techniques each with its strengths and weaknesses. We refer the reader to Arlot and Celisse for an overview of the most popular methods in literature [107].

Literature suggests that the most effective and common are leave-one-out-cross validation (LOOCV), bootstrapping and its variants, and k-fold cross validation [108]. LOOCV involves removing one observation and training on each combination of n-1 samples. This exhaustive approach is simple but can be computationally expensive for use in larger datasets. Consider that the basic unbalanced UNSW training datasets where n=70,000 (700 attack, 69,300 normal) would require us to train 70,000 models for each classifier. Bootstrapping simply samples with replacement a specified number of times and tests on the out-of-bag samples (OOB). Efron and Tibshirani introduced an improved method specifically for cross-validation called the 0.632 + bootstrap [109]. K-fold cross validation involves a similar procedure to LOOCV but instead of leaving one observation out, a portion of size k is left out for testing. We train on each combination of the k-1 portions and use the k^{th} fold for testing. Five and ten are the

most common sizes for k. When k=10, we essentially train on 90% of the data and test on the remaining 10%, ten times with each fold left out once for testing. We then average the performance to calculate the chosen accuracy measure.

The LOOCV method is clearly impractical for this research, which leaves us with either bootstrapping or k-fold. The research conducted by Molinaro et al. [108] suggest that 0.632 + bootstrapping can exhibit more bias than k-fold for many types of data. While k-fold does exhibit bias, it can be reduced by repeating the procedure and averaging across the k-folds. For this thesis, we use 10-fold cross validation repeated three times giving us thirty distinct models to view performance before settling on choice of model parameters.

The difficulty with cross-validation in this research is the sheer number of distinct datasets. With three classifiers and 840 distinct datasets (60 original, 240 under sampled, 540 over sampled), this would require that we train and test 2,520 distinct classification models. Consider that we have two tuning parameters for neural networks, the number of nodes in the hidden layer and the decay parameter. If we choose four distinct values for each, we have eight possible combinations to test. With 10-fold cross validation, this implies that we train a neural network on each of those eight combinations ten times. We see that to test each at even just a small handful of parameter levels quickly becomes impractical. This is further complicated by the fact that many of the over sampled datasets cannot be effectively resampled. The bootstrapped datasets contain numerous duplicates that will inevitably end up in both the training and testing sets during cross validation creating biased and overly optimistic results. The SMOTE and ROSE sampling datasets do not contain exact duplicates but are still problematic as the synthetic observations share a common heritage from the original samples.

For these reasons, we performed cross validation only on a handful of the original and under sampled datasets. Fortunately, we find that for most of the combination of tuning parameters the differences in performance are small. We show some example results in the tables below. Each contains the average ROCCED and the ROCCED standard deviation (ROCCED SD) for each combination of parameters. For neural networks, we tried various combinations with the number of nodes varying from 10 to 20 and between 0 and 1 for the decay parameter. Table 10 shows example results from the UNSW dataset. The first is a balanced set (700 attack/700 normal) and the second comes from a dataset with only a 10% balance (700 attack/6300 normal). Table 11 shows similar results from the CDX data. The first is balanced (300 attack/300 normal) and the second is unbalanced (300 attack/ 2700 normal). The best two combinations of parameters as measured by ROCCED are shown in bold. Based on these results and others not shown, the most promising combination for neural networks using the UNSW data is 20 nodes and a decay parameter of 0.1. For the CDX data, we use only 10 nodes and a decay parameter of 0.01.

Table 10: UNSW Neural network Parameter Tuning Results

Balanced training set example

# of Nodes	decay	ROCCED	ROCCED SD
10	0	0.185	0.033
10	0.01	0.169	0.043
10	***0.1***	***0.152***	***0.030***
10	0.5	0.155	0.035
12	0	0.185	0.040
12	0.01	0.163	0.034
12	0.1	0.162	0.037
12	0.5	0.153	0.035
15	0	0.191	0.045
15	0.01	0.187	0.041
15	0.1	0.153	0.031
15	0.5	0.155	0.037
20	0	0.201	0.041
20	0.01	0.174	0.038
20	***0.1***	***0.152***	***0.039***
20	0.5	0.156	0.038

Unbalanced training set example

# of Nodes	decay	ROCCED	ROCCED SD
10	0	0.263	0.029
10	0.01	0.259	0.025
10	0.1	0.267	0.028
10	0.5	0.285	0.034
12	0	0.252	0.039
12	0.01	0.253	0.033
12	0.1	0.255	0.031
12	0.5	0.278	0.030
15	0	0.256	0.033
15	0.01	0.258	0.034
15	0.1	0.256	0.034
15	0.5	0.277	0.031
20	0	0.248	0.026
20	***0.01***	***0.251***	***0.031***
20	***0.1***	***0.251***	***0.034***
20	0.5	0.273	0.032

Table 11: CDX Neural network Parameter Tuning Results

Balanced training set example

# of Nodes	decay	ROCCED	ROCCED SD
8	0	0.138	0.040
8	0.01	0.130	0.044
8	0.1	0.149	0.048
8	0.5	0.145	0.052
10	0	0.145	0.057
10	***0.01***	***0.130***	***0.040***
10	0.1	0.151	0.045
10	0.5	0.157	0.050
12	0	0.133	0.041
12	0.01	0.133	0.041
12	0.1	0.151	0.050
12	0.5	0.155	0.055
15	0	0.136	0.041
15	***0.01***	***0.130***	***0.042***
15	0.1	0.150	0.051
15	0.5	0.158	0.053

Unbalanced training set example

# of Nodes	decay	ROCCED	ROCCED SD
8	0	0.280	0.022
8	0.01	0.280	0.021
8	0.1	0.284	0.020
8	0.5	0.309	0.024
10	***0***	***0.270***	***0.024***
10	***0.01***	***0.269***	***0.019***
10	0.1	0.279	0.021
10	0.5	0.309	0.024
12	0	0.271	0.024
12	0.01	0.276	0.020
12	0.1	0.291	0.019
12	0.5	0.294	0.025
15	0	0.255	0.023
15	0.01	0.271	0.019
15	0.1	0.287	0.020
15	0.5	0.299	0.025

We show similar results for random forests and SVMs in tables 12 through 15. We tried m_{try} values from four to ten. The default setting for m_{try} in the random Forest R package [85] is to use the square root of the number of features. Our results largely conclude with this rule of thumb. We use an m_{try} of eight for the UNSW data and five for the CDX data.

Table 12: UNSW Random Forest Parameter Tuning Results

Balanced training set example

m_{try}	ROCCED	ROCCED SD
4	0.149	0.077
5	0.143	0.076
6	0.143	0.076
8	**0.139**	**0.073**
10	**0.139**	**0.073**

Unbalanced training set example

m_{try}	ROCCED	ROCCED SD
4	0.254	0.039
5	0.244	0.039
6	0.241	0.038
8	**0.235**	**0.039**
10	**0.234**	**0.039**

Table 13: CDX Random Forest Parameter Tuning

Balanced training set example

m_{try}	ROCCED	ROCCED SD
4	0.106	0.058
5	0.104	0.057
6	0.108	0.061
8	0.117	0.062
10	0.121	0.064

Unbalanced training set example

m_{try}	ROCCED	ROCCED SD
4	0.360	0.080
5	0.362	0.075
6	0.363	0.087
8	0.376	0.092
10	0.391	0.087

Tables 14 and 15 show the SVM results. We examined six levels of the cost parameter, *C*, ranging from one to 32. As in the other results, there is little practical difference between the examined levels. While *C*=32 appears to be promising, it may risk overfitting when we measure against the test set so we use a value of 16 for both datasets. Table 16 summarizes the parameter choices for each classifier and dataset.

Table 14: UNSW SVM Parameter Tuning Results

Balanced training set example

Cost parameter	ROCCED	ROCCED SD
1	0.179	0.047
2	0.166	0.046
4	0.160	0.046
8	**0.158**	**0.048**
16	**0.152**	**0.045**
32	0.159	0.044

Unbalanced training set example

Cost parameter	ROCCED	ROCCED SD
1	0.347	0.026
2	0.338	0.025
4	0.322	0.028
8	0.321	0.028
16	**0.317**	**0.025**
32	**0.316**	**0.026**

Table 15: CDX SVM Parameter Tuning Results

Balanced training set example

Cost parameter	ROCCED	ROCCED SD
1	0.148	0.051
2	0.144	0.055
4	0.144	0.050
8	**0.144**	**0.051**
16	**0.139**	**0.055**
32	0.141	0.051

Unbalanced training set example

Cost parameter	ROCCED	ROCCED SD
1	0.372	0.049
2	0.320	0.053
4	0.297	0.049
8	0.297	0.049
16	**0.293**	**0.048**
32	**0.292**	**0.047**

Table 16: Model Parameter Summary

Model	Parameter	UNSW	CDX
Neural Network	number of nodes	20	10
	decay parameter	0.1	0.01
Random Forest	m_{try}	8	5
SVM	cost parameter	16	16
Gradient Boosting	Number of rounds	100	50
	Depth	3	2
	Shrinkage parameter	0.3	0.4

IV. Results

4.1 Unbalanced Results

To examine the effects of the class imbalance, we train each classifier on the initial balanced samples and then on the thirty samples with a realistic balance. Each combination is then tested on the hold-out test set described in section III. The charts below confirm the negative effects of the class imbalance. Figure 19 shows the performance of each classifier on the UNSW data in ROC space with the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis. Each point represents either the single instance of the original balanced data or the mean of the thirty unbalanced training datasets. The type of classifiers is designated by shape and the balance level by color. We notice that for each classifier exceptional accuracy is possible with the large balanced samples in the blue figures noting that random forest and SVM achieved near perfect accuracy. When we discard the majority of the attack observations and train on a realistic balance, shown in red, we see that the classifiers do become biased towards the majority class with near zero FPRs and substantially lower TPRs.

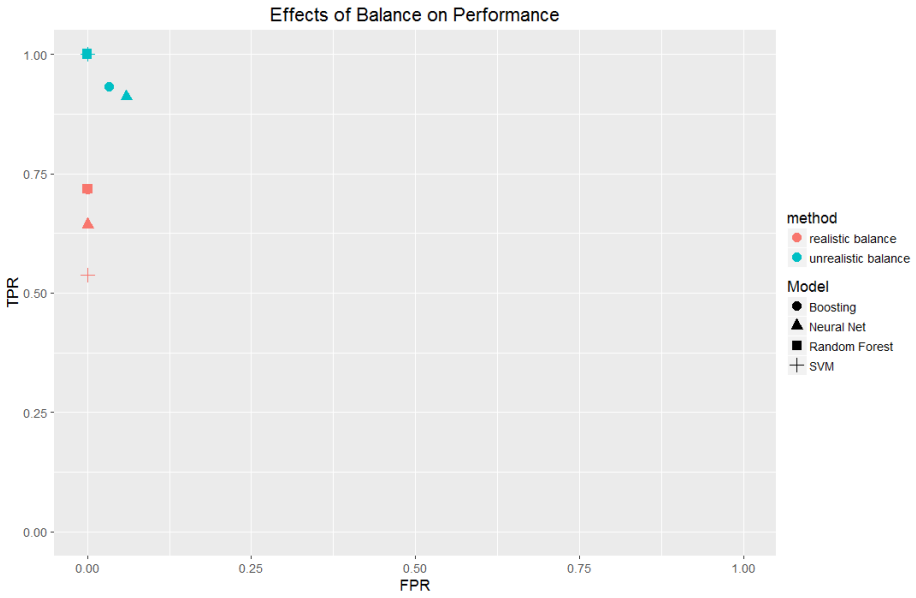

Figure 19: UNSW Unbalanced Performance Comparisons

The CDX data shows a similar phenomenon in Figure 20 with a more pronounced reduction in performance due to the more extreme class imbalance of 0.2% versus 1% in the UNSW data.

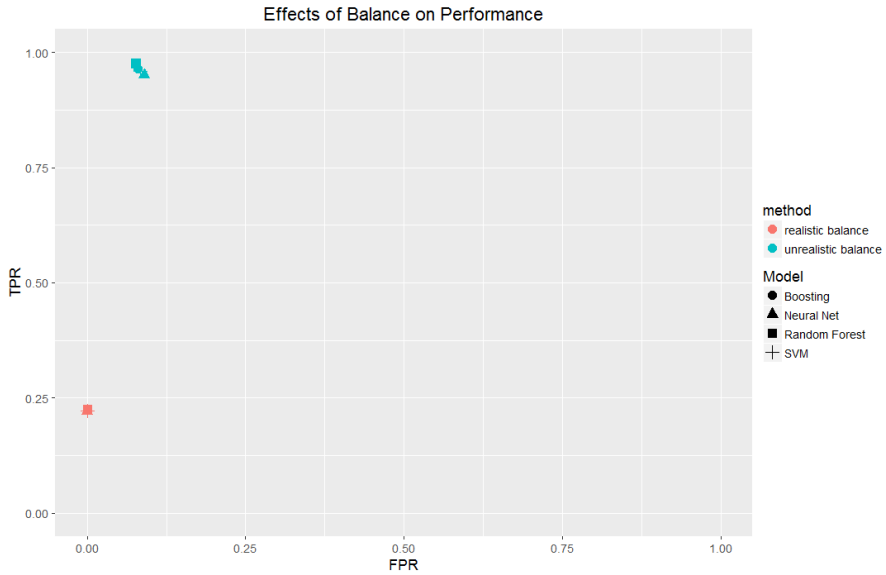

Figure 20: CDX Unbalanced Performance Comparisons

The two charts above show the problems of unbalanced data but do not imply a binary choice between the two levels of performance. We expect that with different sampling techniques we can improve on the initial unbalanced performance levels. We examine these techniques in the following sections.

4.2 Under Sampling

This section examines the performance of each classifier using under sampling of the normal class observations. We keep the number of attack observations fixed and under sample the normal class at the levels described in section III. We display the results of the under sampling for the UNSW data below in figure 21. The figure on the left shows the performance of each level of balance as measured by the mean ROCCED value for the thirty samples. Each of the four classifiers is designated by color. We annotate each point with a label showing the number of attack observations on the left and the number of normal observations on the right. As a reference, the right figure shows a scatter plot with the mean TPR and FPR as shown earlier in figures 19 and 20 to show the source of the improvement in ROC space. To distinguish the classifiers, we plot the points on the right graph by both color and shape. We note that each classifier shows improvement as we continue to discard the normal observations until we reach a perfect class balance of 700 observations of each class. We see in the right figure that we begin to generate more false positives but a much greater number of true positives represented by the movement toward the (0,1) point in the upper left corner representing perfect classification accuracy.

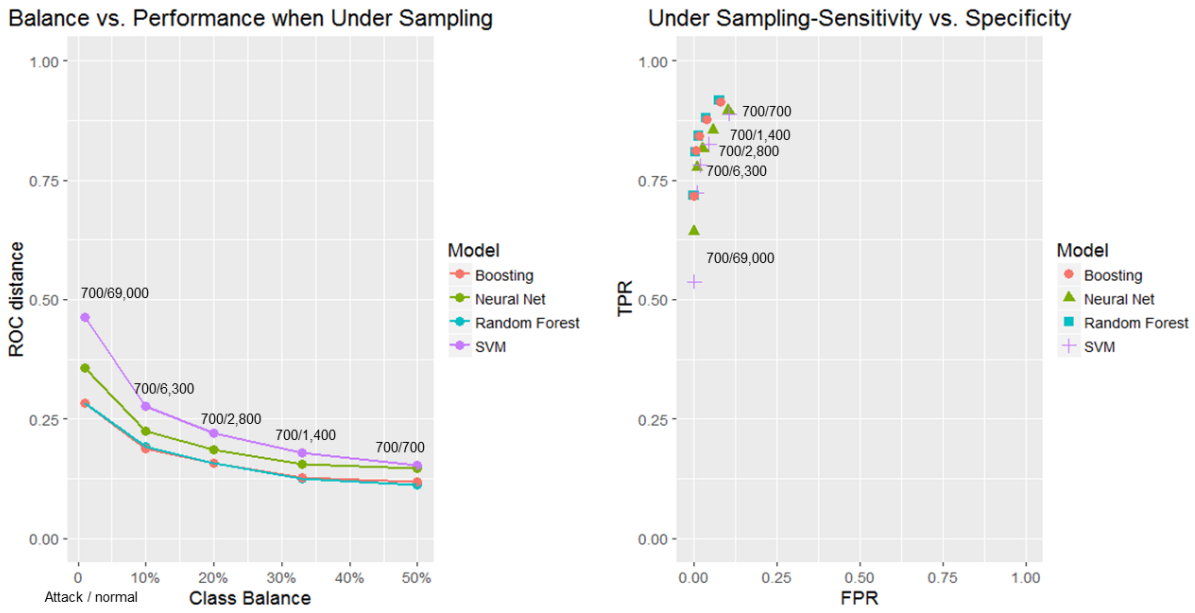

Figure 21: UNSW Under Sampling Performance

Figure 22 shows the same behavior in the CDX data. Due to the more extreme imbalance, we see a much larger improvement as we move to a 10% balance but the same basic behavior with performance increasing as we approach a perfect class balance of 300 observations of each class for all four classifiers.

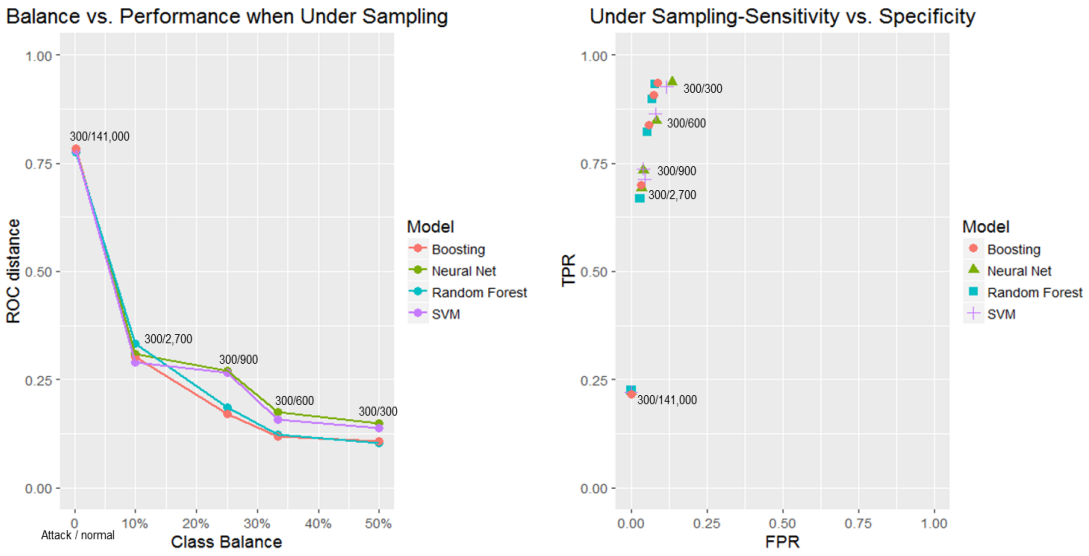

Figure 22: CDX Under Sampling Performance

Both datasets show that the ensemble methods, random forest and gradient boosting, appear to be superior to the single models but very close to each other. We show a 95% confidence interval calculated from the t-distribution for each of the four classifiers in the UNSW dataset at the 50% class balance level in table 17 and a graphical representation in figure 23. We see that the separation between each classifier is statistically significant although perhaps not practically significant between the random forest and boosting classifiers.

Table 17: UNSW Best Under Sampling Confidence Intervals

Classifier	Lower 95% CI	Mean ROCCED	Upper 95% CI
RF	0.1118	**0.1132**	0.1147
GB	0.1165	**0.1182**	0.1200
NNET	0.1449	**0.1467**	0.1485
SVM	0.1529	**0.1543**	0.1557

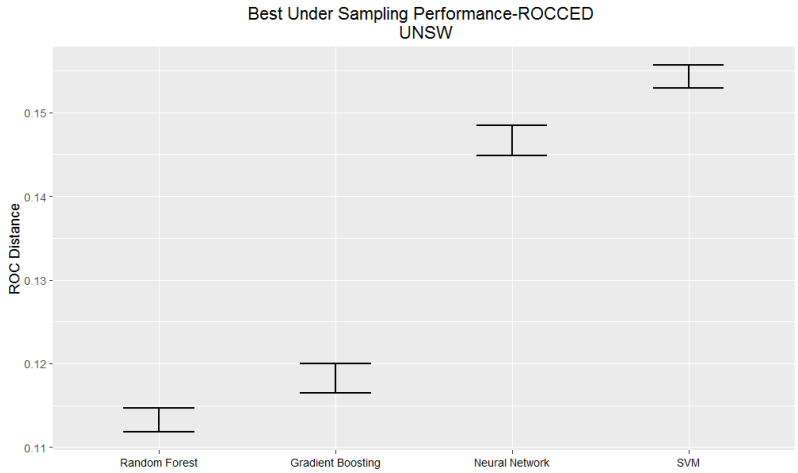

Figure 23: UNSW Best Under Sampling Confidence Intervals

The CDX data shows a similar breakdown in table 18 and figure 24 although with overlaps between random forest and gradient boosting and a change of position between SVM and neural networks.

Table 18: CDX Best Under Sampling Confidence Intervals

Classifier	Lower 95% CI	Mean ROCCED	Upper 95% CI
RF	0.1021	**0.1040**	0.1059
GB	0.1048	**0.1073**	0.1099
SVM	0.1352	**0.1393**	0.1434
NNET	0.1464	**0.1494**	0.1525

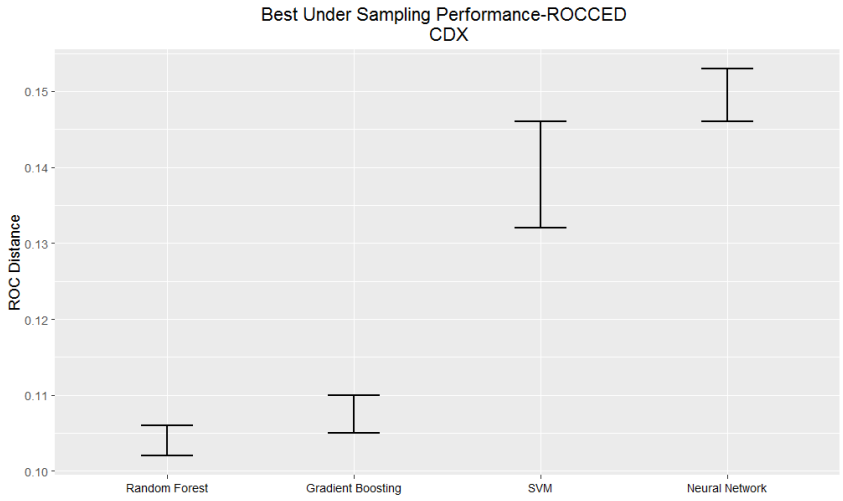

Figure 24: CDX Best Under Sampling Confidence Intervals

4.3 Over Sampling

In this section we examine the results of over sampling where we artificially increase the number of attack observations with the two synthetic techniques covered in section 2.4, SMOTE and ROSE, and traditional sampling with replacement referred to here as bootstrapping. We increase the attack observations with each of these methods while keeping the all of the normal observations. The performance of the over sampling methods varied by classifier type and with some variation between the UNSW and CDX datasets. For comparisons, we include the under sampling results in the charts in this section. In the remainder of the section, we will examine the results of over and under sampling by classifier. We see that generally under sampling outperforms over sampling but with some exceptions. Figure 25 shows the results of over sampling and the three methods of oversampling for neural networks in the UNSW dataset with each sampling technique indicated by color.

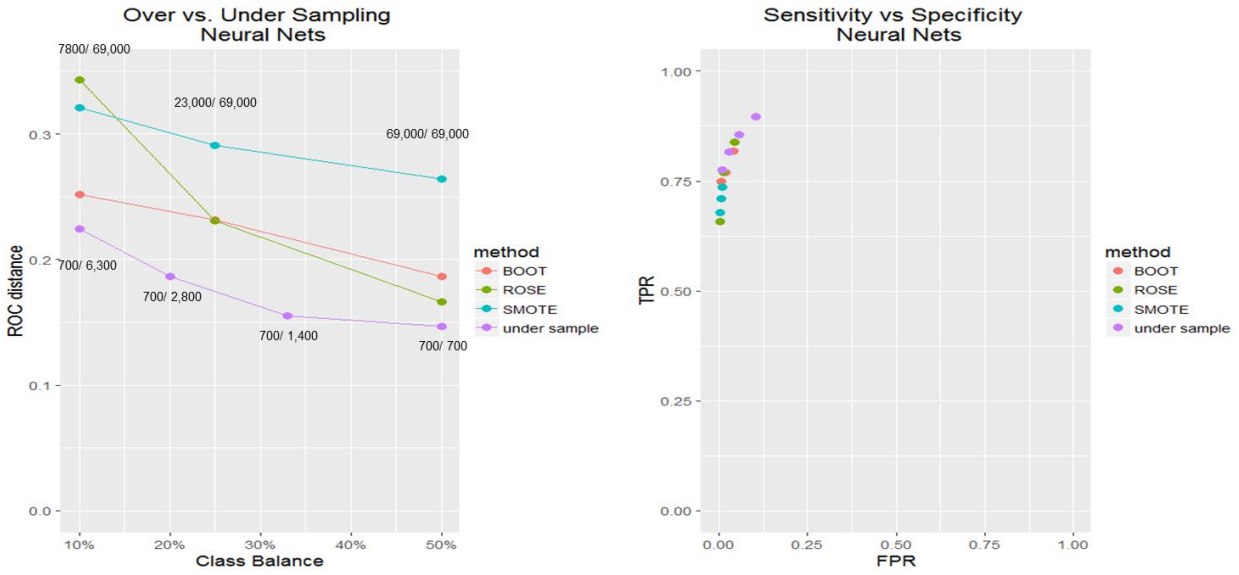

Figure 25: Neural network Performance Under vs. Over Sampling (UNSW)

For the UNSW dataset, we see that increasing the over sampling improves each classifier, peaking at a balanced sample consisting of 69,000 observations of each class. We do note, however, that the performance of each over sampled classifier is less than that of the under sampling which shows greater performance with far fewer observations. A training set of 1,400 observations will generally be preferable to a training set of 138,000 observations. The CDX data paints a slightly different picture in Figure 26. Increasing the over sampling improves performance with two methods eclipsing the performance of the best under sampling. We show the results of the three best combinations in table 19 below.

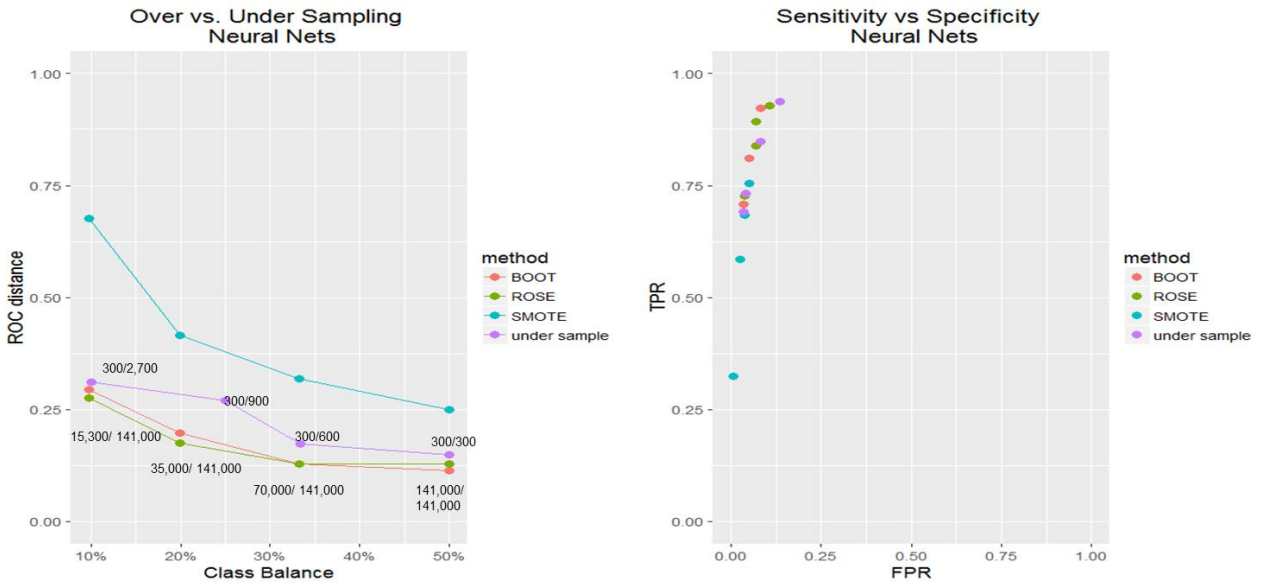

Figure 26: Neural network Performance Under vs. Over Sampling (CDX)

We see in table 19 that the two over sampling techniques are superior to the classifier trained with the under sampled dataset but at much greater time and computational costs. The under sampled training set requires only 600 observations which can complete training in seconds while the others contain hundreds of thousands of observations which require additional time and effort both to create the new data as well as training the classifier on a training dataset containing hundreds of thousands of observations.

Table 19: Best Neural network Sampling Combinations (CDX)

Method	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Over Sampling Bootstrap	141,000	141,000	0.1055	**0.1133**	0.1211
Over Sampling ROSE	70,000	141,000	0.1211	**0.1283**	0.1359
Under Sampling	300	300	0.1464	**0.1494**	0.1525

Support Vector Machines (SVM), like neural networks, generally perform better when trained with the under sampled data. The UNSW results show that the bootstrapped samples are close but the table 20 shows that there is a statistically significant separation.

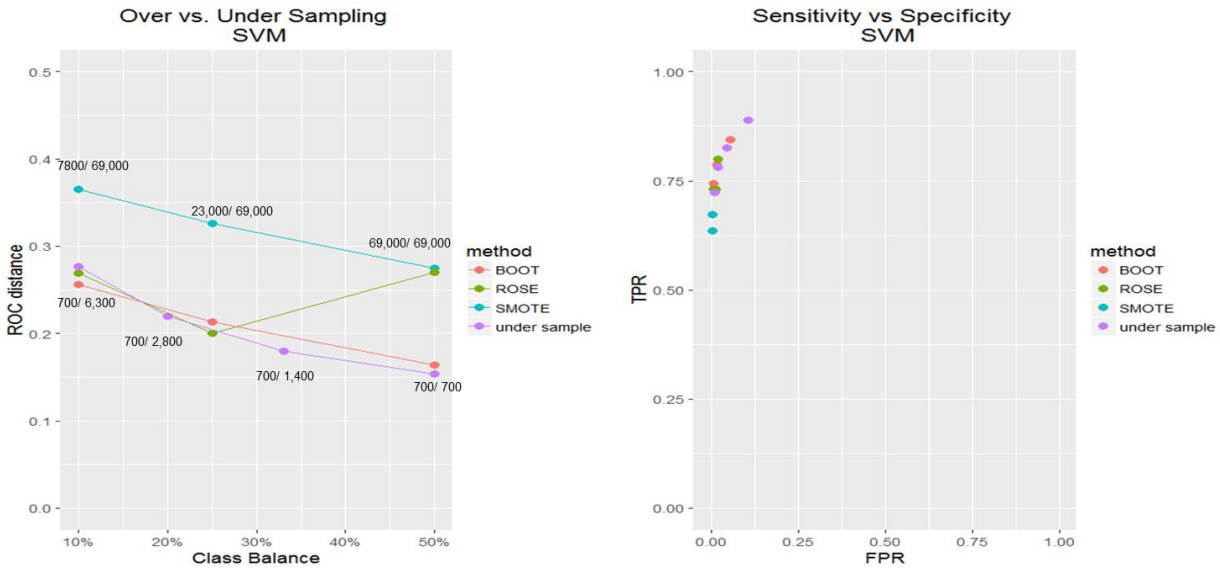

Figure 27: SVM Performance Under vs. Over Sampling (UNSW)

Table 20: Best SVM Sampling Combinations (UNSW)

Method	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Under Sampling	700	700	0.1529	0.1543	0.1557
Over Sampling Bootstrap	69,300	69,300	0.1620	0.1644	0.1668

Like the UNSW data, the SVMs trained on the CDX data improve little from SMOTE or ROSE data but the bootstrapped data is more than competitive with under sampling. Figure 28 and table 21 show a clear separation albeit with a much higher computational burden.

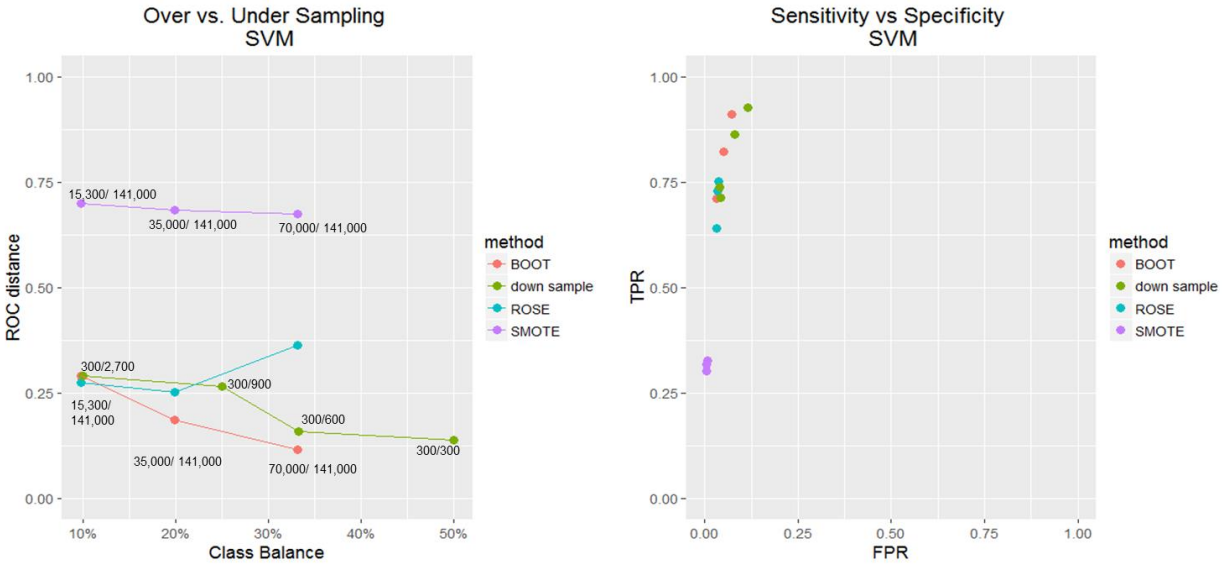

Figure 28: SVM Performance Under vs. Over Sampling (CDX)

Table 21: Best SVM Sampling Combinations (CDX)

Method	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Over Sampling Bootstrap	70,000	141,000	0.1086	0.1151	0.1216
Under Sampling	300	300	0.1352	0.1393	0.1434

We next examine the over sampled data with the random forest classifier. Random forests are clearly not improved by the introduction of additional samples either from repeated observations with bootstrapping or the synthetic data from SMOTE and ROSE. Figure 30 shows the stark difference in performance in the left figure. We note from the right figure the large number of false positives generated from training on the synthetic ROSE data.

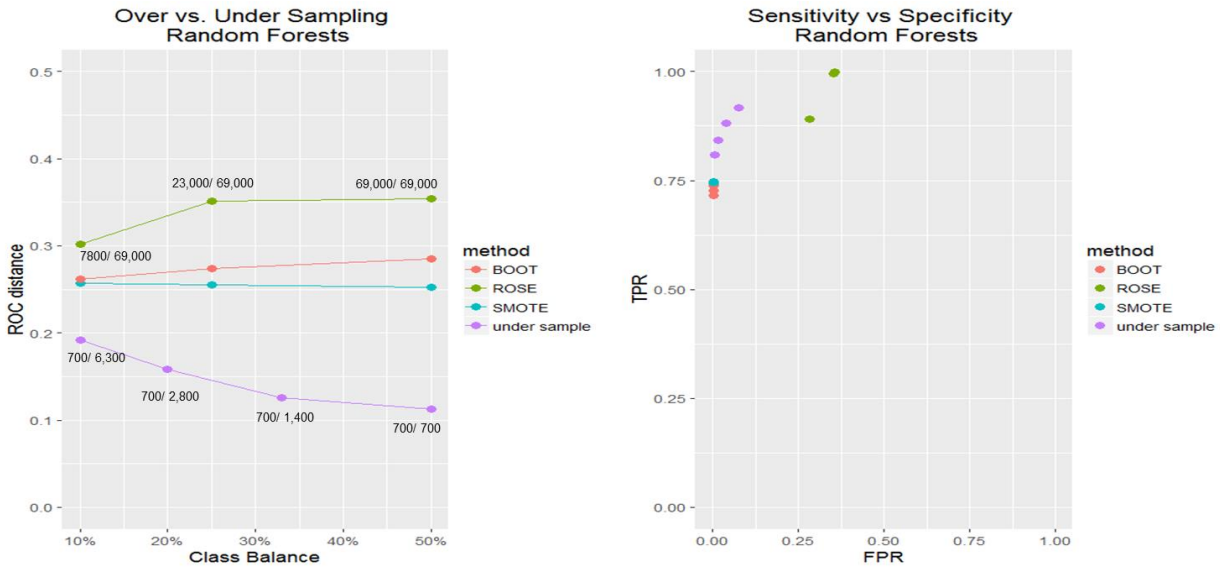

Figure 29: Random Forest Performance Under vs. Over Sampling (UNSW)

Examining the CDX data, we see the same general trend in figure 31 but with some improvement from the bootstrapped samples. The mean ROCCED values with their confidence intervals are in table 22 showing the clear superiority from the under sampling method.

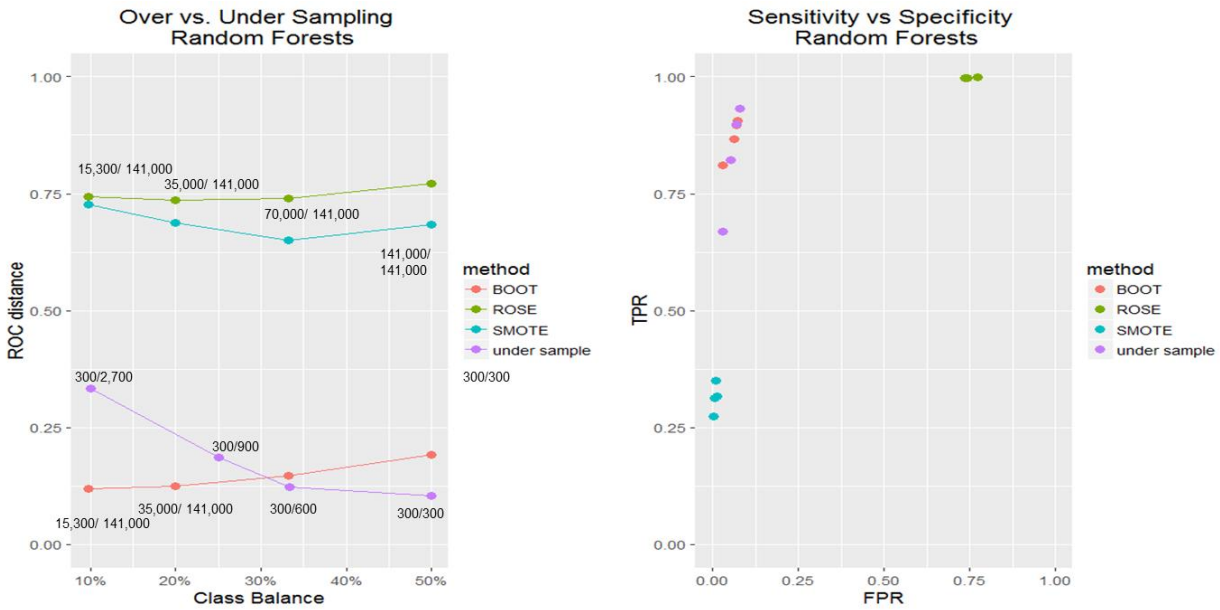

Figure 30: Random Forest Performance Under vs. Over Sampling (CDX)

Table 22: Best Random Forest Sampling Combinations (CDX)

Method	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Under Sampling	300	300	0.1021	0.1040	0.1059
Over Sampling Bootstrap	15,300	141,000	0.1138	0.1192	0.1246
Over Sampling Bootstrap	35,000	141,000	0.1212	0.1249	0.1286

Gradient Boosting, another tree based ensemble method, shows results similar to those of Random Forests but with a more striking result for the ROSE datasets. The under sampled data is clearly superior but we note the extremes of each level of ROSE that are pushed to the extreme upper right hand corner of the right side graphic in figure 32. With an FPR of nearly 1, the gradient boosting classifier is predicting nearly all of the observations to be attack traffic. We show later in section 4.5 that the ROSE method introduced some noise to the datasets. Servedio

and Long showed that boosting classifiers are highly susceptible to failure in the presence of even a small amount of noise in the data [100]. We explore some techniques to address the noisy data in section 4.5.

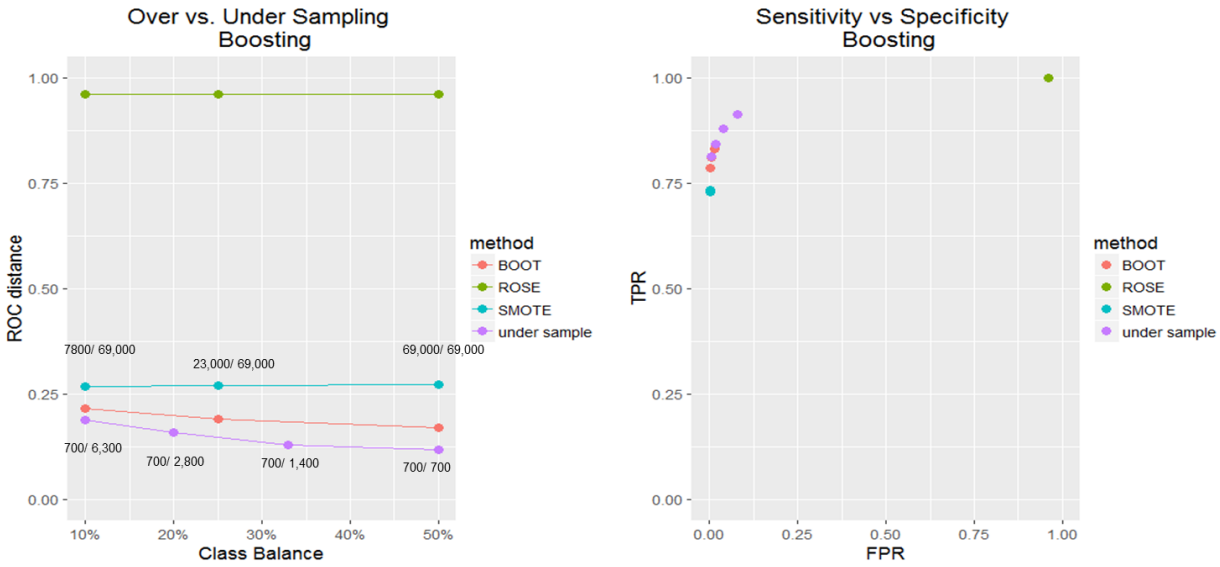

Figure 31: Gradient Boosting Performance Under vs. Over Sampling (UNSW)

The CDX data shows a similar pattern for SMOTE and ROSE as seen in the UNSW data but bootstrapping appears to be competitive with under sampling. The data in table 23 shows the overlap in the confidence intervals. Given the orders of magnitude more data required for the bootstrapped classifier, we credit a slight advantage to the under sampling method.

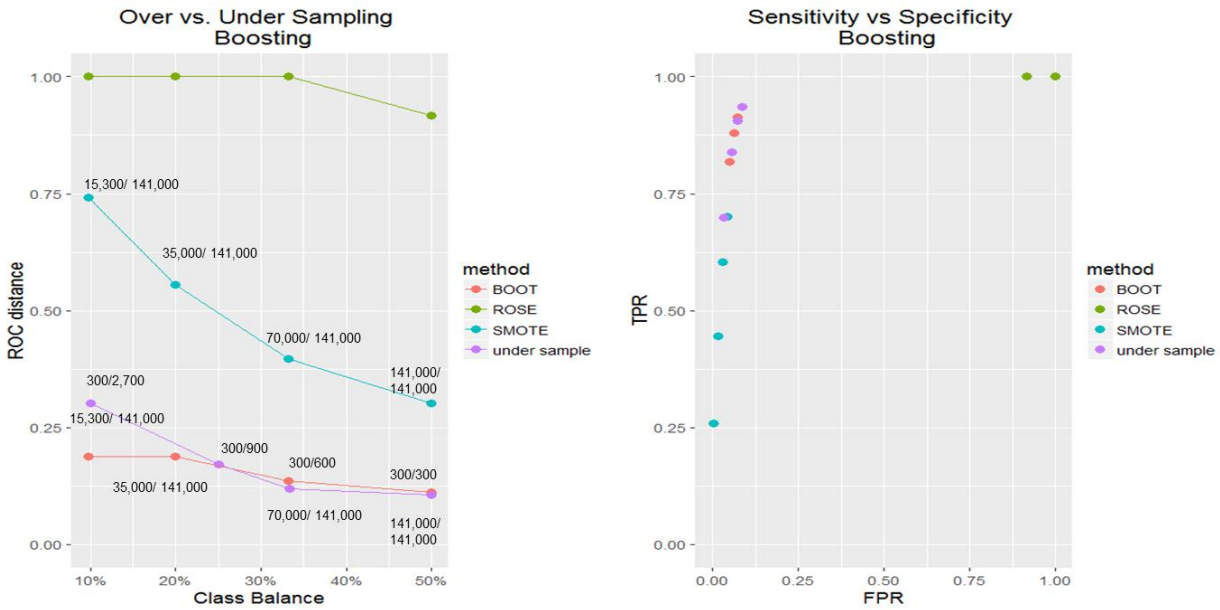

Figure 32: Gradient Boosting Performance Under vs. Over Sampling (CDX)

Table 23: Best Gradient Boosting Sampling Combinations (CDX)

Method	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Under Sampling	300	300	0.1048	0.1073	0.1098
Over Sampling-Bootstrap	141,000	141,000	0.1092	0.1118	0.1145

4.4 Combinations of Over and Under Sampling

We see from sections 4.2 and 4.3 that in six of the eight combinations of classifier and sampling technique, that under sampling yielded the best performance. We examined under and over sampling separately for experimental purposes but these techniques are not mutually exclusive. In this section, we test promising combinations of over and under sampling for each dataset. We bias towards under sampling and balanced datasets due to the results from the

previous section and to create smaller more practical training datasets. The various combinations of over and under sampling are shown for each dataset in figures 33 and 34.

Bootstrap

Bootstrap 1	
attack	700
normal	700
Total	1,400

Bootstrap 2	
attack	1,400
normal	1,400
Total	2,800

Bootstrap 3	
attack	3,500
normal	3,500
Total	7,000

SMOTE

SMOTE 1	
attack	700
normal	700
Total	1,400

SMOTE 2	
Attack	1,400
Normal	1,400
Total	2,800

SMOTE 3	
attack	3,500
normal	3,500
Total	7,000

ROSE

ROSE 1	
attack	700
normal	700
Total	1,400

ROSE 2	
attack	1,400
normal	1,400
Total	2,800

ROSE 3	
attack	3,500
normal	3,500
Total	7,000

Figure 33: Over and Under Sampling Combinations (UNSW)

Bootstrap

Bootstrap 1	
attack	600
normal	600
Total	1,200

Bootstrap 2	
attack	900
normal	900
Total	1,800

Bootstrap 3	
attack	1,500
normal	1,500
Total	3,000

SMOTE

SMOTE 1	
attack	600
normal	600
Total	1,200

SMOTE 2	
Attack	900
Normal	900
Total	1,800

SMOTE 3	
attack	1,500
normal	1,500
Total	3,000

ROSE

ROSE 1	
attack	600
normal	600
Total	1,200

ROSE 2	
attack	900
normal	900
Total	1,800

ROSE 3	
attack	1,500
normal	1,500
Total	3,000

Figure 34: Over and Under Sampling Combinations (CDX)

The above nine combinations of over and under sampling increase coupled with four levels of under sampling and 12 levels of over sampling create 25 distinct sampling combinations. Results for each combination in both datasets are included in appendix B. For the sake of brevity, we show and discuss the top ten for each dataset as well as the best five for each classifier. Tables 24 and 25 show the best combinations of classifier and sampling technique for the UNSW and CDX datasets respectively. We see that random forest and gradient boosting

dominate the top ten for the UNSW dataset. Strictly under sampling shows a slight improvement over the combination techniques. We note the appearance of SMOTE in the top ten list but point out that the SMOTE data is not significantly different from its bootstrapped analogue for either the random forest or gradient boosting classifiers.

Table 24: Top Ten Classifier-Sampling Combinations (UNSW)

Classifier	Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Random Forest	Under Sampling	700	700	0.1118	**0.1132**	0.1147
Gradient Boosting	Under Sampling	700	700	0.1165	**0.1182**	0.1200
Random Forest	Combination SMOTE	1400	1400	0.1192	**0.1217**	0.1243
Gradient Boosting	Combination SMOTE	1400	1400	0.1228	**0.1246**	0.1264
Random Forest	Under Sampling	700	1400	0.1231	**0.1257**	0.1283
Random Forest	Combination Bootstrap	1400	1400	0.1236	**0.1258**	0.1281
Gradient Boosting	Combination Bootstrap	1400	1400	0.1251	**0.1274**	0.1297
Gradient Boosting	Under Sampling	700	1400	0.1261	**0.1282**	0.1302
Gradient Boosting	Combination Bootstrap	2100	2100	0.1267	**0.1370**	0.1472
Random Forest	Combination Bootstrap	2100	2100	0.1285	**0.1391**	0.1496

The CDX data also shows the superiority of random forests and boosting but with over sampling and two single model classifiers cracking the top ten. We conjecture that the CDX classifiers show more improvement from over sampling due to the smaller number of observations. We expect to see more benefit from over sampling in datasets containing fewer observations of the minority class.

Table 25: Top Ten Classifier-Sampling Combinations (CDX)

Classifier	Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Random Forest	Under Sampling	300	300	0.1021	**0.1040**	0.1059
Gradient Boosting	Combination Bootstrap	1500	1500	0.1040	**0.1068**	0.1097
Gradient Boosting	Under Sampling	300	300	0.1048	**0.1073**	0.1099
Gradient Boosting	Combination Bootstrap	900	900	0.1068	**0.1089**	0.1111
Gradient Boosting	Over Sampling Bootstrap	141000	141000	0.1092	**0.1119**	0.1145
Gradient Boosting	Combination Bootstrap	600	600	0.1084	**0.1122**	0.1161
Neural Network	Over Sampling Bootstrap	141000	141000	0.1055	**0.1133**	0.1211
SVM	Over Sampling Bootstrap	70000	141000	0.1086	**0.1151**	0.1216
Random Forest	Combination Bootstrap	600	600	0.1145	**0.1188**	0.1232
SVM	Combination Bootstrap	1500	1500	0.1166	**0.1191**	0.1217

We next examine the top five combinations for the single classifiers that made only brief appearances for the top ten overall. The best combinations for each dataset for neural networks are listed in tables 26 and 27. Unlike the tree based ensemble methods, neural networks saw improved accuracy with the introduction of large numbers of synthetic data via over sampling in the CDX data.

Table 26: Top Five Neural Network Combinations (UNSW)

Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Under Sampling	700	700	0.1449	**0.1467**	0.1485
Combination SMOTE	1400	1400	0.1483	**0.1503**	0.1523
Combination Bootstrap	1400	1400	0.1533	**0.1553**	0.1573
Under Sampling	700	1400	0.1532	**0.1557**	0.1581
Combination Bootstrap	3500	3500	0.1522	**0.1621**	0.1720

Table 27: Top Five Neural Network Combinations (CDX)

Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Over Sampling Bootstrap	141000	141000	0.1055	**0.1133**	0.1211
Over Sampling ROSE	70000	141000	0.1209	**0.1284**	0.1359
Over Sampling Bootstrap	70000	141000	0.1200	**0.1285**	0.1370
Over Sampling ROSE	141000	141000	0.1224	**0.1300**	0.1377
Combination Bootstrap	1500	1500	0.1362	**0.1382**	0.1402

The top five for SVMs are similar to neural networks with under sampling and combinations featuring more under sampling achieving the best results in the UNSW data. Conversely, over sampling alone achieves the best results in the CDX dataset but with a wide

confidence interval suggesting that the much smaller bootstrapped dataset may be equal in performance with much less computational effort.

Table 28: Top Five SVM Combinations (UNSW)

Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Under Sampling	700	700	0.1529	**0.1543**	0.1557
Combination Bootstrap	1400	1400	0.1568	**0.1590**	0.1613
Combination ROSE	1400	1400	0.1581	**0.1612**	0.1642
Over Sampling Bootstrap	69300	69300	0.1620	**0.1644**	0.1667
Combination Bootstrap	2100	2100	0.1514	**0.1662**	0.1810

Table 29: Top Five SVM Combinations (CDX)

Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
Over Sampling Bootstrap	70000	141000	0.1086	**0.1151**	0.1216
Combination Bootstrap	1500	1500	0.1166	**0.1191**	0.1217
Combination Bootstrap	900	900	0.1176	**0.1216**	0.1256
Combination Bootstrap	600	600	0.1286	**0.1338**	0.1390
Under Sampling	300	300	0.1352	**0.1393**	0.1434

In this section we saw that the tree-based ensemble methods provide greater classification accuracy than their single model counterparts. We also saw that under sampling or combinations of under sampling with moderate amount of over sampling provided better results than over

sampling alone. In the few cases where over sampling achieved slightly better results than the other methods, the size of the datasets are likely to be prohibitive for the time sensitive nature of intrusion detection. While these results are similar to other studies comparing the performance of these classifiers [110], we defer from making any pronouncements about the superiority of any particular classifier from these results. Hand [111] cautions against such pronouncements of classifier superiority noting that results, "obtained in laboratory conditions may not translate to superiority in real-world conditions and, in particular, the apparent superiority of highly sophisticated methods may be illusory."

We have examined these techniques with well-labeled data and largely noise free features. In the dynamic world of intrusion detection, constantly changing adversary behavior may make such training datasets unlikely to occur in the real world. Stephen Jou, the chief technology officer (CTO) of the cyber security firm Interset, describes a common technique where malicious actors can 'hide in plain sight' and train a model to view their behavior as normal called model poisoning [112]. To combat such behavior, professionals deploying a machine learning based IDS must continue to search for new data and new features to train their systems to identify the newest forms of malicious traffic. Domingos [113] calls this process, known as feature engineering, the most important factor in the success or failure of creating a classifier. Feature engineering involves much trial and error and often adds an element of noise to training data. In the next sections, we examine classifier performance in the context of noisy data using the synthetic datasets

4.5 Classification in the Presence of Noise

In this section, we first examine some of the issues of using kernel density estimates employed by the ROSE method. The synthetic ROSE data inadvertently blurred some of the class boundaries within several features. We intend it to be not a criticism of ROSE, which is a novel approach to synthetic data creation but is likely, not appropriate for this type of data. The ROSE data does, however, inject noise into the training data providing a method to view the classifiers outside the "laboratory conditions" mentioned in section 4.4.

ROSE employs a Gaussian kernel which, with an appropriate choice of h or a scaling parameter, can accurately estimate a density function even for data that is not necessarily Gaussian. The drawback for using a Gaussian kernel is that it is not bounded by zero like many distributions in the exponential family. A cursory inspection of the ROSE generated data show that there were some negative values for features that measure time or file size which obviously cannot exist in reality. A small number of negative values need not hinder a particular feature's ability to teach a classifier about the class boundaries within that feature if the simulated values reasonably match the real-world data. In some of the features, this was not the case and it appears that the tree based classifiers created spurious partitions in the data.

We examine the source packet average inter-arrival time abbreviated, sinpkt, as an example. Packet inter-arrival times are commonly very small, typically in small fractions of a second. In certain applications, an internet protocol (IP) conversation may stay open with long gaps between packet arrivals. We see in the histograms from figure 35, that the sinpkt feature are most commonly near zero for both classes, however, the normal traffic has some very large times, several of which are near six thousand seconds. The attack traffic has a fairly small standard deviation of 52.6 while the standard deviation of the normal traffic is over 11,000. The

ROSE package sets the initial *h* for each feature as the standard deviation with a unique value for each class. Ideally, we want to craft an appropriate *h* or scaling parameter for each feature. Unfortunately, the ROSE package [55] only allows for a single scaling parameter for each class that it applies to every feature. Knowing that several features had very large standard deviations, we chose a very small scaling parameter of 0.05 for the UNSW data and 0.01 for the CDX data. This created reasonable density estimates for most of the features in the dataset with some exceptions, notably features related to arrival times like sinpkt.

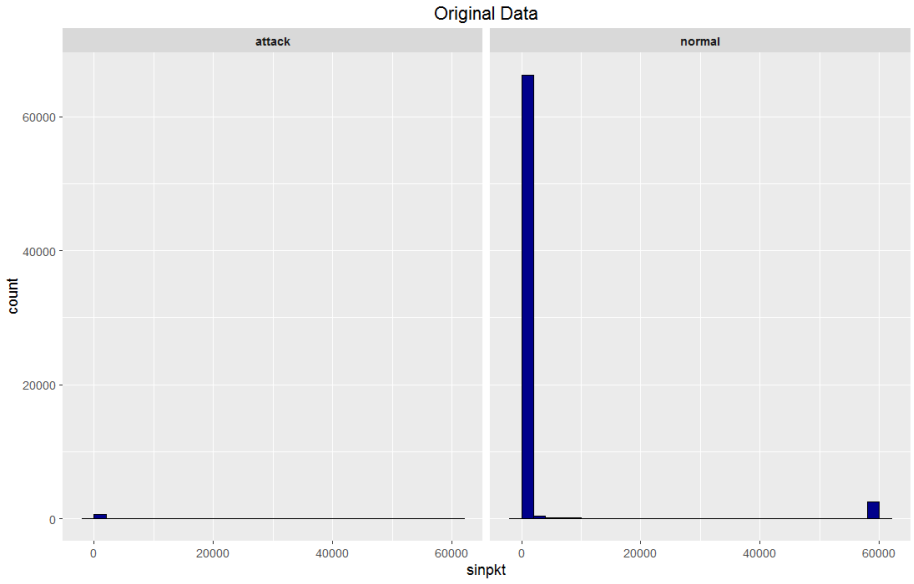

Figure 35: Original source packet inter-arrival times (UNSW)

Figure 36 shows histograms of the sinpkt feature from one of the ROSE data files. We note that both have negative values but the normal data shows that nearly half of the generated values are negative with many very large absolute values. In the random forest classifier, the algorithm picks only a few features at a time and chooses the best to partition the data. It's unlikely to choose sinpkt using the original data as most of the observations are near zero save a few large normal observations. The ROSE data shown in figure 36, however, allows the data to create a

partition the data at -100 and cleanly separate almost half of the normal observations from the attack observations.

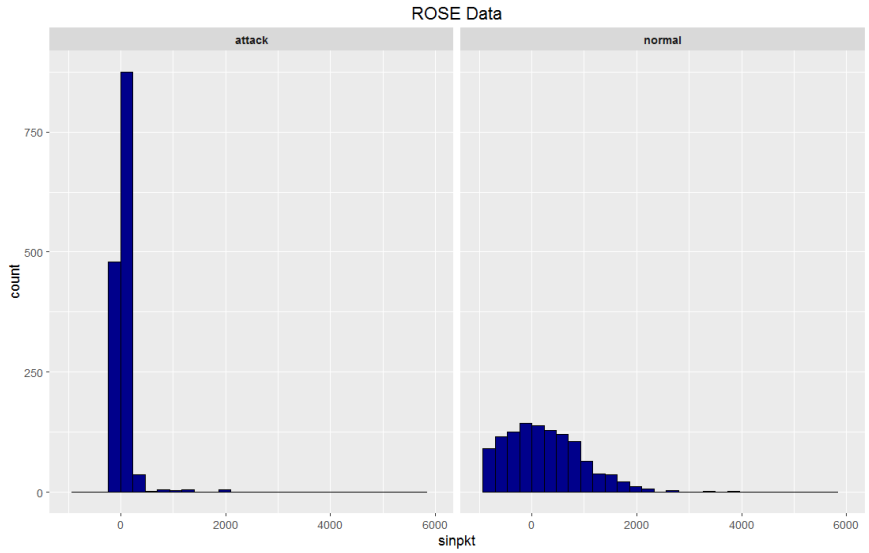

Figure 36: ROSE generated source packet inter-arrival times (UNSW)

In most software's implementation of the random forest algorithm, a feature importance score proposed by Breiman is calculated based on how well the trees can partition the data when a particular feature is used. The reader is referred to Breiman [81] for a detailed description. With this score, we can confirm our suspicion of the spurious splits from sinpkt. We calculate these variable importance scores, scale them so that the most important feature is 100, and plot them using the caret package. Figure 37 shows that with the ROSE data, the classifier determined that sinpkt was the fourth most important feature in the dataset. The full names and descriptions of the abbreviated feature names on the y-axis can be found in table 4 from section 2.

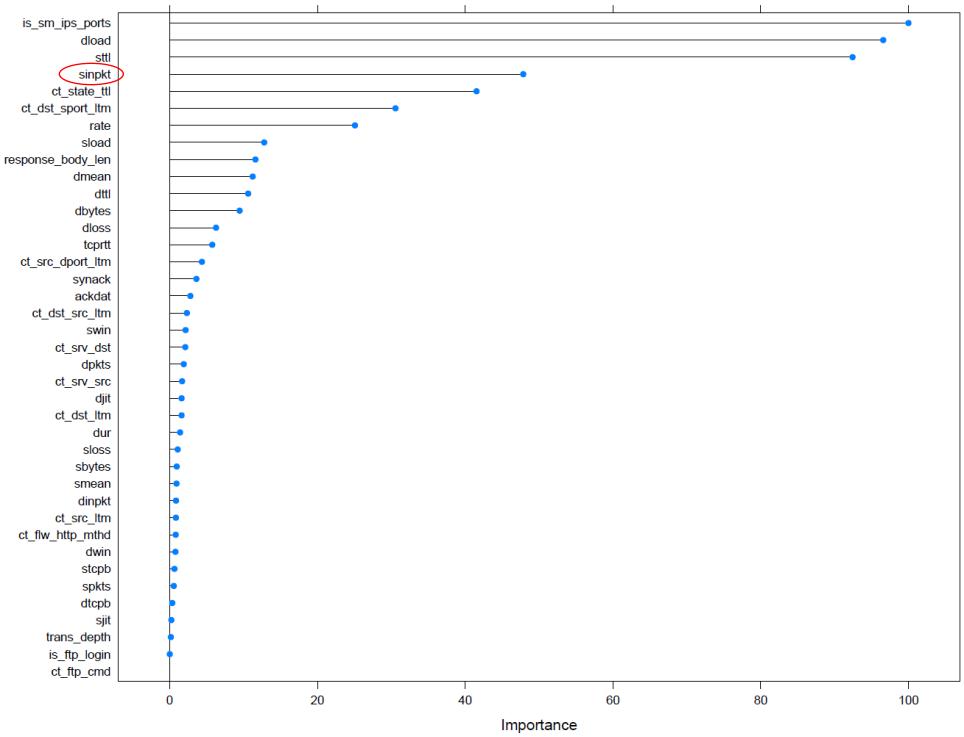

Figure 37: Random Forest Feature Importance-ROSE data

These spurious splits, as expected, do not generalize to the test data where we will not see negative inter-arrival times. Figure 38 shows the results of the over sampling data using ROSE for each of the four classifiers. We note that both tree methods, which are the most accurate using the original data, collapse when trained with the ROSE data. We also note the striking comparison of neural networks with the other classifiers. Continually adding the synthetic ROSE data increases the performance of neural networks while even a small amount pushes gradient boosting to the upper right hand corner in ROC space.

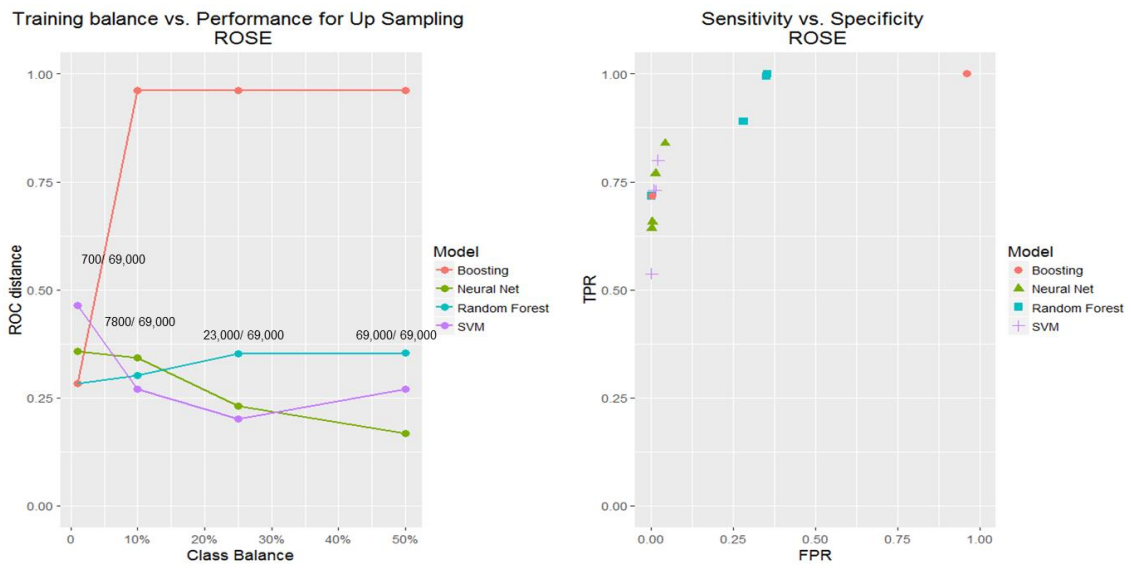

Figure 38: ROSE Over Sampling Performance (UNSW)

We next examine feature importance for neural networks with the ROSE data to compare with random forest. Unlike random forests, neural networks have no native methods to calculate feature importance. Several methods to measure feature importance for neural networks have been proposed [114] [115]. For this research, we use the variable importance measure developed by Gevrey et al. [116] which is a modification of the technique first proposed by Goh [117]. The method assigns importance by measuring the relative size of the absolute value of weights leaving each input node. We scale the importance measures to a maximum of 100 and plot them in Figure 39.

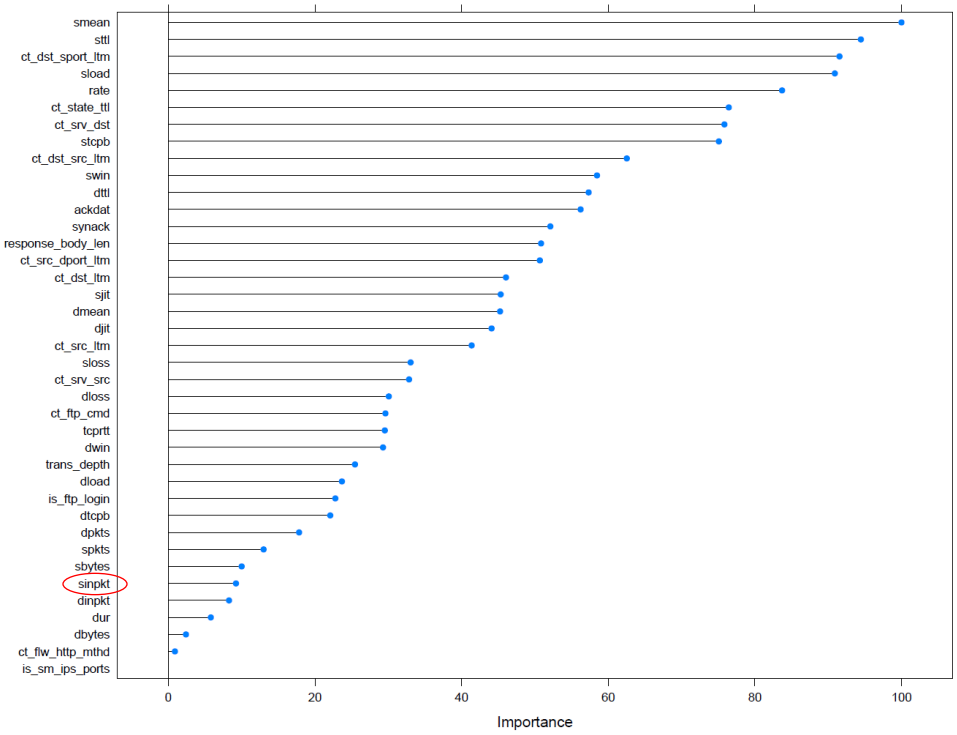

Figure 39: Neural Network Feature Importance-ROSE data

Figure 39 shows that neural networks correctly under weighted the spurious feature sinpkt. We posit that the structure of neural networks makes them better suited to handle noisy features than either Random Forest or Gradient Boosting. Neural networks consider each feature simultaneously, adjusting the weights at each iteration to minimize error. With enough iterations, a neural network will sort out the appropriate features by underweighting the bad features. Random Forests, however, do not view each feature simultaneously. By design, they only consider a portion of the features at a time in the hopes of finding local features normally missed by traditional bootstrap aggregated tree methods where a few features often dominate. The byproduct of this is the tendency to make spurious splits if confronted with too many noisy or irrelevant features. Gradient Boosting has an even bigger problem in that it is often drawn to noise in trying to solve hard to classify observations as pointed out by Long and Servedeo [100].

In the next section, we discuss the role of feature selection in improving the less robust classifiers with neural networks.

4.6 Neural Networks for Feature Selection and Ensembling

While sophisticated classifiers like Random Forests and Gradient Boosting are somewhat fragile, their fragility does not imply a mutually exclusive choice between them and more robust classifiers like neural networks. A number of feature selection methods can be applied to find the best features before training a less robust classifier. Such methods are numerous in literature. We mention the broad categories here using the terminology of John et al. [118]. The first major category of feature selection techniques are wrapper methods such as recursive feature elimination [119], genetic algorithms [120], and simulated annealing [121]. The other main category is filter methods. The reader is referred to Saeys et al. [122] for a thorough review of the many filter methods in literature. While each of these methods has strengths and weaknesses, they complicate the time sensitive intrusion detection process by adding another step to select features each time we retrain a classifier with either new data or new features. Pinto [123] suggests that this may need to happen every few days for a modern machine learning based IDS. Here we present an efficient method to use neural networks for feature selection. This allows for an initial estimate based on the neural network's prediction, which we can then later update with additional classifiers trained with the features selected by the initial neural network.

A neural network may rank features differently depending on the random starting weights. To get a more robust view of the features, we train the network multiple times with different starting weights and rank the features at each iteration. We first define some basic notation. S is the training set, p is the set of features, t is the variable importance score threshold, and M is the number of training iterations. The procedure follows the pseudo-code below:

For i = (1 to M)
1. train a neural network with S

2. compute $varImp_i \; \forall \; p \; \in S$
end
4. $AvgImp_p = \frac{1}{M} * \sum_{i=1}^{m} varImp_i$

5. Retain features, p', where $AvgImp_p \geq t$

We calculate the variable importance and scale to 100 with the Gevrey process described in section 4.5. Using 60 for the *t*, value we see promising results. We use 20 of the ROSE datasets from the USNW dataset containing 700 normal and 700 attack observations and retrain the tree-based classifiers with the p' features. Figure 40 shows the results. We note the drastic improvement in ROCCED with the elimination of the noisy and irrelevant features for both random forests and gradient boosting but neither eclipses the original neural network results.

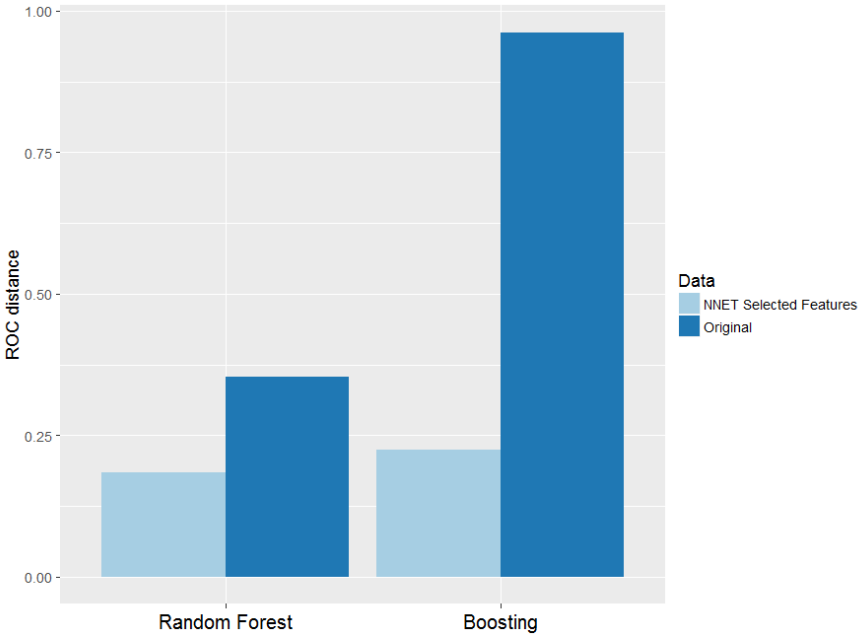

Figure 40: Classifiers Results with Neural Network Selected Features

With these improved classifiers, we now examine if these classifiers can be ensembled to improve on the original single model classifiers. To clarify terminology, random forests and gradient boosting methods are ensemble classifiers but using multiple instances of the same basic model type. These are referred to as homogenous classifiers. We can also ensemble classifiers of different types. These are referred to as heterogeneous classifiers. There exist in literature many methods to ensemble heterogeneous classifiers. The reader is referred to Reid [124] for a survey of the many ensemble methods for heterogeneous classifiers. Here we attempt the basic ensemble method (BEM) first proposed by Perrone and Cooper[125]. As the name implies, BEM is the simplest and most basic type of ensemble classifier. To calculate the predicted class we simply average the predicted class probabilities of each individual classifier. Table 30 shows some sample results from an ensemble created with a neural network and a random forest. The predicted class probabilities in columns two through four represent the probability of being an attack observation. We assign the predicted class with the traditional cutoff score of 0.50. Notice in the fourth observation, the random forest predicts the normal observation incorrectly but the neural network score brings the BEM score below the threshold to predict the class accurately. The opposite occurs in the fifth observation where the random forest's prediction corrects the neural network.

Table 30: Sample Basic Ensemble Method Predictions

Observation	NNET	RF	BEM	BEM Predicted Class	Actual Class
1	0.122	0.602	0.362	normal	normal
2	0.137	0.548	0.342	normal	normal
3	0.284	0.478	0.381	normal	normal
4	0.161	0.740	0.451	normal	normal
5	0.610	0.379	0.495	normal	normal

We create ensemble classifiers by combining the original neural network with the improved random forests classifier trained on the features selected by the neural network ranking procedure. In sixteen of the twenty runs, the random forest or the ensemble eclipsed the performance of the neural network. We show the performance in Figure 41.

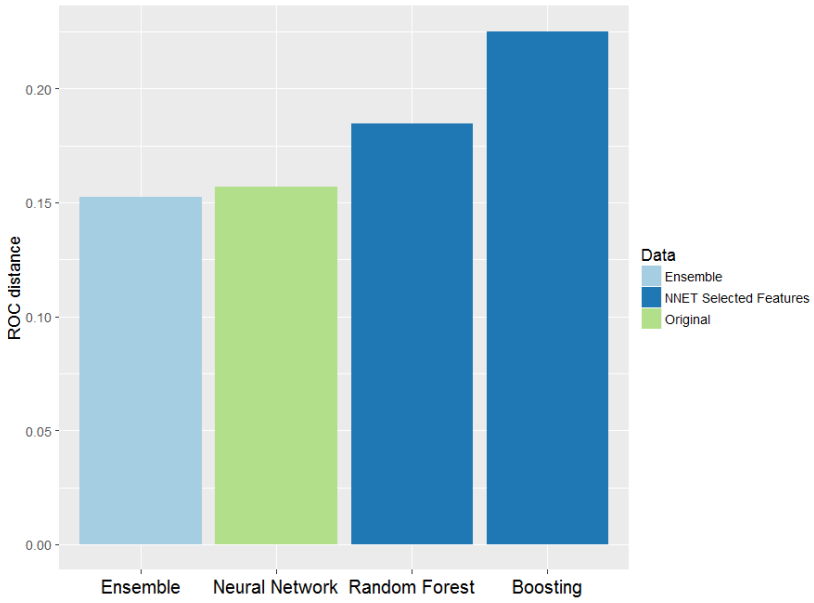

Figure 41: Results with Neural Network Selected Features

While the improvement over the individual classifier is small, the results are encouraging. More sophisticated ensembling methods such as stacking [126] or Bayesian Model Averaging (BMA) [127] have shown to outperform BEM in a number of domains. This is outside the scope of this research but speaks to the potential of using the neural networks as a precursor to other more powerful classifiers.

V. Conclusions and Recommendations

5.1 Conclusions

From the two datasets, we examined 47 combinations of over and under sampling. We trained and tested each combination on 20-30 samples for a grand total of 4,980 unique classifiers. From those classifiers, we can conclude that under sampling is generally more effective than over sampling for most of our classifiers. We saw that neural networks and SVMs improved using larger over sampled datasets in the CDX data that began with fewer observations of the attack traffic (700 vs 300). We note that the classifiers trained on the under sampled data were nearly as effective with a much lighter computational burden giving us confidence in our general pronouncement on the superiority of under sampling.

There are, of course, a number of caveats. We heed Wolpert's [128] no-free-lunch theorem knowing that we can never truly declare one machine learning technique or method as always superior to another. Different datasets with different features might show improvement with over sampling. We are confident, however, in recommending to professionals creating network intrusion detection systems with machine learning methods to first attempt the more promising under sampling technique. Only after seeing poor performance with under sampling should over sampling be considered. We also conclude that any over sampling should first look to traditional sampling with replacement or bootstrapping. Creating synthetic data with SMOTE and ROSE never statistically outpaced simple bootstrapping. Such methods also have the disadvantage of adding additional complexity and time in an already complicated and time sensitive process.

On the question of classifier performance, we also refrain from any firm declarations of superiority. We found the tree based ensemble methods of random forests and gradient boosting

to be superior when trained on most of our datasets but note the lack of realism using such laboratory conditions. The ROSE data that blurred the class distinctions in several features provided noisier datasets for comparisons closer to real world conditions. We found that neural networks robustness to noise make them an attractive candidate for both their accuracy in the presence of noisy features and their ability to help identify the most useful features for powerful but more fragile classifiers.

5.2 Contributions and Future Research

This research makes two primary contributions to the field of machine learning based network intrusion detection. The first is identifying the simpler method of under sampling as the most promising way to deal with class imbalance in problem. Related to this larger conclusion, we also expose synthetic data creation methods' inability to outperform simple bootstrapping for over sampling network intrusion data. The second contribution is the identification of neural networks utility for network intrusion detection despite its apparent inferior performance in laboratory conditions. Neural networks robustness and ability to simultaneously perform feature selection make them an attractive alternative in field of hundreds of potential classifiers.

For future research, we recommend further investigation of neural networks' ability to act as a precursor for a heterogeneous ensemble. We saw in this research how neural networks were able to find features that allowed fragile classifiers to contribute to a basic ensemble method (BEM). More sophisticated ensemble methods can likely improve on these results. We also recommend the use of chronological data for repeated training and testing. Measuring classifiers' ability to adapt to new data and possibly new features as adversaries alter their behavior and techniques would provide the most realistic view of network intrusion detection.

Appendix A. Hardware/Software overview

Other than minor record keeping of results with spreadsheets, the entirety of this research is conducted within the R statistical computing environment [129]. An outgrowth of the S language developed at Bell labs, R has become a popular open source language for statistics and data visualization [130]. Because R is open source and free, it has developed a large community of developers who collaborate to improve the software by recommending improvements and bug fixes. R version 3.2.1 [131], was used for this research. R's main advantage lies in the sharing of code and functions from the user community in the form of packages. The Comprehensive R Network (CRAN) site has posted over 7,000 packages since its inception. Table 31 summarizes the packages used in this thesis with a short description of each. The data preparation and predictive modeling all took place on a Hewlett-Packard 6305 desktop running Windows 7 with 8 GB of RAM and a 3.6 GHz AMD-A4 5300B processor.

Table 31: R Package Summary

Package	Author	Version	Purpose
nnet	Venables and Ripley [74]	7.3-9	Artificial neural networks
randomForest	Liaw and Werner [85]	4.6-10	Random Forests
kernlab	Karatzoglou et al. [91]	0.9-22	Support Vector Machines
xgboost	Chen et al. [98]	0.4-2	Gradient Boosting
caret	Kuhn [132]	6.0-58	Data partitioning, pre-processing, cross validation
DMwR	Torgo [46]	0.4.1	SMOTE data creation
ROSE	Lunardi et al. [55]	0.0-3	ROSE data creation
dplyr	Wickham [133]	0.4.3	Data sampling, reshaping, feature importance
ggplot2	Wickham [134]	1.01	Graphics/plotting

Appendix B. Full Results

UNSW Results

Rank	Classifier	Sampling Method	# of attack observations	# of normal observations	lower 95% CI	Mean ROCCED	upper 95% CI
1	RF	Under	700	700	0.112	0.113	0.115
2	GB	Under	700	700	0.116	0.118	0.120
3	RF	Combination-SMOTE	1400	1400	0.119	0.122	0.124
4	GB	Combination-SMOTE	1400	1400	0.123	0.125	0.126
5	RF	Under	700	1400	0.123	0.126	0.128
6	RF	Combination-BOOT	1400	1400	0.124	0.126	0.128
7	GB	Combination-BOOT	1400	1400	0.125	0.127	0.130
8	GB	Under	700	1400	0.126	0.128	0.130
9	GB	Combination-BOOT	2100	2100	0.127	0.137	0.147
10	RF	Combination-BOOT	2100	2100	0.129	0.139	0.150
11	GB	Combination-BOOT	1400	1400	0.135	0.142	0.150
12	NNET	Under	700	700	0.145	0.147	0.149
13	NNET	Combination-SMOTE	1400	1400	0.148	0.150	0.152
14	RF	Combination-BOOT	3500	3500	0.145	0.154	0.163
15	SVM	Under	700	700	0.153	0.154	0.156
16	NNET	Combination-BOOT	1400	1400	0.153	0.155	0.157
17	NNET	Under	700	1400	0.153	0.156	0.158
18	GB	Under	700	2800	0.156	0.158	0.161
19	RF	Under	700	2800	0.156	0.159	0.161
20	SVM	Combination-BOOT	1400	1400	0.157	0.159	0.161
21	GB	Combination-SMOTE	3500	3500	0.155	0.160	0.165
22	SVM	Combination-ROSE	1400	1400	0.158	0.161	0.164
23	NNET	Combination-BOOT	3500	3500	0.152	0.162	0.172
24	NNET	Combination-BOOT	2100	2100	0.151	0.163	0.176
25	SVM	Over-BOOT	69300	69300	0.162	0.164	0.167
26	SVM	Combination-BOOT	2100	2100	0.151	0.166	0.181
27	SVM	Combination-BOOT	3500	3500	0.154	0.166	0.179
28	NNET	Over-ROSE	69300	69300	0.163	0.167	0.172
29	RF	Combination-SMOTE	3500	3500	0.160	0.167	0.174
30	SVM	Combination-SMOTE	1400	1400	0.164	0.167	0.171
31	GB	Combination-SMOTE	2100	2100	0.160	0.168	0.175
32	RF	Combination-SMOTE	2100	2100	0.161	0.168	0.175
33	SVM	Combination-ROSE	3500	3500	0.153	0.168	0.184
34	GB	Over-BOOT	69300	69300	0.166	0.169	0.171
35	NNET	Combination-ROSE	1400	1400	0.154	0.170	0.186

36	SVM	Combination-ROSE	2100	2100	0.150	0.170	0.189
37	NNET	Combination-ROSE	3500	3500	0.159	0.176	0.193
38	NNET	Combination-ROSE	2100	2100	0.155	0.180	0.206
39	SVM	Under	700	1400	0.177	0.180	0.183
40	NNET	Combination-SMOTE	3500	3500	0.176	0.183	0.189
41	NNET	Under	700	2800	0.183	0.186	0.189
42	NNET	Over-BOOT	69300	69300	0.181	0.186	0.192
43	NNET	Combination-SMOTE	2100	2100	0.178	0.187	0.196
44	GB	Under	7000	6300	0.185	0.188	0.191
45	GB	Over-BOOT	23300	69300	0.187	0.190	0.192
46	RF	Under	7000	6300	0.189	0.192	0.195
47	SVM	Over-ROSE	23300	69300	0.199	0.201	0.203
48	SVM	Combination-SMOTE	3500	3500	0.196	0.202	0.209
49	SVM	Over-BOOT	23300	69300	0.211	0.214	0.216
50	SVM	Combination-SMOTE	2100	2100	0.201	0.214	0.227
51	GB	Over-BOOT	7800	69300	0.213	0.215	0.217
52	NNET	Over-BOOT	23300	69300	0.214	0.218	0.221
53	SVM	Under	700	2800	0.216	0.220	0.223
54	NNET	Under	7000	6300	0.221	0.224	0.227
55	NNET	Over-ROSE	23300	69300	0.224	0.231	0.237
56	NNET	Over-BOOT	7800	69300	0.248	0.252	0.255
57	RF	Over-SMOTE	69300	69300	0.248	0.252	0.256
58	RF	Over-SMOTE	23300	69300	0.251	0.255	0.260
59	SVM	Over-BOOT	7800	69300	0.253	0.256	0.258
60	RF	Over-SMOTE	7800	69300	0.254	0.257	0.260
61	RF	Over-BOOT	7800	69300	0.258	0.261	0.265
62	NNET	Over-SMOTE	69300	69300	0.252	0.264	0.277
63	GB	Over-SMOTE	7800	69300	0.265	0.268	0.271
64	SVM	Over-ROSE	7800	69300	0.264	0.269	0.275
65	SVM	Over-ROSE	69300	69300	0.255	0.270	0.285
66	GB	Over-SMOTE	23300	69300	0.266	0.270	0.275
67	GB	Over-SMOTE	69300	69300	0.267	0.272	0.278
68	RF	Over-BOOT	23300	69300	0.270	0.274	0.278
69	SVM	Over-SMOTE	69300	69300	0.262	0.275	0.287
70	SVM	Under	7000	6300	0.270	0.276	0.283
71	RF	Original	700	69300	0.280	0.282	0.285
72	GB	Original	700	69300	0.280	0.283	0.286
73	RF	Over-BOOT	69300	69300	0.281	0.285	0.289
74	NNET	Over-SMOTE	23300	69300	0.282	0.291	0.300
75	NNET	Over-SMOTE	7800	69300	0.310	0.321	0.333

76	SVM	Over-SMOTE	23300	69300	0.316	0.326	0.337
77	NNET	Over-ROSE	7800	69300	0.317	0.343	0.368
78	RF	Combination-ROSE	3500	3500	0.349	0.352	0.356
79	RF	Over-ROSE	23300	69300	0.349	0.352	0.356
80	RF	Combination-ROSE	1400	1400	0.354	0.354	0.354
81	RF	Over-ROSE	69300	69300	0.354	0.354	0.354
82	NNET	Original	700	69300	0.353	0.357	0.361
83	RF	Combination-ROSE	2100	2100	0.348	0.358	0.368
84	SVM	Over-SMOTE	7800	69300	0.357	0.366	0.374
85	RF	Over-ROSE	7800	69300	0.322	0.370	0.418
86	SVM	Original	700	69300	0.459	0.463	0.467
87	GB	Combination-ROSE	1400	1400	0.962	0.962	0.962
88	GB	Combination-ROSE	2100	2100	0.962	0.962	0.962
89	GB	Combination-ROSE	3500	3500	0.962	0.962	0.962
90	GB	Over-ROSE	7800	69300	0.962	0.962	0.962
91	GB	Over-ROSE	23300	69300	0.962	0.962	0.962
92	GB	Over-ROSE	69300	69300	0.962	0.962	0.962

CDX Results

Rank	Classifier	Sampling Technique	# of attack observations	# of normal observations	Lower 95% CI	Mean ROCCED	Upper 95% CI
1	RF	Under	300	300	0.1021	0.1040	0.1059
2	GB	Combination-BOOT	1500	1500	0.1040	0.1068	0.1097
3	GB	Under	300	300	0.1048	0.1073	0.1099
4	GB	Combination-BOOT	900	900	0.1068	0.1089	0.1111
5	GB	Over-BOOT	141000	141000	0.1092	0.1119	0.1145
6	GB	Combination-BOOT	600	600	0.1084	0.1122	0.1161
7	NNET	Over-BOOT	141000	141000	0.1055	0.1133	0.1211
8	SVM	Over-BOOT	70000	141000	0.1086	0.1151	0.1216
9	RF	Combination-BOOT	600	600	0.1145	0.1188	0.1232
10	SVM	Combination-BOOT	1500	1500	0.1166	0.1191	0.1217
11	RF	Over-BOOT	15300	141000	0.1138	0.1192	0.1246
12	GB	Under	300	600	0.1130	0.1193	0.1256
13	SVM	Combination-BOOT	900	900	0.1176	0.1216	0.1256
14	RF	Under	300	600	0.1187	0.1239	0.1292
15	RF	Combination-BOOT	900	900	0.1212	0.1249	0.1286
16	RF	Over-BOOT			0.1196	0.1255	0.1313
17	NNET	Over-ROSE	70000	141000	0.1209	0.1284	0.1359
18	NNET	Over-BOOT	70000	141000	0.1200	0.1285	0.1370
19	NNET	Over-ROSE	141000	141000	0.1224	0.1300	0.1377
20	SVM	Combination-BOOT	600	600	0.1286	0.1338	0.1390
21	GB	Over-BOOT	70000	141000	0.1293	0.1358	0.1424
22	NNET	Combination-BOOT	1500	1500	0.1362	0.1382	0.1402
23	SVM	Under	300	300	0.1352	0.1393	0.1434
24	NNET	Combination-BOOT	900	900	0.1402	0.1420	0.1438
25	NNET	Combination-BOOT	600	600	0.1415	0.1434	0.1454
26	RF	Combination-BOOT	1500	1500	0.1381	0.1436	0.1492
27	RF	Over-BOOT	70000	141000	0.1397	0.1468	0.1539
28	NNET	Under	300	300	0.1464	0.1494	0.1525
29	SVM	Under	300	600	0.1507	0.1625	0.1744
30	GB	Under	300	900	0.1631	0.1717	0.1803
31	SVM	Combination-ROSE	1500	1500	0.1687	0.1785	0.1883

32	NNET	Under	300	600	0.1667	0.1804	0.1942
33	NNET		35000	141000	0.1589	0.1827	0.2064
34	SVM	Over-BOOT	35000	141000	0.1612	0.1851	0.2091
35	RF	Under	300	900	0.1786	0.1856	0.1926
36	GB	Over-BOOT	35000	141000	0.1792	0.1889	0.1986
37	SVM	Combination-ROSE	600	600	0.1800	0.1931	0.2062
38	SVM	Combination-ROSE	900	900	0.1872	0.1965	0.2058
39	NNET	Over-BOOT	35000	141000	0.1835	0.1972	0.2109
40	NNET	Combination-SMOTE	600	600	0.2016	0.2259	0.2501
41	NNET	Combination-SMOTE	900	900	0.2132	0.2344	0.2556
42	NNET	Combination-SMOTE	1500	1500	0.2169	0.2417	0.2664
43	NNET	Over-SMOTE	141000	141000	0.2162	0.2500	0.2837
44	SVM	Under	300	900	0.2607	0.2659	0.2710
45	NNET	Under	300	900	0.2649	0.2702	0.2755
46	SVM	Over-ROSE	35000	141000	0.2327	0.2711	0.3096
47	SVM	Over-ROSE	15300	141000	0.2476	0.2756	0.3037
48	NNET	Over-ROSE	15300	141000	0.2619	0.2794	0.2970
49	SVM	Combination-SMOTE	600	600	0.2320	0.2824	0.3329
50	SVM	Combination-SMOTE	1500	1500	0.2361	0.2829	0.3298
51	SVM	Over-ROSE	70000	141000	0.2711	0.2858	0.3004
52	GB	Over-BOOT	15300	141000	0.2811	0.2861	0.2911
53	SVM	Combination-SMOTE	900	900	0.2419	0.2887	0.3355
54	SVM	Over-BOOT	15300	141000	0.2834	0.2908	0.2981
55	NNET	Over-BOOT	15300	141000	0.2873	0.2935	0.2997
56	SVM	Under	300	2700	0.2938	0.2997	0.3056
57	GB	Under	300	2700	0.2853	0.3025	0.3197
58	GB	Over-SMOTE	141000	141000	0.2642	0.3027	0.3413
59	NNET	Under	300	2700	0.3014	0.3102	0.3190
60	NNET	Over-SMOTE	70000	141000	0.3006	0.3193	0.3380
61	RF	Under	300	2700	0.3208	0.3327	0.3447
62	GB	Combination-SMOTE	600	600	0.3077	0.3556	0.4036
63	NNET	Combination-ROSE	1500	1500	0.3230	0.3563	0.3896
64	GB	Combination-SMOTE	900	900	0.3174	0.3638	0.4102
65	NNET	Combination-ROSE	900	900	0.3240	0.3639	0.4039
66	GB	Combination-SMOTE	1500	1500	0.3318	0.3765	0.4213
67	RF	Combination-SMOTE	600	600	0.3397	0.3818	0.4239
68	RF	Over-BOOT	141000	141000	0.3826	0.3927	0.4029

69	GB	Over-SMOTE	70000	141000	0.3491	0.3970	0.4449
70	RF	Combination-SMOTE	600	600	0.3585	0.3980	0.4375
71	NNET	Over-ROSE	600	600	0.3736	0.4110	0.4484
72	NNET	Over-SMOTE	35000	141000	0.3570	0.4160	0.4750
73	RF	Combination-SMOTE	1500	1500	0.3861	0.4241	0.4622
74	GB	Over-SMOTE	35000	141000	0.4861	0.5546	0.6231
75	RF	Over-SMOTE	70000	141000	0.6213	0.6500	0.6786
76	SVM	Over-SMOTE	15300	141000	0.6080	0.6735	0.7389
77	NNET	Over-SMOTE	15300	141000	0.6091	0.6760	0.7430
78	RF	Over-SMOTE	35000	141000	0.6671	0.6870	0.7069
79	SVM	Over-SMOTE	35000	141000	0.6041	0.6872	0.7704
80	SVM	Over-SMOTE	70000	141000	0.5640	0.6892	0.8144
81	RF	Over-SMOTE	15300	141000	0.7137	0.7263	0.7388
82	RF	Combination-ROSE	1500	1500	0.7227	0.7343	0.7460
83	RF	Over-ROSE	35000	141000	0.7218	0.7360	0.7501
84	RF	Over-ROSE	70000	141000	0.7218	0.7360	0.7501
85	RF	Over-ROSE	141000	141000	0.7229	0.7375	0.7521
86	RF	Combination-ROSE	900	900	0.7288	0.7406	0.7525
87	GB	Over-SMOTE	15300	141000	0.7103	0.7413	0.7723
88	RF	Combination-ROSE	600	600	0.7302	0.7416	0.7530
89	RF	Over-ROSE	15300	141000	0.7291	0.7442	0.7592
90	RF	None	300	141000	0.7733	0.7750	0.7766
91	SVM	None	300	141000	0.7784	0.7785	0.7786
92	NNET	None	300	141000	0.7772	0.7799	0.7826
93	GB	None	300	141000	0.7823	0.7832	0.7841
94	GB	Combination-ROSE	600	600	0.8371	0.8936	0.9501
95	GB	Over-ROSE	70000	141000	0.8395	0.9071	0.9746
96	GB	Combination-ROSE	900	900	0.8395	0.9071	0.9746
97	GB	Over-ROSE	15300	141000	0.8535	0.9163	0.9792
98	GB	Combination-ROSE	1500	1500	0.8418	0.9163	0.9908
99	GB	Over-ROSE	141000	141000	0.8458	0.9283	1.0107
100	GB	Over-ROSE	35000	141000	0.9997	0.9997	0.9997

References

[1] L. Whitney, “U.S. Cyber Command prepped to launch,” *CNET*, 2010. [Online]. Available: http://www.cnet.com/news/u-s-cyber-command-prepped-to-launch/. [Accessed: 24-Oct-2015].

[2] S. Singh and S. Silakari, “A Survey of Cyber Attack Detection Systems,” *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 9, no. 5, 2009.

[3] R. Sommer and V. Paxson, “Outside the Closed World: On Using Machine Learning For Network Intrusion Detection,” in *Security and Privacy (SP), 2010 IEEE Symposium on*, 2010, pp. 305–316.

[4] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, “Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods,” *Syst. Man, Cybern. Part C Appl. Rev. IEEE Trans.*, vol. 40, no. 5, 2009.

[5] “Cybersecurity, Data Science and Machine Learning: Is All Data Equal?” [Online]. Available: http://www.computerworld.com/article/2908507/cybersecurity-data-science-and-machine-learning-is-all-data-equal.html. [Accessed: 24-Oct-2015].

[6] “The 2014 Ziegel Award — Applied Predictive Modeling.” [Online]. Available: http://appliedpredictivemodeling.com/blog/2015/8/12/the-2014-ziegel-award. [Accessed: 10-Sep-2015].

[7] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. New York, NY: Springer New York, 2013.

[8] J. P. Anderson Co, “Computer Security Threat Monitoring and Surveillance,” Fort Washington, PA, 1980.

[9] D. E. Denning, “An Intrusion-Detection Model,” *IEEE Trans. Softw. Eng.*, vol. 13, no. 2, pp. 222–232, 1987.

[10] “UCI Machine Learning Repository: KDD Cup 1999 Data Data Set.” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data. [Accessed: 19-Dec-2015].

[11] “1998 DARPA Intrusion Detection Evaluation Data Set,” 2015. [Online]. Available: http://www.ll.mit.edu/ideval/data/1998data.html. [Accessed: 01-Jan-2015].

[12] C. Jirapummin, N. Wattanapongsakorn, and P. Kanthamanon, “Hybrid Neural Networks for Intrusion Detection System,” *Proc. of ITC–CSCC*, 2002.

[13] D. Zhang, “Hybrid Neural Network and C4.5 for misuse detection," *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, pp. 2–5, 2003.

[14] M. Moradi and M. Zulkerine, “A Neural Network Based System for Intrusion Detection and Classification of Attacks,” *Proc. 2004 IEEE Int. Conf. Adv. Intell. Syst. Appl.*, 2004.

[15] X. Li, S. Wang, and Z. Y. Dong, Eds., *Advanced Data Mining and Applications*, vol. 3584. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[16] Z. Pan, H. Lian, G. Hu, and G. Ni, “An Integrated Model of Intrusion Detection Based on

Neural Network and Expert System," in *Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on*, p. 672.

[17] S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas, "Modeling intrusion detection system using hybrid intelligent systems," *J. Netw. Comput. Appl.*,vol. 30, no. 1, 2007.

[18] J. Yao, S. Zhao, and L. Fan, "An Enhanced Support Vector Machine Model for Intrusion Detection," *Rough Sets Knowl. Technol.*, pp. 538–543, 2006.

[19] O. Depren, M. Topallar, E. Anarim, and M. K. Ciliz, "An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks," *Expert Syst. Appl.*, vol. 29, no. 4, pp. 713–722, 2005.

[20] M. Zulkernine and J. Zhang, "Network intrusion detection using random forests," *Proc. Third Annu. Conf. Privacy, Secur. Trust*, pp. 53–61, 2005.

[21] J. T. Yao, S. L. Zhao, and L. V Saxton, "A study on fuzzy intrusion detection," *Proc. SPIE*, vol. 5812, pp. 23–30.

[22] Y. Bouzida and F. Cuppens, "Neural networks vs. decision trees for intrusion detection," in *Communications, 2006. ICC '06. IEEE International Conference on*, 2006, pp. 2394–2400.

[23] M. Panda and M. R. Patra, "Network Intrusion Detection Using Naïve Bayes," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 7, no. 12, 2007.

[24] V. Katos, "Network intrusion detection: Evaluating cluster, discriminant, and logit analysis," *Inf. Sci. (Ny).*, vol. 177, no. 15, pp. 3060–3073, 2007.

[25] W. Hu and W. Hu, "Network-based intrusion detection using adaboost algorithm," *Proc. - 2005 IEEE/WIC/ACM Int. Web Intell. WI 2005*, vol. 2005, pp. 712–717, 2005.

[26] H. A. Nguyen and D. Choi, "Application of Data Mining to Network Intrusion Detection: Classifier Selection Model," *LNCS*, vol. 5297, pp. 399–408, 2008.

[27] C. S. Leung, M. Lee, J. H. Chan, M. Panda, and M. R. Patra, "Semi-Naïve Bayesian Method for Network Intrusion Detection System," *LNCS*, vol. 5863, pp. 614–621, 2009.

[28] J. McHugh, "The 1998 Lincoln Laboratory IDS Evaluation," pp. 145–161, Oct. 2000.

[29] M. V Mahoney and P. K. Chan, "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection," *Proc. Sixth Int. Symp. Recent Adv. Intrusion Detect.*, vol. 2820, no. Ll, pp. 220–237, 2003.

[30] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009.

[31] S. Brugger and J. Chow, "An assessment of the DARPA IDS Evaluation Dataset using Snort," *UCDAVIS Dep. Comput. Sci.*, pp. 1–19, 2007.

[32] L. Cai, J. Chen, Y. Ke, T. Chen, and Z. Li, "A new data normalization method for unsupervised anomaly intrusion detection," *J. Zhejiang Univ. Sci. C*, vol. 11, no. 10, pp. 778–784, 2010.

[33] S. Revathi, "A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning

Techniques for Intrusion Detection," *Int. J. Eng. Res. Technol.*, vol. 2, no. 12, pp. 1848–1853, 2013.

[34] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers, "Malware type recognition and cyber situational awareness," in *Proceedings - SocialCom 2010: 2nd IEEE International Conference on Social Computing, PASSAT 2010: 2nd IEEE International Conference on Privacy, Security, Risk and Trust*, 2010.

[35] K. L. Moore, T. J. Bihl, K. W. Bauer, and T. E. Dube, "Cyber Data Feature-Extraction and Artificial Neural Network Based Feature-Selection for Classifying Cyber-Traffic Threats," *J. Def. Model. Simul.*, 2016.

[36] "Deep Instinct." [Online]. Available: http://www.deepinstinct.com/#/how-we-do-it. [Accessed: 30-Jan-2016].

[37] N. Gao, L. Gao, Q. Gao, and H. Wang, "An Intrusion Detection Model Based on Deep Belief Networks," in *Second International Conference onAdvanced Cloud and Big Data (CBD)*, 2014.

[38] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning - ICML '07*, 2007, pp. 791–798.

[39] Y. Li, R. Ma, and R. Jiao, "A Hybrid Malicious Code Detection Method based on Deep Learning," *Int. J. Secur. Its Appl.*, vol. 9, no. 5, pp. 205–216, 2015.

[40] W. Choe, "Detection of rare events and rule extraction by neural networks and decision trees," *Theses and Dissertations Available from ProQuest*. pp. 1 – 155, 1999.

[41] K. Veropoulos, C. Campbell, N. Cristianini, and Others, "Controlling the sensitivity of support vector machines," in *Proceedings of the international joint conference on artificial intelligence*, 1999.

[42] C.-F. Lin and S.D. Wang, "Fuzzy support vector machines.," *IEEE Trans. Neural Netw.*, vol.13, no. 2, pp. 464-471, 2002.

[43] R. Batuwita and V. Palade, "FSVM-CIL: Fuzzy support vector machines for class imbalance learning," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 3, 2010.

[44] B. Lakshmanan, A. J. Priscilla, S. Ponni, and V. Sankari, "Evaluation of imbalanced datasets using fuzzy support vector machine-class imbalance learning (FSVM-CIL)," in *International Conference on Recent Trends in Information Technology, ICRTIT 2011*, 2011.

[45] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[46] L. Torgo, *Data Mining with R, learning with case studies*. Chapman and Hall, 2010.

[47] N. V Chawla, A. Lazarevi, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: Improving Prediction of the Minority Class in Boosting," in *Knowledge Discovery in Databases: PKDD 2003*, 2003, pp. 107–119.

[48] S. Hu, Y. Liang, L. Ma, and Y. He, "MSMOTE: Improving Classification Performance when Training Data is imbalanced," Second Inernational Workshop on Comp. Sci. and

Eng.,2009.

[49] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, 2010.

[50] R. E. Schapire, "A brief introduction to boosting," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2, no. 5, pp. 1401–1406, 1999.

[51] G. Menardi and N. Torelli, "Training and assessing classification rules with unbalanced data," *Work. Pap. Ser.*, no. 2, 2010.

[52] M. Rosenblatt, "Remarks On Some Nonparametric Estimates Of A Density Function," *Ann. Math. Stat.*, no. 27, pp. 832–837, 1956.

[53] M. C. Jones, J. S. Marron, and S. J. Sheather, "A Brief Survey of Bandwidth Selection for Density Estimation," *J. Am. Stat. Assoc.*, vol. 91, no. 433, pp. 401–407, 1996.

[54] S. J. Sheather and M. C. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *J. R. Stat. Soc. Ser. B*, vol. 53, no. 3, pp. 683–690, 1991.

[55] N. Lunardon, G. Menardi, and N. Torelli, "ROSE : A Package for Binary Imbalanced Learning," *R Journal.*, 2014.

[56] "UCI Machine Learning Repository." [Online]. Available: http://archive.ics.uci.edu/ml/index.html. [Accessed: 11-Sep-2015].

[57] G. M. Weiss and F. Provost, "The Effect of Class Distribution on Classifier Learning: An Empirical Study, Technical Report ML-TR-44," 2001.

[58] S. L. Salzberg, "C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993," *Mach. Learn.*, vol. 16, no. 3, pp. 235–240, Sep. 1994.

[59] C. Drummond and R. C. Holte, "C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," *Work. Learn. from Imbalanced Datasets II*, pp. 1–8, 2003.

[60] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data."

[61] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An Empirical Study of Learning from Imbalanced Data Using Random Forest," in *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*, 2007, vol. 2, pp. 310–317.

[62] "Weka Data Mining with Open Source Machine Learning Software in Java." [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed: 10-Sep-2015].

[63] J. Burez and D. Van Den Poel, "Handling class imbalance in customer churn prediction," *Expert Syst. Appl.*, vol. 36, pp. 4626–4636.

[64] J. C. Deville and Y. Tillé, "Efficient balanced sampling: The cube method," *Biometrika*, vol. 91, no. 4, pp. 893–912, 2004.

[65] P. Jeatrakul, K. W. Wong, C. C. Fung, P. Jeatrakul, K. W. Wong, and C. C. Fung, "Classification of Imbalanced Data by Combining the Complementary Neural Network and SMOTE Algorithm," vol. 6444, no. 2, pp. 152–159, 2010.

[66] N. Moustafa and J. Slay, "UNSW-NB15: A Comprehensive Data set for Network

Intrusion Detection systems (UNSW-NB15 Network Data Set)," in *Military Communications and Information Systems Conference (MilCIS)*, 2015.

[67] "Ixia Network|Security|Application Performance." [Online]. Available: http://www.ixiacom.com/. [Accessed: 24-Dec-2015].

[68] "CVE - Common Vulnerabilities and Exposures (CVE) ." [Online]. Available: https://cve.mitre.org/. [Accessed: 24-Dec-2015].

[69] A. A. Aly and C. It, "Tracking and Tracing Spoofed IP Packets to Their Sources," *The Sixth Annual U.A.E. Research Conference*, pp. 100–106.

[70] K. L. Moore, "Salient Feature Selection Using Feed-Forward Neural Networks and Signal-to-Noise Ratios with a Focus Toward Network Threat Detection and Classification," MS Thesis, AFIT, 2014.

[71] "Cyber Research Center - DataSets." [Online]. Available: http://www.usma.edu/crc/sitepages/datasets.aspx. [Accessed: 24-Dec-2015].

[72] C. Gershenson, "Artificial neural networks for beginners," *arXiv Prepr. cs/0308031*, 2003.

[73] R. A. FISHER, "The Use Of Multiple Measurements In Taxonomic Problems," *Ann. Eugen.*, vol. 7, no. 2, pp. 179–188, Sep. 1936.

[74] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S.*, 4th ed. New York, NY: Springer New York, 2002.

[75] J. McLelland and D. Rumelhart, "Parallel distributed processing," in *Explorations in the microstructure of cognition 2*, 1986.

[76] W. A. Young, T. Bihl, and G. R. Weckman, "Artificial Neural Networks for Business Analytics," *Encycl. Bus. Anal. Optim.*, vol. 1, pp. 193–208, 2014.

[77] A. Krogh and J. Hertz, "A Simple Weight Decay Can Improve Generalization," *Adv. Neural Inf. Process.*, 1992.

[78] D. Sarkar, "Methods to speed up error back-propagation learning algorithm," *ACM Comput. Surv.*, vol. 27, no. 4, pp. 519–544, Dec. 1995.

[79] R. Battiti, "First-and second-order methods for learning: between steepest descent and Newton's method," *Neural Comput.*, 1992.

[80] C. G. Broyden, J. E. Dennis, and J. J. Moré, "On the Local and Superlinear Convergence of Quasi-Newton Methods," *IMA J. Appl. Math.*, vol. 12, no. 3, pp. 223–245, Dec. 1973.

[81] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[82] L. Breiman, *CART: Classification and regression trees.* Belmont, CA: Wadsworth, 1983.

[83] J. Mingers, "An Empirical Comparison of Pruning Methods for Decision Tree Induction," *Mach. Learn.*, vol. 4, no. 2, pp. 227–243.

[84] T. Oshiro, P. Perez, and J. Baranauskas, "How Many Trees in a Random Forest?," in *Machine Learning and Data Mining in Pattern Recognition*, vol. 7376, P. Perner, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 154–168.

[85] A. Liaw and M. Wiener, "Classification and Regression by randomForest," *R News*, vol.

3, no. 2, pp. 18–22, 2002.

[86] V. Vapnik, “Pattern recognition using generalized portrait method,” *Autom. Remote Control*, vol. 24, pp. 774–780, 1963.

[87] V. Vapknik, *Statistical Learning Theory Vol 1*. New York, NY: Wiley, 1998.

[88] K. P. Bennett and C. Campbell, “Support vector machines: Hype or Hallelujah?,” *ACM SIGKDD Explor. Newsl.*, vol. 2, no. 2, pp. 1–13, 2000.

[89] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Berlin: Springer, pp-417-423, 2001.

[90] T. Hastie, H. Edu, S. Rosset, R. Tibshirani, J. Zhu, and J. Edu, “The Entire Regularization Path for the Support Vector Machine,” *J. Mach. Learn. Res.*, vol. 5, pp. 1391–1415, 2004.

[91] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, “kernlab - An S4 Package for Kernel Methods in R,” *J. Stat. Softw.*, vol. 11, no. 9, pp. 1–20, Nov. 2004.

[92] J. Platt, “Fast training of support vector machines using sequential minimal optimization,” *Adv. Kernel Methods - Support Vector Learn.*, 1999.

[93] L. Zhuang and H. Dai, “Parameter optimization of kernel-based one-class classifier on imbalance learning,” *J. Comput.*, 2006.

[94] O. Chapelle and Y. Chang, “Yahoo! Learning to Rank Challenge Overview.,” *Yahoo! Learn. to Rank Chall.*, 2011.

[95] M. Kearns, “Thoughts on Hypothesis Boosting.,” *Unpublished Manuscript,* 1988. Available: https://www.cis.upenn.edu/~mkearns/papers/boostnote.pdf.

[96] Y. Freund and R. Schapire, “Experiments with a new boosting algorithm,” *ICML*, 1996.

[97] L. Breiman, “Arcing classifier (with discussion and a rejoinder by the author),” *Ann. Stat.*, vol. 26, no. 3, pp. 801–849, Jun. 1998.

[98] T. Chen and M. Benesty, “CRAN - Package xgboost.” 2015. Available: https://cran.r-project.org/web/packages/xgboost/index.html.

[99] Y. Freund, R. Schapire, and N. Abe, “A short introduction to boosting,” *Journal-Japanese Soc. Artif.*, 1999.

[100] P. M. Long and R. A. Servedio, “Random classification noise defeats all convex potential boosters,” *Mach. Learn.*, vol. 78, no. 3, pp. 287–304, 2009.

[101] T. Fawcett, “An introduction to ROC analysis,” *Pattern Recognit. Lett 27.*, pp. 861-874, 2006.

[102] D. M. W. Powers and Ailab, “Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation,” *J. Mach. Learn. Technol. ISSN*, vol. 2, no. 1, pp. 2229–3981, 2011.

[103] V. Labatut and H. Cherifi, “Accuracy Measures for the Comparison of Classifiers,” Jul. 2012.

[104] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.

[105] W. J. Youden, “Index for rating diagnostic tests,” *Cancer*, vol. 3, no. 1, pp. 32–35, 1950.

[106] A. Pérez-Garrido, A. M. Helguera, F. Borges, M. N. D. S. Cordeiro, V. Rivero, and A. G. Escudero, “Two new parameters based on distances in a receiver operating characteristic chart for the selection of classification models.,” *J. Chem. Inf. Model.*, vol. 51, no. 10, pp. 2746–59, Oct. 2011.

[107] S. Arlot and A. Celisse, “A survey of cross-validation procedures for model selection,” *Stat. Surv.*, vol. 4, pp. 40–79, 2010.

[108] A. M. Molinaro, R. Simon, and R. M. Pfeiffer, “Prediction error estimation: a comparison of resampling methods,” *Bioinformatics*, vol. 21, no. 15, pp. 3301–3307, May 2005.

[109] B. Efron and R. Tibshirani, “Improvements on Cross-Validation: The 632+ Bootstrap Method,” *J. Am. Stat. Assoc.*, vol. 92, no. 438, pp. 548–560, Jun. 1997.

[110] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, “Do we need hundreds of classifiers to solve real world classification problems?,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, Jan. 2014.

[111] D. J. Hand, “Classifier Technology and the Illusion of Progress,” *Stat. Sci.*, vol. 21, no. 1, pp. 1–14, Feb. 2006.

[112] T. Armerding, “Machine learning: Cybersecurity dream-come-true or pipe dream? | CSO Online,” *CSO*, 2015. [Online]. Available: http://www.csoonline.com/article/3015670/security/machine-learning-cybersecurity-dream-come-true-or-pipe-dream.html. [Accessed: 09-Jan-2016].

[113] P. Domingos, “A few useful things to know about machine learning,” *Communications of the ACM.*, vol. 55, no. 10, pp. 78-87, 2012.

[114] K. W. Bauer, S. G. Alsing, and K. A. Greene, “Feature screening using signal-to-noise ratios,” *Neurocomputing*, vol. 31, no. 1, pp. 29-44, 2000.

[115] K. L. Priddy, S. K. Rogers, D. W. Ruck, G. L. Tarr, and M. Kabrisky, “Bayesian selection of important features for feedforward neural networks,” *Neurocomputing*, vol.5, no. 2, pp. 91-103, 1993.

[116] M. Gevrey, I. Dimopoulos, and S. Lek, “Review and comparison of methods to study the contribution of variables in artificial neural network models,” in *Ecological Modelling*, vol 160, no. 3, pp. 249-264, 2003.

[117] A. T. C. Goh, “Back-propagation neural networks for modeling complex systems,” *Artif. Intell. Eng.*, vol. 9, no. 3, pp. 143-151, 1995.

[118] G. H. John, R. Kohavi, and K. Ppeger, “Irrelevant Features and the Subset Selection Problem,” *Machine learning: proceedings of the eleventh international conference*, pp. 121–129, 1994.

[119] X. Zhang, X. Lu, Q. Shi, and L. Harris, “Recursive SVM feature selection and sample classification for mass-spectrometry and microarray data,” *BMC Bioinformatics*, vol. 7, no. 1, 2006.

[120] J. Yang and V. Honovar, “Feature Subset Selection Using a Genetic Algorithm,” in *Feature Extraction and Selection*, Springer, 1998, pp. 117–136.

[121] J. Debuse and V. Rayward-Smith, “Feature Subset Selection within a Simulated Annealing Data Mining Algorithm,” *J. Intell. Inf. Syst.*, vol. 9, no. 1, pp. 57–81, 1997.

[122] Y. Saeys, I. Inza, and P. Larranaga, “A review of feature selection techniques in bioinformatics,” *Bioinformatics*, vol. 23, no. 19, p. 2507–2517, 2007.

[123] A. Pinto, “Secure because Math:A deep-dive on Machine Learning-based Monitoring,” in *Black Hat 2014*, 2014.

[124] S. Reid, “A review of heterogeneous ensemble methods,” *Dep. Comput. Sci. Univ. Color.*, 2007.

[125] M. P. Perrone and L. N. Cooper, “When Networks Disagree: Ensemble Methods for Hybrid Neural Networks,” Oct. 1992.

[126] D. Wolpert, “Stacked generalization,” *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[127] J. Hoeting and D. Madigan, “Bayesian model averaging,” in *Proceedings of the AAAI Workshop on Integrating Multiple Learned Models*, 1998.

[128] D. Wolpert, “The supervised learning no-free-lunch theorems,” *Soft Comput. Ind.*, 2002.

[129] “R: The R Project for Statistical Computing.” 2015. [Online]. Available: https://www.r-project.org. [Accessed 29-Nov-2015]

[130] S. Cass, “The 2015 Top Ten Programming Languages - IEEE Spectrum,” *IEEE Spectrum*, 2015. [Online]. Available: http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages. [Accessed: 01-Jan-2016].

[131] “Download R-3.2.1 for Windows. The R-project for statistical computing.” [Online]. Available: https://cran.r-project.org/bin/windows/base/old/3.2.1/. [Accessed: 01-Jan-2016].

[132] M. Kuhn, “Building Predictive Models in R Using the caret Package,” *J. Stat. Softw.*, vol. 28, no. 5, pp. 1–26, Nov. 2008.

[133] H. Wickham and R. Francois, “A Grammar of Data Manipulation.” 01-Sep-2015. Available: https://cran.r-project.org/web/packages/dplyr/index.html.

[134] H. Wickham, *An Implementation of the Grammar of Graphics*. New York: Springer, 2009.

SF 298

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704–0188

1. REPORT DATE (DD–MM–YYYY)	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24-03-2016	Master's Thesis	Sep 2014 - Mar 2016

4. TITLE AND SUBTITLE
Methods to Address Extreme Class Imbalance in Machine Learning Based Network Intrusion Detection Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Walter, Russell W., MAJ, USA

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENS-MS-16-M-131

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
H.Q. US Air Force
Analysis, Assessments, and Lessons Learned
(AF/A9)
1570 Air Force phone: 571-256-2147
Pentagon, Washington D.C. 20330-1570
ATTN: Major Todd J. Paciencia Todd.j.paciencia.mil@mail.mil

10. SPONSOR/MONITOR'S ACRONYM(S)
AF/A9FC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Despite the considerable academic interest in using machine learning methods to detect cyber-attacks and malicious network traffic, there is little evidence that modern organizations employ such systems. Due to the targeted nature of attacks and cybercriminals' constantly changing behavior, valid observations of attack traffic suitable for training a classifier are extremely rare. Rare positive cases combined with the fact that the overwhelming majority of network traffic is benign create an extreme class imbalance problem. Using publically available datasets, this research examines the class imbalance problem by using small samples of the attack observations to create multiple training sets that reflect a realistic class imbalance. A variety of techniques to alleviate the imbalance are examined including under sampling the majority class and three techniques to over sample the minority attack observations by creating new synthetic observations. We test these methods on four of the most popular machine learning classifiers. We examine two single model classifiers, artificial neural networks and support vector machines, and two ensemble methods, gradient boosting and random forests. We find that under sampling generally outperforms oversampling techniques and that the ensemble methods both outperform single models. We show that the apparent superiority of the ensemble methods may be illusory due to the "laboratory conditions" of using well-crafted public datasets. By introducing an element of noise into the training data, we show that neural networks' robustness to noise make it the preferred approach in real world settings where the more sophisticated ensemble methods fail. We also present a technique where neural networks are used to select features from the noisy dataset that improve the performance of random forests and gradient boosting allowing for the creation of an improved ensemble classifier.

15. SUBJECT TERMS
Machine Learning, Network Intrusion Detection Systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kenneth W. Bauer Jr., AFIT/ENS
U	U	U	UU	118	19b. TELEPHONE NUMBER (Include Area Code) (937) 255-3636, ext 4328 (kenneth.bauer@afit.edu)

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18