Anomaly Detection using prior knowledge: application to TCP/IP traffic

Alberto Carrascal[1], Jorge Couchet[2], Enrique Ferreira[3] and Daniel Manrique[4]
1: NEIKER: Instituto Vasco de Investigación y Desarrollo, acarrascal@neiker.net
2: Shell Corporation, Uruguay, jorge.couchet@shell.com
3: Facultad de Ingeniería y Tecnologías, Universidad Católica del Uruguay, enferrei@ucu.edu.uy
4: Dpto. Inteligencia Artificial. Facultad de Informática. Univ. Politécnica de Madrid, dmanrique@fi.upm.es

Abstract. This article introduces an approach to anomaly intrusion detection based on a combination of supervised and unsupervised machine learning algorithms. The main objective of this work is an effective modeling of the TCP/IP network traffic of an organization that allows the detection of anomalies with an efficient percentage of false positives for a production environment. The architecture proposed uses a hierarchy of Self-Organizing Maps for traffic modeling combined with Learning Vector Quantization techniques to ultimately classify network packets. The architecture is developed using the known SNORT intrusion detection system to preprocess network traffic. In comparison to other techniques, results obtained in this work show that acceptable levels of compromise between attack detection and false positive rates can be achieved.

1 Introduction

Nowadays, Information Technology (IT) constitutes a necessity in most organizations. Actually, companies of all sizes have their vital infrastructure based on IT for all their activities. This strong dependence has its risks, e.g. an interruption of the IT services can cause severe problems, endangering the company's assets and image or even worse, its clients as well [1].

The stability of an IT platform may be affected in several ways. The main sources of instability are the following: problems related to hardware; application problems; inadequate personal training and Information Security [2].

In the last years we have seen a steep rise in the importance of the Information Security as a main issue for companies and consequently, the amount of resources invested in technological solutions to this problem has increased accordingly. Table

1, taken from CERT [3], shows the increasing risks associated to Information Security since 1990.

Table 1. Number of security incidents reported to CERT annually.

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
#Incidents	252	406	773	1,334	2,340	2,412	2,573	2,134	3,734	9,859

Year	2000	2001	2002	2003	2004
#Incidents	21,756	52,658	82,094	137,529	204,625

The growth showed in security incidents makes the development of efficient techniques for intrusion detection a necessity. Mainly, there are two ways to approach the development of an Intrusion Detection System (IDS): Misuse Detection (MD) and Anomaly Detection (AD) [4].

Techniques based on Misuse Detection work with patterns, usually called Signatures, which are configured to match attacks based on some known system vulnerability. Most IDS available today correspond to the MD type, since they are easier to implement. However, MD has some important drawbacks that affect its effectiveness: first, they are somewhat rigid, only able to detect those attacks for which a signature is available. Secondly, a signature database has to be available and maintained regularly and manually since signatures can only be created once a type of attack has been detected and therefore has compromised several systems already. Finally, an intruder with sufficient knowledge of signatures may modified the attacks slightly to avoid known signatures, cheating the IDS based on them.

The Anomaly Detection approach uses Machine Learning (ML) algorithms [5] to model normal activity in an organization. In this way it may detect deviations that can be considered abnormal or suspicious. Most of the drawbacks attributed to MD systems can be overcome by the use of an anomaly detection IDS, which may be able to adapt dynamically and automatically to the relevant characteristics of an organization's activities. In this way, it would not be necessary to know the attacks beforehand to detect them, improving the response time to a security attack. Consequently, the majority of the recent research in the Intrusion Detection area is focused in this direction as can be seen in [6,7,8].

One of the main issues with AD systems is a high percentage of false positive detections (*Normal* cases classified as *Attacks*). This is a very important issue to resolve for practical purposes. In a typical system the percentage of normal traffic is considerably larger than abnormal traffic, therefore, an IDS with a high percentage of false positives could potentially generate an alert file with most of its records due to false positives instead of real anomalies.

Several methods that use ML techniques such as Support Vector Machines (SVM) or K-Nearest Neighbor (KNN) to build an AD system. They have shown a high rate of detection but also a high percentage of false positives as well, making them very difficult to implement in a real system [8]. Recently, some other works have made use of Self Organizing Maps (SOM) [9] to address the issue of false positives. They show a comparable detection rate with a significant decrease in the number of false positive detections [10, 11]. In these works the SOMs are trained

using information at packet and connection levels obtained with the IDS called Bro [12]. Although the results are positive, the percentage of false positives is still large to use such a detection system in a real situation. There may be a better way to exploit the information acquired by the SOMs to classify the incoming traffic.

In this work a new architecture aiming to solve the AD problem is presented. It is based on a hierarchy of SOMs combined with LVQ [9] to reduce the percentage of false positives. Only packet level information is used to analyze its contribution in the detection process. The system is implemented using the IDS Snort [13].

1.1 Self Organizing Maps

A SOM [9] is a type of neural network with a competitive unsupervised learning algorithm that performs a transformation of the input space. In general, it consists of a 2D dimensional map of neuron-like units. Each unit has an n-by-1 weight vector m_i associated, with n the dimension of the input space. That structure also determines a neighborhood relationship between the units. The basic SOM has a fix structure and number of units. The number of units determines the granularity of the transformation affecting the overall sensitivity and generalization ability of the map.

During the iterative training process, the unit weights will be adjusted to find common features, correlations and categories within the input data. Because of this, it is usually said that the neurons self organize themselves. Actually, the map tends to approximate the probability density of the input data. Weight vectors tend to zones where there is more input data and few units will cover zones of the input space where there is less information.

During a training step an input vector *x* is randomly chosen and the unit weight vector closer to *x* is found. That defines a winner unit c, such that

$$c = \arg\min_i \|x - m_i\| \tag{1}$$

where the symbol ||.|| refers to a norm, usually the Euclidean. The weight vectors of the unit c and its neighboring units are adjusted to get closer to the input data vector. A common update rule is given by

$$m_i(k+1) = m_i(k) + \alpha(k)(x - m_i(k)),\ i \in N_c(k) \tag{2}$$

with k denoting the training step, *x(k)* is the input vector chosen from the input data, $N_c(k)$ is the neighborhood set of unit c and α(k) is the learning rate at step k. The learning rate α(k) goes between 0 and 1 and decreases with k. Training evolves in two phases. During the first phase "big" values of α are used (from 0.3 to 0.99) while the second phase sees smaller values of α (below 0.1). $N_c(k)$ is usually fixed. Bigger neighborhoods are sometimes used in the first training phase.

1.2 Learning Vector Quantization

Learning Vector Quantization (LVQ) may also be considered a type of neural network like the SOM architecture [9]. An LVQ network has a set of units and weight vectors m_i associated to them. There are several training algorithms for LVQ. The algorithm used in this paper, LVQ1, is a supervised learning algorithm for

classification, i.e. each input vector has a class assigned to them that the network would like to learn. Initially, each unit is assigned to a class. At step k, given a vector *x* randomly chosen from the input data, we find the weight vector closer to it m_c, with *c* given by (1). The vector m_c is updated in the following way:

$$m_c(k+1) = \begin{cases} m_c(k) + \alpha(k)(x - m_c(k)), \text{ if } x \text{ in same class as } m_c \\ m_c(k) - \alpha(k)(x - m_c(k)), \text{ else} \end{cases} \quad (3)$$

where α(k), the learning rate, is bound between 0 and 1 and can be constant or decrease monotonically with each time step.

1.3 Fundamentals of TCP/IP

Transmission Control Protocol (TCP) and Internet Protocol (IP) refers to the most widely used protocols to send and receive data through a network system. Because they work together at different levels of the system (transport and network layers respectively), they are usually named together as TCP/IP.

TCP/IP specifies how to establish and close a connection between processes in different parts of the network and how to send and receive messages between them. A TCP entity accepts messages from a process and breaks them up in pieces up to 64K bytes to send as *datagrams* by the corresponding IP entity. It is up to TCP to guarantee that all datagrams are received and the original message reassembled correctly. TCP datagrams have header and data sections. The minimum TCP header is 20 bytes long. It mainly contains source and destination addresses, packet sequence number for reassembly, several flags for connection purposes (URG, ACK, EOM, RST, SYN and FIN), a header checksum, some optional parameters and its own length.

An IP entity takes TCP datagrams, add its own IP header to generate what is called *network packets* and sends them to its destination. The IP header, which is also 20 bytes minimum, contains source and destination addresses, header and total length, protocol, type of service, flags, checksum and other attributes. In particular, type of service (1 byte) allows the user to select the quality of service it wants, from speed for voice connections to reliability for file transfer uses.

This is a very brief overview of TCP/IP, interested readers should refer to [14] for a complete explanation of computer networks and protocols. In this paper the information contained in the TCP and IP headers of the network packets is used to detect anomalies in network traffic.

2 Anomaly Detection based on SOM/LVQ

The three-layer architecture proposed is shown in Figure 1. The input to the classifier proposed is a vector comprising the main attributes of a window of predefined length of network packets. The output gives the class to which the input is classified: *Normal* or *Attack.*

The first layer examines the variation of each attribute over the time window separately. The second layer correlates the information from the first layer between

attributes and makes a three-class decision: *Normal*, *Attack* or *Indefinite*. The class *Indefinite* introduced refers to the vectors that are *close* to the *Normal* class but have some *Attack* elements. The third layer decides whether the vectors in class *Indefinite* should be in *Normal* or *Attack* using a larger SOM network.

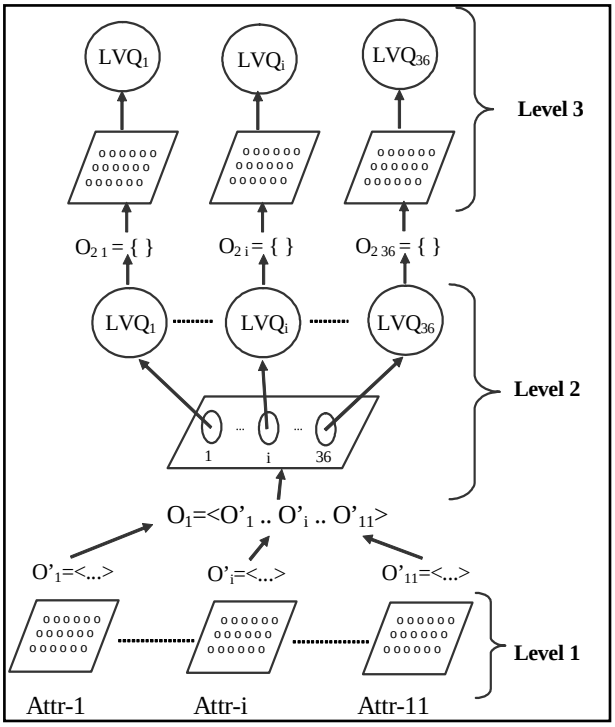

Fig. 1. Main architecture proposed.

2.1 First Level - Feature Clustering

The first level is composed of eleven SOMs, one per attribute selected to characterize TCP traffic. Each SOM takes an input vector of dimension twenty, given by a time window of length twenty of the selected attribute as shown in Figure 2. The goal at this level is to identify the main features of each attribute to obtain a model of the traffic analyzed. Once these SOMs are trained, six units per SOM are selected to reduce the dimensionality of the information to be passed to the second layer. The criteria used to select which units to select are two: the use of a *Potential Function* [10], or the selection of three units associated to normal traffic and three units associated to attack traffic.

2.2 Second Level - Aggregation and Classification

The second level of the architecture, shown in Figure 3, consists in a 6-by-6 SOM and a set of LVQs, one per SOM unit. The input vector to this SOM is

composed of the distances of the input vector to the first layer, to all the units selected in the first level SOMs. Therefore, the dimension of the input vector to the second level is 6 times 11. The objective of this SOM is to capture the correlations between the features found by the SOMs of the first level, in order to make a better characterization of the traffic being studied.

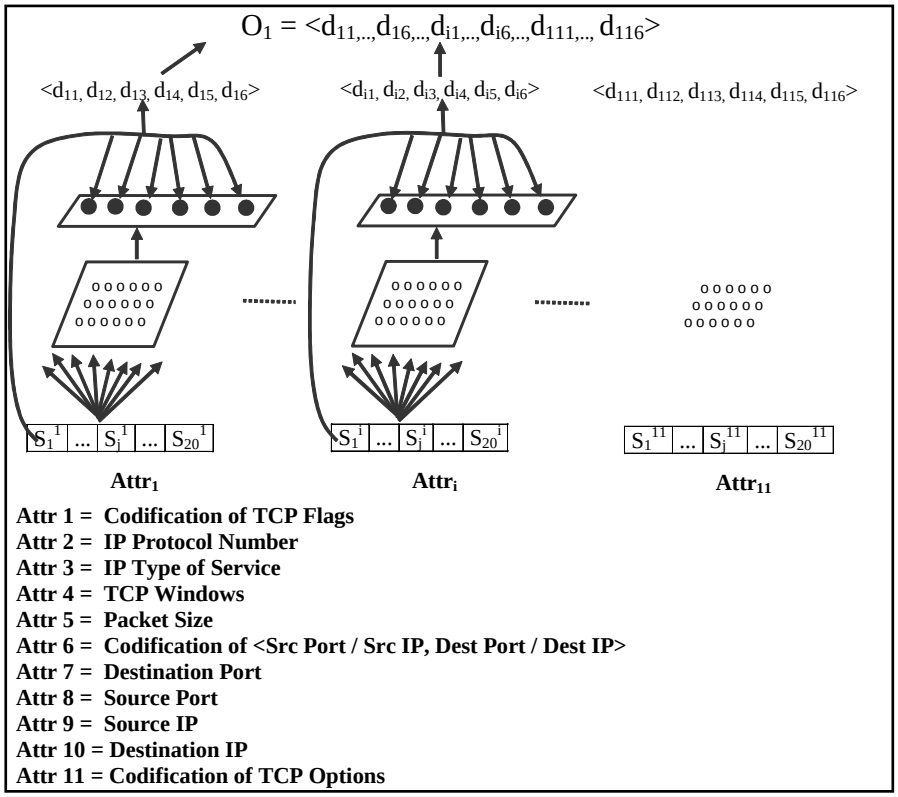

Fig. 2. Attributes extracted and information flow at the first level.

Once the SOM is trained, each LVQ associated to each unit is tuned in a supervised manner using the subset of inputs to the SOM that makes the unit associated to the LVQ the *winner* in the SOM sense. The label used to train the LVQs for each input is defined as:

$$MyLabel = \frac{\#\, attack\ packets}{\#\, packets} \tag{4}$$

where the number is computed over the time window of the input vector.

It is possible to train the LVQ network in two ways: using the values of *MyLabel* as defined or discretizing the values of *MyLabel* in *Normal*, *Attack* and *Indefinite*, where: 0 < *MyLabel*(*Indefinite*) < *Attack_threshold*. In tests, the value of *Attack_threshold* used was 30 %. In the last case, the system is making a 3-class decision at this level, leaving the final classification of the packets labeled in the *Indefinite* class to the next level.

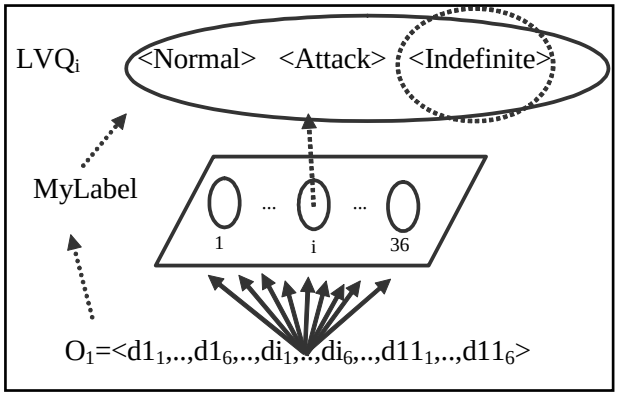

Fig 3. Second level processing.

2.3 Third Level - Indefinite Class Processing

At this level the architecture proposed analyses those input windows that have a relatively low percentage of attack packets, i.e. lower than *Attack_threshold*. Each LVQ from the second layer is associated to a 10-by-10 SOM that is trained with the packets linked to the windows classified as *Indefinite* by that particular LVQ. A LVQ network is assigned to each SOM to make the final classification using as input the distances of each packet to all the units in the associated SOM. The overall processing is shown in Figure 4.

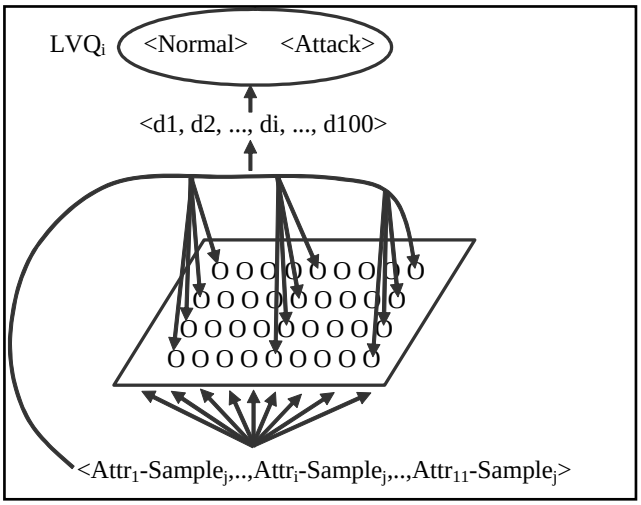

Fig. 4. Third level processing.

3 Experimental Results

The main objective in this work is to find an acceptable compromise between the Detection Rate (DR) and False Positive Rate (FPR) defined by:

$$DR = \frac{\#\,\text{Attacks Detected}}{\text{Total Number of Attacks}}, \quad FPR = \frac{\#\,\text{of False Positives}}{\text{Total Number of Normals}}$$

Data for experiments is taken from the DARPA 1998 Intrusion Detection Problem [15]. Two sets were built: the training set contains **1:514,848** records while the test set has **765,029** records. Results shown are computed over the test sets for networks trained using the training set. Training and test sets were generated from DARPA data using SNORT to extract the TCP/IP attributes selected. Only information at the packet level is used. The preprocessor developed works as a plugin to SNORT. It is also able to extract attributes at other levels (e.g. TCP connection) and protocols (e.g. UDP and ICMP). The implementation of SOM and LVQ is based on the package SOM_PAK [16]. Some modifications to this package were necessary to handle large data files as required for this application.

The packet attributes used, shown in Figure 2, were selected by its relevance in the TCP/IP protocol and hence its importance to model network traffic. The attributes that may have several values in the same packet, such as the TCP flags, were coded to reduce the dimensionality of the problem.

Tables 2, 3 and 4 summarize the results obtained with the most relevant experiments run on the architecture proposed. They use the following notation:

Classification: Refers to the amount of classifications performed by the system.

Correct: Represent the number of correct classifications per class made.

Deviations: Show the number of incorrect classifications of each type made.

The initials **N**, **I** and **A** are used for *Normal*, *Indefinite* and *Attack* respectively. In the case of deviations we use, for example, **N-I** to indicate the packets belonging to the *Normal* class but classified as *Indefinite*.

It should be noted that, at the third level the *Indefinite* class is sorted between *Normal* and *Attack*. At this level the SOM is not trained with a time window but with each of the twenty individual packets that composed each window. Due to this, each pattern at the second level is transformed into twenty patterns at the third level.

Table 2. Use of Potential function and *My Label* for training.

	Classification			**Correct**			**Deviations**					
	N	**I**	**A**	**N**	**I**	**A**	**N-I**	**N-A**	**I-N**	**I-A**	**A-N**	**A-I**
Level 2	396674	132355	236000	227250	7657	208396	41845	14382	28584	13222	140840	82853
Level 3	522269	0	71111	481170	0	50926	0	20185	0	0	41099	0

Table 3. Use of Potential function and *My Label* discretized for training.

	Classification			**Correct**			**Deviations**					
	N	**I**	**A**	**N**	**I**	**A**	**N-I**	**N-A**	**I-N**	**I-A**	**A-N**	**A-I**
Level 2	648290	66580	50159	239197	11238	33256	36754	7526	28848	9377	380245	18588
Level 3	859386	0	429934	701701	0	112103	0	317831	0	0	157685	0

Table 4. Use of 3 centers associated to *Normal* and 3 to *Attack*, and *My Label* discretized.

	Classification			Correct			Deviations					
	N	**I**	**A**	**N**	**I**	**A**	**N-I**	**N-A**	**I-N**	**I-A**	**A-N**	**A-I**
Level 2	413132	44929	306968	272431	3922	301335	6591	4455	44363	1178	96338	34416
Level 3	637367	0	261213	246029	0	233798	0	27415	0	0	391338	0

It can be observed that the best results are achieved when the selection of units to represent level one information is based on the classes to discriminate, in this case three for each class. The performance of this option is then:

$$DR = \frac{301335 + 233798/20}{301335 + 96338 + 34416} = 72\% \quad FPR = \frac{4455 + 27415/20}{272431 + 6591 + 4455} = 2\%$$

With respect to the results achieved in [10], the value of FPR is improved by 73% while DR decreased by only 19 %. In general, Table 5 resumes some results obtained in previous works where ML techniques were used for AD [8, 10].

Table 5. Results of other works using AD methods

Method	**Detection Rate**	**False Positive Rate**
Clustering	93%	10%
K-NN	91%	8%
SVM	98%	10%
SOM Hierarchy	89%	7.6%

It can be noted that the DR achieved here is below the ones obtained in the works presented in Table 5. A better inspection of the experiments also show that:

- An important percentage of the attacks presented in the data sets used corresponds to the *user to root* type. This type of attacks are quite difficult to model using TCP/IP protocol characteristics only.
- The prototype implemented is classifying TCP/IP packets up to now. Since a connection usually consists of hundreds of packets, it is expected that the **DR** may improve once the connection information is added to the system.

4 Conclusions

The AD method developed in this work introduces an efficient mechanism to reduce the false positives getting closer to generate sufficiently reliable alerts to be able to use an AD based IDS in a production environment.

It can also be observed that packet information is an important component in the detection of attacks. This work serves as a guide as to which packet information to use to model traffic for this purpose. However, the need to use connection level information is pointed out as well to reach an efficient DR with low FPR.

This investigation is based on a priori knowledge of traffic packets combined with ML techniques to set up the model and pattern recognition techniques for traffic classification. However, many steps are developed in an empirical manner which

limits the conclusions that can be made. Therefore, there is the need to develop formal ways to obtain bounds on rates and efficiency of the AD methods.

From the results of this work we are following this research in two directions. First, we are exploring more efficient ML techniques for the intrusion detection problem. Besides, we are developing a theoretical framework to obtain a deeper understanding of the problem which may allow us to get robust models that are feasible to implement in the real world.

5 References

1. CSI, Computer Security Institute. *2004 CS//FBI Computer Crime and Security Survey*. (2004); http://www.gocsi.com/.

2. C. Kruegel, F. Valeur, and G. Vigna, *Intrusion Detection and Correlation - Challenges and Solutions* (Springer Verlag, New York, 2005).

3. CERT Coordination Center, CERT/CC Statistics (1988-2005); http://www.cert.org/stats/.

4. E. Carter. *Cisco Secure Intrusion Detection System* (Cisco Press, 2001).

5. T.M. Mitchell. *Machine Learning* (McGraw-Hill, 1997).

6. D.S. Kim, H.-N. Nguyen, and J.S. Park, Genetic algorithm to improve SVM based network intrusion detection system, 19th International Conference on Advanced Information Networking and Applications, Vol. 2, (2005), pp.155-158.

7. M. Markou, and S. Singh, Novelty Detection: A Review, Part II: Neural Network Based Approaches. Signal Processing, Vol. 83 (2003), pp. 2499-2521.

8. E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S Stolfo, A Geometric Framework for Unsupervised Anomaly Detection: Detecting intrusions in unlabeled data, In D. Barbara and S. Jajodia, editors, Applications of Data Mining in Computer Security (Kluwer, 2002).

9. T.Kohonen. *Self Organizing Maps*, Third Extended Edition (Springer, 2001).

10. H.G. Kayacik. *Hierarchical Self Organizing Map Based IDS on KDD Benchmark*. Master's thesis, Dalhousie University (2003).

11. P. Lichodzijewski, A.N. Zincir-Heywood, and M.I. Heywood, Host -based Intrusion Detection Using Self-Organizing Maps, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, IJCNN (2002).

12. Bro. Intrusion detection system (2005); http://www.bro-ids.org/

13. Snort. open source network intrusion prevention and detection system (2005); http://www.snort.org.

14. A. S. Tanenbaum. *Computer Networks* (2nd Edition, Prentice-Hall, 1989).

15. MIT Lincoln Laboratory (1998); http://www.ll.mit.edu/IST/ideval/data/data index.html.

16. SOM_PAK**,** Helsinki University of Technology, Laboratory of Computer and Information Science (2005); http://www.cis.hut.fi/research/som_lvq_pak.shtml.