Improvements on Self-Organizing Feature Maps for User-to-Root and Remote-to-Local Network Intrusion Detection on the 1999 KDD Cup Dataset

Ryan Wilson, Charlie Obimbo
School of Computer Science
University of Guelph
Guelph, Ontario, Canada

Abstract

The problem of network intrusion detection is one that is ever-changing, ever-evolving, and is always in need of improvement. Since the method of attack is constantly changing, intrusion detection systems must also be constantly improved in order to compensate for the threat of new attacks. This paper is written to outline the improvements made upon the original paper published by Wilson et al. in which a self-organizing feature map-based intrusion detection system was trained using the 1999 KDD Cup competition training dataset and was used to successfully classify 63% of all user-to-root attacks within the 1999 KDD Cup competition testing dataset. This result shows an improvement of over five times the number of successfully detected user-to-root attacks by the winner of the 1999 KDD Cup competition, submitted by Bernard Pfahringer.

1. Introduction

Intrusion detection systems play a pivotal role in preventing malicious attacks on the integrity, security, and reliability of a computer network. The problem of successful network intrusion detection is a complex one. There are frequent network intrusions and they can have crippling consequences. They are often centered on politics, with one side targeting opposition systems in an attempt to disrupt proceedings. Such an occurrence took place in Russia in 2009, where several government opposition websites were deliberately targeted by Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks [5]. Network attacks do not only take place in hostile environments like Eastern Europe, but also in North America. In 2008, CNN was victimized by, not one but, two DDoS attacks in quick succession [6]. As well, in 2008, the internet's Top Level Domain (TLD) systems were systematically attacked by a DDoS, resulting in the crashing of the entire internet for several hours. These attacks are eventually thwarted, with the quantifiable cost of financial loss, as well as the immeasurable cost resulting from system downtime. With attacks changing weekly, it is imperative for detection systems to become more generalized to accommodate for the ever-changing parameters of what constitutes an attack. The need for such a system sparked the topic of the 1999 Knowledge Discovery and Data Mining (KDD) Cup competition.

The original paper published by Wilson et al. [1] analyzed the results from the KDD Cup competition, discovering that each system used a supervised learning algorithm. These systems produced mixed results for User-to-Root and Remote-to-Local attacks. They sought to eliminate the programmer's "understanding" of the problem, and allow the system to learn without the encumbrance of a programmer's self-conceived notions of how the system should learn.

Self-Organizing Feature Maps (SOFM) were used, with the aim of allowing the system to analyze the raw data without human interference to create a solution set which would separate the data into the various intrusion types. Those intrusion types are Denial of Service, Probing, User-to-Root, and Remote-to-Local.

The research conducted by Wilson et al. [1] showed solid results for successfully detecting intrusions for all but one attack category. Their system showed detection rates of 93%, 66%, 63%, 0% and 96% for denial-of-service, probe, user-to-root, remote-to-local attacks, and normal traffic respectively.

The glaring issue with these results was the absence of successful detection for remote-to-local attacks. This paper aimed to resolve this disparity by implementing two enhancements upon the initial method. First, in the training phase of the methodology the dimension of data, used to categorize similar data and ultimately determine successful intrusion detection, was reduced. Secondly, in the post-processing phase of the methodology, a biased pruning algorithm was implemented in order to increase the percentage of successful intrusion detection within remote-to-local attacks. Each of these approaches will be explained, in conjunction with those found in [1], in Chapter 3.

2. Literature Review

In this chapter, an overview of network intrusion detection and the 1999 KDD Cup competition will be provided. As well, the functionality of a self-organizing feature map will be explained in detail.

Network intrusion detection is the process of detecting network traffic in order to passively, or actively, prevent an attack on a system. Detection is performed in one of two ways: anomaly detection and misuse detection.

2.1. Anomaly Detection

Anomaly detection works by establishing a baseline for normal network behaviour. A system trained for anomaly detection analyzes known "normal" network behaviour to establish this baseline and, after successful training, is used to detect any network behaviour outside of the established baseline. This method is quite successful in detecting any type of "non-normal" behaviour, but has trouble defining specific intrusion types, and has a very high false-positive rate. The high false-positive rate is due to the system's lack of experience with the overall set of normal network behaviour. In essence, the system flags any unknown traffic as an intrusion.

2.2. Misuse Detection

Misuse detection works on the opposite principle to anomaly detection. Misuse detection works by analyzing known network intrusion patterns to establish a signature for such an intrusion to be used in analyzing future network traffic. This has the advantage of having a very low false-positive rate compared to anomaly detection, but has the limitation of not being able to detect new intrusions very well. Since network intrusions change rapidly to counteract existing systems, misuse detection systems fall short when detecting these new attacks and are, traditionally, not ideal. As mentioned earlier, the winner of the 1999 KDD Cup competition detected 13% and 8% of U2R and R2L attacks respectively [2, 9], within the testing dataset used.

2.3. 1999 KDD Cup competition

The 1999 Knowledge Discovery and Data mining (KDD) Cup competition was designed to tackle the issue of ever-changing network intrusions. This competition is an annual event organized by the Association for Computing Machinery (ACM) Special Interest Group on Knowledge Discovery and Data Mining, the self-proclaimed leading professional organization of data miners [7]. The task of the 1999 competition was to ”build a network intrusion detector, a predictive model capable of distinguishing between ’bad’ connections, called intrusions or attacks, and ’good’ normal connections” [8].

This task was undertaken by various groups from around the world, each providing a unique perspective on the problem, as well as different systems for solving it. The general results were promising, but with an obvious shortfall with the detection rates for user-to-root (U2R) and remote-to local (R2L) network intrusions. The winning system posted correct detection rates of 13% and 8% for U2R and R2L attacks respectively [2, 9]. This seemed puzzling as a successful system should be able to successfully detect roughly the same percentage of attacks throughout each type. Further analysis revealed a possible explanation that suggested that the dataset itself was flawed, and that the training and testing datasets showed a dissimilar target hypothesis; implying that one could not be trained to test for the other [10].

The KDD dataset is a set of data consisting of two parts, training data and testing data. The training data is a set consisting of approximately five million rows of data, with the testing set consisting of approximately five hundred thousand rows. These rows are captured data packets, converted into numerical values. Each row in the dataset consists of 4 initial classifiers, 37 float values, as well as a type value. The type value is used to label the row to illustrate if the data is of normal type, or if it is an intrusion type.

2.4. The Self-Organizing Feature Map

Self-organizing feature maps (SOFMs) [11, 12, 13] are the brainchild of Finnish academic Dr. Teuvo Kohonen. Dr. Kohonen is a prominent researcher and is currently a professor emeritus of the Academy of Finland. Kohonen has made tremendous contributions to the field of artificial intelligence; his most notable being that of the SOFM, or Kohonen Map, named in his honor.

Before this paper delves into the inner workings of a SOFM, the general concept must first be explained. As mentioned earlier, a SOFM is an unsupervised learning technique. This approach differs from a traditional artificial neural network – from which SOFMs are derived– as an artificial neural network is supervised. Supervised learning implies that the network being trained is trained in a particular direction, and the success of the network is determined by how well it learns in that direction. SOFMs differ in this respect as they are not given a formal direction in which to learn. Instead, a SOFM is presented with a set of data and determines on its own how best to learn that data. This technique has both its advantages and drawbacks, which will be discussed in the following sections.

Due to the unsupervised learning of the SOFM, they are generally better suited towards classification problems rather than forecasting problems, to which artificial neural networks are vastly superior. In general, considering SOFMs as a clustering methodology goes a long way towards understanding how they function.

It is with the realization that SOFMs are best suited for classification that their design is based. SOFMs are implemented as, usually, a two dimensional grid, or map, of nodes, each with its own candidate vector of data. As one can see in Figure 1, the map of nodes is arranged as a grid.

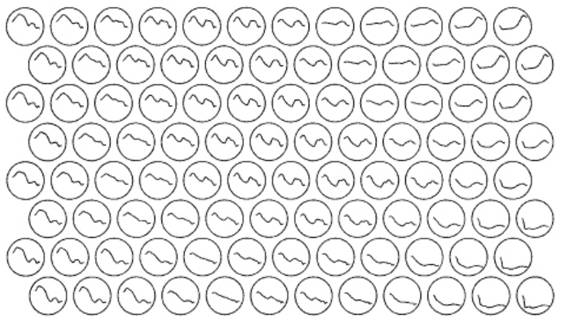

Figure 1. A SOFM as a Grid

When creating a SOFM, the first consideration one must make is how large a map is required. This consideration is generally dependent on the number of different classifications, or groups, that are necessary. For instance, if there were only two possible groupings, then having a map of 3000 nodes is not logical. Contrastingly, if there are many possible groupings, a sufficiently large map will be required. The size of a map for any given problem can be determined empirically, through several intelligent estimations and simulations.

Secondly, a time interval is required which tells the system when to stop training. Much like that of the upper bound in a feed-forward network, this interval, represented as (λ), imposes a hard stop to training. This bound is checked after each row during the SOFM's training.

Thirdly, knowing the dimensionality of the data is required. The SOFM must know how to reduce the dimensionality of the data in order to represent it in the two dimensional map. In order to achieve this, knowing the dimensionality of the input vector is a must. Again, this value is different for each problem, and is easily deciphered.

Fourthly, and this is the most vital piece of information in the initial setup, a neighbourhood function must be selected. Neighbourhood functions allow the training of the surrounding, or neighbour, nodes of the map. The neighbourhood function is represented as ($\theta(t)$) A good example of a commonly used neighbourhood function is that of the Gaussian curve. The selection of this curve as a neighbourhood function implies that the designer of the system wants a few of the winner's neighbours to be trained along with the winner, and then have the influence of the training gradually reduce as the distance from the winner increases. Modifying this neighbourhood function can change the learning of the map. For instance, if the peak of the curve has its width increased, it suggests that the designer would like a larger radius around the winning node to be trained more intensely than a by narrow peak. The candidate node is treated as being at the peak of the curve.

Figure 2 illustrates the gradual descent of learning on nodes around the winning, or candidate, node. It can be observed that the learning around the candidate node is circular. That implies that a distance is calculated from the candidate node 360 degrees around it, to encompass every neighbour node within the distance threshold. Figure 1.2 also shows that as the distance increases, the effect of the learning algorithm decreases (denoted by the ever-thinner circles). This effect is known as gradient descent and is important because one would hope that only the closest nodes to the candidate node are trained as effectively as possible.

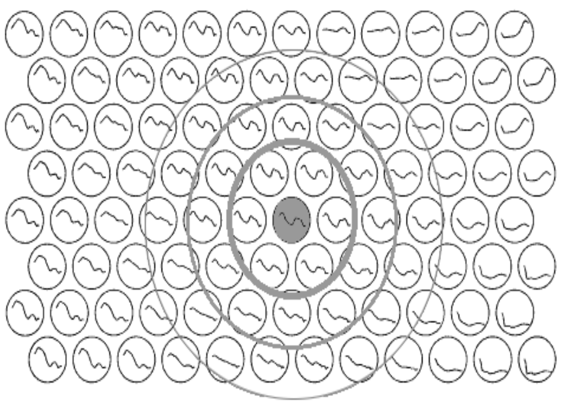

Figure 2. Example of Gradient Descent

As stated earlier, once the initial values have been determined, it is time to go ahead and train the map. The learning algorithm for a SOFM has five steps. Each will be described in detail. The variables required are as follows:

t	=	Current Iteration
λ	=	Iteration Limiter
Wv	=	Current Weight Vector
D	=	Target Input
$\theta(t)$	=	Neighbourhood Function
$\alpha(t)$	=	Learning Restraint

It has been shown in the previous subsection that the variables (λ, θ(t)) have been initialized in the initial setup. The remaining variables will now be explained.

t - Current Iteration
The variable (t) is the counter of the current iteration through the SOFM's training. This variable is consecutively incremented until it is greater than (λ).

Wv - Current Weight Vector
The variable (Wv) is the vector value stored in a node that is used to find similarity between itself and the input vector. These values are all set randomly at startup, and are modified during training. Because these values are initially random, they do not represent any training data. This is an important fact because it means that the map does not compare input vectors to other training data.

D - Target Input
The variable (D) is the target input and is the input vector for any iteration (t). On each iteration, an input vector (D) is selected from the training data and is calculated against the winning node in order to train the map.

α(t) - Learning Restraint
The variable (α(t)) is the learning restraint of the SOFM. The learning restraint is responsible for restricting the learning of the map as time increases. This is implemented so that the map learns much slower, and more gradually, near the end of training, as to allow the map to converge towards a solution set.

The SOFM learning algorithm follows five steps. These steps are as follows:

1. Randomize the map's nodes' weight vectors
2. Grab an input vector
3. Traverse each node in the map
   i. Use Euclidean distance formula to calculate the similarity between the input vector and the map's node's weight vector
   ii. Track the node that produces the smallest distance (this node is the winner)
4. Update the nodes in the neighbourhood of winner by pulling them closer to the input vector
   $$Wv(t+1) = Wv(t) + \Theta(t)\alpha(t)(D(t) - Wv(t))$$
5. Increment t and repeat while $t < \lambda$

Next, this paper will look at the methodology that was undertaken.

3. Methodology

This chapter will explain the reasoning behind the approach undertaken in this paper, as well as adjustments that had to be made in order to integrate the dataset into the SOFM.

3.1. Preprocessing

After analyzing the dataset, and because of the unbiased nature of the SOFM neighbourhood comparison algorithm, it was realized that some preprocessing of the dataset is required before the algorithm may properly read it. As opposed to other machine learning techniques that employ biases in order to balance the influence of certain columns of data, the SOFM comparison algorithm uses a Euclidean distance calculation which does not have that luxury. Instead, because every column of data is treated with equal significance, the data in the dataset must be normalized in order to make each piece of data relative to each other piece.

In order to normalize the data, the dataset is loaded into memory, which each value being transformed by the following formula:

$$NormalizedValue = \left(\frac{InitialValue}{\log(\max + 1)} \right)$$

The max value differs dependent on which normalization type is used, local or global. When using local normalization, *max* is the maximum value of the column of data in which *InitialValue* is found. When using global localization, *max* is the maximum value of the entire dataset.

3.2. Training the SOFM

Training the SOFM conducted by [1] is rather straight forward. The data from the training dataset is read row by row, compared against the existing map to determine the most similar node, the neighbourhood is adjusted accordingly, and the system moves to the next row. This entire process is documented in Chapter two, and it is this exact process that is followed for the map training in this research. After the map has been sufficiently exposed to the training dataset, it is time to begin classification.

In order to reduce the complexity of the comparison algorithm after the map has been trained, the training dataset was once again compared against the map. The data was compared using the same Euclidean distance formula as prior but instead of training the map further, a matrix was used to keep track of the number of times a particular node on the map was selected as the candidate node. This allows for the data to be laid out on a two dimensional grid.

For this paper, the training of the map was conducted in a similar manner as [1], but with the difference being the reduction of the dimensionality of data. Whereas their map trained the SOFM using the 37 dimensions of data within the training dataset, the work conducted in this paper reduced the dimension to 26, a reduction of 11 columns, or 30%. This was done to hopefully accomplish two things. First, the reduction of data would decrease the map training time, as well as the testing time afterwards. Secondly, and most importantly, the reduction would hopefully increase the percentage of successful attack detection by relieving the Euclidean distance similarity calculator of the useless data that this author hoped had caused detection issues for the remote-to-local attacks.

The idea for reducing the dimension of data came from [3], where Baig et al. had chosen the fifteen most influential features of data to train an AODE-based intrusion detection system, with promising results.

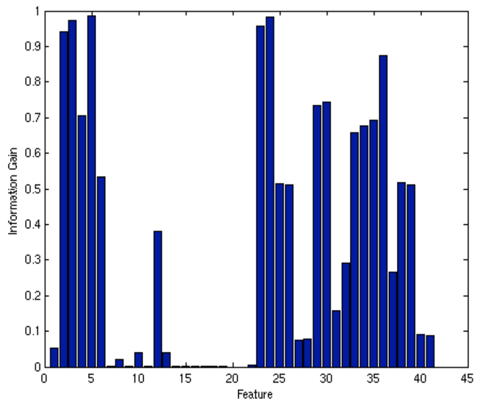

Figure 3. Results From Kayacik's Feature Relevance Analysis [4]

As for choosing which features to remove, an analysis from Dalhousie University provided the solution. Kayacik et al. [4] conducted a feature relevance analysis on the 1999 KDD Cup training dataset to determine which features were most important to correctly classify the data. Their analysis yielded the results shown in Figure 3.

Figure 3 shows there are several features that offer very little information to the overall solving of the intrusion problem, and even more that offer no information at all. It is with these results that this paper eliminated those features. For this paper, the features were removed: 7, 9, 11, 14, 15, 16, 17, 18, 19, 20, and 21. These values represent the features of the dataset which provide the least amount of information to the task of differentiating attacks from normal data. It is important to note that both [1], and the work conducted in this paper, did not use features 1-4. After analyzing the results from [4], future work should include possibly incorporating features 1-4, and especially features 2 and 3, as they yield the 5th and 6th most important information gain of the entire dataset. This inclusion will be discussed in the Future Work section of this paper.

3.3. Refinement

This section will explain the steps taken after the initial training of the SOFM in order to reduce the complexity of the analysis of the results, as well as to reduce the false positive count for each attack type.

After the matrices had tabulated the sums of hits on each particular node, a separate matrix for each attack type (DoS, Probe, U2R, R2L, Normal), a vector was created with the Cartesian points of each "hotspot" –the spot in the matrix where the number of hits was not 0– to further reduce the complexity of the comparison for test data against the trained map. These vectors dramatically reduced the number of nodes checked in order to find the candidate node. In the research conducted, the size of the map was 100x100. This creates 10000 cells that need to be checked each iteration of the algorithm in order to find the candidate node. By creating the vectors of hotspots, the number of nodes that are required to be checked is decreased 100-fold. Table 1 shows the size of each of the vectors for each attack type, using the 10% training set, as well as for local and global normalization. Table 1 also shows that by using global normalization, the number of hotspots decreases, meaning that the map was trained tighter.

After analyzing the results with the hotspot vectors and global normalization, it was discovered that the false positive rate was somewhat high. After some consideration and analysis, it was noticed that some of the vectors overlapped on the map, causing a hit on more than one vector, and skewing the results. To improve upon this, a pruning algorithm was implemented.

Table 1. Length of Hotspot Vectors [1]

	Local Normalization	*Global Normalization*

10% Training Set	`DOS Size: 59`
	`Probe Size:69`
	`U2R Size: 11`
	`R2L Size: 30`
		`DOS Size: 32`
		`Probe Size:34`
		`U2R Size: 9`
		`R2L Size: 22`

The algorithm used for pruning the vectors was simple, yet extremely effective. The algorithm used compared each vector against each other vector, and if any overlapping nodes were found, then the node with the least positive influence on its respective category was eliminated from the respective vector. For instance, if an overlapping node was found in the Probing vector and the Denial of Service vector, a comparison was done between those two types. If the node in question only provided 1 positive hit in training out of 1000 for the Probing category, and that same node provided 50 positive hits in training out of 1000 for the Denial of Service category, then the node would be eliminated from the Probing vector and would remain in the Denial of Service vector. Table 2 illustrates the reduction in the number of hotspots due to the vector pruning algorithm. The desire to implement such an algorithm it is quite evident.

Table 2. Hotspot Vectors with Vector Pruning [1]

	Local Normalization	*Global Normalization*

Without Pruning	`DOS Size:  59`
	`Probe Size:69`
	`U2R Size:  11`
	`R2L Size:  30`
		`DOS Size:  32`
		`Probe Size:34`
		`U2R Size:  9`
		`R2L Size:  22`

With Pruning	`DoS Size:  39`
	`Probe Size:58`
	`U2R Size:  11`
	`R2L Size:  22`
		`DoS Size:  17`
		`Probe Size:27`
		`U2R Size:  8`
		`R2L Size:  7`

Figure 4 shows the results published by Wilson et al. [1]. Figure 4 displays the successful classification of three of the four attacks types, as well as a high percentage of correct normal data classification. What Figure 4 also shows is the lack of correct detection of the remote-to-local attack type. This flaw was the basis for the biased pruning algorithm implemented in this paper.

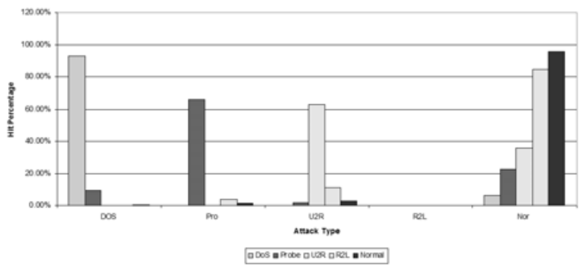

Figure 4. Detection Rates With Vector Pruning [1]

Wilson's pruning algorithm removed the weaker of the two conflicting hotspots, regardless of type. The pruning algorithm implemented in this work biased the pruning algorithm towards maintaining hotspots for attack types, regardless of the negative impact for other types; notably normal data detection. A simulation was conducted with a bias on the R2L vector when compared to the normal vector. As Figure 4 shows, most of the false positive results for the R2L attack type were found in the normal data so this is, naturally, where a bias would be most useful. Figure 6, found in the next chapter, displays the results of this simulation.

4. Results

After refining the method undertaken by [1], and conducting simulations to attempt to improve their results, this paper has found the following.

Figure 5 shows the results of the reduced-feature SOFM training and testing. The results from Figure 5 show a reduction in detection rate for denial of service, user-to-root, and normal data, but an increase in the detection rate of probe attacks by 11%. These results show that removing the least important features does not have the overall effect of increasing detection rate, as was initially thought. Instead, the elimination of these features only improved the probe attack detection rate. The remote-to-local attack rate remained at 0%, which reinforces the need for a biased pruning algorithm.

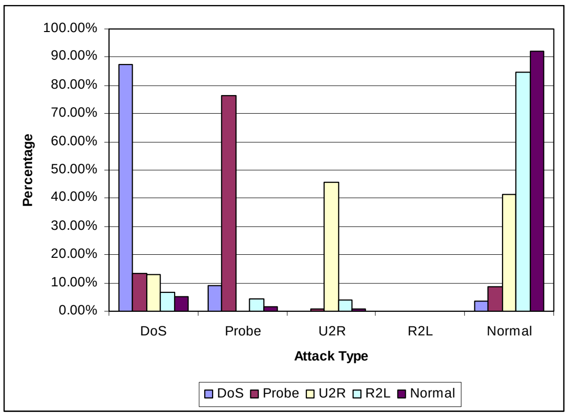

Figure 5. Results From Feature Reduced SOFM Training

Figure 6 shows the results of the R2L-biased pruning algorithm.

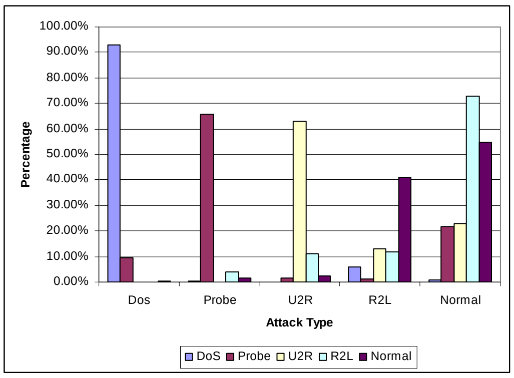

Figure 6. Results From R2L-Biased Vector Pruning Simulation

These results show that implementing a biased pruning algorithm on a full-feature set-trained SOFM yields a 12% improvement for R2L attacks, but this advancement brings with it a drawback of 41% positive detection for normal data. Figure 6 shows only a 55% positive detection rate for normal data. Figure 6 shows one other key piece of information. It shows that the SOFM is unable to completely separate R2L data from normal data. By biasing the pruning algorithm into forcing successful R2L detections, the algorithm eliminates normal attack detectors that exist on the same SOFM node, causing the high false positive rate.

One last result of note is the overall detection rate. Reducing the number of features used to train and test the system caused the overall detection rate to rise 0.27%, from 99.45% to 99.72%. Although this may seem like a marginal amount, it equates to a detection increase of 1104 new rows of data from the testing dataset not previously detected by [1].

5. Conclusions

After conducting research into improving the original work by Wilson et al. [1], this author has concluded that the biasing of the pruning algorithm has more of a detrimental effect than a positive one. The reduction of features from the dataset yielded a slight increase in probe detection rates, but also yielded a slight reduction for denial-of-service, user-to-root, and normal data. This author feels that more work into feature selection is needed before an ultimate conclusion can be drawn on this matter. Most important is the inclusion of features 1-4, as a discretized value, which may yield more positive results in the future.

6. Future Work

Future work in this area of research is abundant. There are a few key areas that are of particular interest to this author.

First, this author would like to complete a feature relevance analysis for the testing dataset to see which features align, and which differ, from the training dataset as shown in the feature analysis above. The lack of improved results by reducing features implies that the training and testing datasets should have different feature relevance graphs.

Second, creating a vote-based system of SOFMs is of particular interest. This author feels that a more specific system, accomplished by separating the data amongst different maps, would help to separate the R2L data from the normal data, and would dramatically increase results.

Third, this author would be interested in testing this system against another dataset. The KDD Cup dataset, used in this and other work, is in known for being particularly difficult to work with. Since this system has provided very impressive results relative to other systems, it would be interesting to see how it fares with other datasets.

7. Acknowledgements

The authors would like to thank NSERC for support.

8. References

[1] Ryan Wilson and Charlie Obimbo. Self-Organizing Feature Maps for User-to-Root and Remote-to-Local Network Intrusion Detection on the KDD Cup 1999 Dataset, *Proceedings of World Congress on Internet Security (WorldCIS-2011)*, pp. 56-61, 2011.

[2] Bernard Pfahringer. Winning the KDD99 Classification Cup: Bagged Boosting. *SIGKDD Explorations*, 2004.

[3] Zubair A. Baig, Abdulrhman S. Shaheen, and Radwan AbdelAal. An AODE-based Intrusion Detection System for Computer Networks, *Proceedings of World Congress on Internet Security (WorldCIS-2011)*, pp. 42-49, 2011.

[4] H. Günes Kayacık, A. Nur Zincir-Heywood, Malcolm I. Heywood. Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets.

[5] Jose Nazario. Russia: Opposition Websites and DDoS. http://asert.arbornetworks.com/2009/01/russiaopposition-websites-and-ddos/, January 2009. Accessed: April 01, 2010.

[6] Nathan McFeters. Recent CNN distributed denial of service (DDoS) attack explained. http://www.zdnet.com/blog/security/recentcnn-distributed-denial-of-service-ddos-attackexplained/1054/, April 2008. Accessed: April 01, 2010.

[7] ACM KDD Cup. http://sigkdd.org/kddcup/index.php. Accessed: January 01, 2010.

[8] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/ kddcup99.html, October 1999. Accessed: January 01, 2010.

[9] Maheshkumar Sabhnani and Gursel Serpen. Why Machine Learning Algorithms Fail in Misuse Detection on KDD Intrusion Detection Data. *Intelligent Data Analysis*, 8:403–415, September 2004.

[10] Ali Ghorbani, Wei Lu, and Mahbod Tavallaee. *Network Intrusion Detection and Prevention: Concepts and Techniques*. Springer, New York, NY, July 2009.

[11] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer, New York, NY, 1989.

[12] Teuvo Kohonen. *Self-Organizing Maps, 3rd Edition*, Springer-Verlag New York Inc., Secaucus, NJ, 2001.

[13] Yonggang Liu, Robert H Weisberg, and Christopher N. K. Mooers. Performance Evaluation of Self-Organizing Map For Feature Extraction, *Journal of Geophysical Research, Vol. 111, 2006.*