4-2003

Development of Kohonen Neural Network Application as a Pattern Recognition System for an Electronic Nose

Lori Lynn Evesque

DEVELOPMENT OF A KOHONEN NEURAL NETWORK APPLICATION AS A PATTERN RECOGNITION SYSTEM FOR AN ELECTROMC NOSE

by

Lori Lynn Evesque

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 2003

ACKNOWLEDGMENTS

Many people have helped me complete this thesis. I would like to begin by thanking my thesis advisor and committee chair Dr. Robert Trenary. His enthusiasm, intriguing and unconventional teaching style, and interest in neural networks not only convinced me to pursue further studies in the area of neural networks but kept me going when I was feeling this adventure was never going to end.

I thank my husband, Jonathan Davis, for the use of the Cyranose 320 Electronic nose, for guidance in choosing the odorant chemicals used in this investigation, making the flavor lab available to test and train the electronic nose, and most of all, for putting up with my extensive and never-ending educational career. I also greatly appreciate my children's patience with my educational pursuits. I hope they have been enriched by what in their view are extraordinary interests and pursuits compared to the mothers of their friends.

I would like to thank Dr. Jin Li and Olivia Deffenderfer at Cyranose for their help with learning how to operate the Cyranose unit and to train the nose and interpreting the data. I would like to thank the members of my thesis committee Drs. Donna Kaminski and Elise De Donker for reviewing my work. Finally, I would like to thank the many people at Western Michigan University who have helped me complete this research over the last few years. They include Dr. John Kapenga for suggesting various 3-D modelling software, Dr. Thomas Piatkowski for introducing me to the Maple software package used to plot my 3-dimensional clustering results, Dr. Karlis Kaugars for C++ programming assistance and many others.

Lori Lynn Evesque

DEVELOPMENT OF A KOHONEN NEURAL NETWORK APPLICATION AS A PATTERN RECOGNITION SYSTEM FOR AN ELECTRONIC NOSE

Lori Lynn Evesque, M.S.

Western Michigan University, 2003

Electronic noses are used to identify and characterize unknown odors in industry. Chemometrics and neural network algorithms are used as pattern recognition systems for these devices. Experimentation with Kohonen clustering as the pattern recognition system for electronic noses was not noted prior to 1997. [BEG] This thesis investigated the use of a Kohonen neural network algorithm as a clustering algorithm for electronic nose data using the chemometrics algorithms built into the electronic nose as a performance standard. A secondary aim was to improve the clustering and identification capabilities of the Kohonen network.

The unsupervised Kohonen network was not able to cluster the electronic nose data. Duplicating the data pre-processing performed by the electronic nose, fine-tuning the visualization of clustering data, and varying the learning and weight update rates offered minor improvements but did not allow for accurate identification of samples. Significant improvements were obtained when the network was changed to a semi-supervised network by incorporating average sensor values of the known odorant samples into the network weights. This improved the clustering effectiveness of the network to 60-70% compared to a 100% effectiveness of the chemometrics system built into the nose. Several avenues were identified for further study to improve the effectiveness of the neural network for use with the electronic nose.

TABLE OF CONTENTS

Table of Contents—continued

Table of Contents--continued

LIST OF TABLES

LIST OF FIGURES

List of Figures—continued

CHAPTER I

INTRODUCTION

The Limitations of the Biological Nose

This thesis investigates the use of an artificial, electronically based nose to supplement and improve upon the capabilities of the human nose. The effectiveness of two different approaches to a pattern recognition problem using electronic nose sensor data is compared. Specifically this project develops an artificial neural network clustering system and compares its effectiveness to the "chemometric" methods built into the commercially available nose. Chemometrics is defined as the use of mathematical methods to extract important but often hidden information from data. [BEG]

Human beings have been using their senses in work environments for thousands of years. All five senses are used to different degrees in varying occupational areas. For most of the senses, human beings have developed enhancements to aid in the capabilities of the senses. Eye glasses, microscopes, binoculars, etc. all increase the capabilities of the eyes to see objects beyond the normal range of human sight and to correct deficiencies in sight. Many other devices have been developed as aids to sight in places where conditions are difficult or dangerous for humans to work. These include cameras and infrared sensors, which can be used in high and low temperature, high pressure, or other extreme environments. Hearing aids, both to improve deficient hearing and to "listen" beyond the scope of the human ear are also widely used. Equipment is available in industries to "feel" and "look" for faults in manufactured products removing the reliance on human hands and eyes. Senses that have not been elevated much beyond the capabilities of humans are

those of taste and smell.

While there are a few cases where humans utilize the superior sense of smell of certain animals, particularly dogs, it is noteworthy that applications in which taste and smell are important usually rely on human beings. This is especially true in the food and perfume industries, which utilize both highly trained individuals and average ability people in the development and improvement of perfumes and food products.

Unfortunately, the human nose is limited for many of these tasks. Human beings are predominantly visual mammals and do not use smell as a dominant sense. Therefore, the sense of smell is not as highly developed as it is in other animals, such as dogs. With many people, it can almost be said to be dormant or vestigial. It is possible for some people to be trained to better discriminate between different odorant molecules but such training is time consuming, expensive, and not all people can learn to develop this innate ability. The salaries paid to qualified "smellers" in the perfume industry (who can easily make six-figure salaries) amply support this claim.

There are other disadvantages to using humans where a sense of smell is important. The human nose tires easily, limiting the duration a person can accurately discriminate between smells. It is also possible for a person's sense of smell to be temporarily blocked after smelling very strong or noxious odors (such as when a person smells a skunk) or from exposure to odors for long periods, inducing anosmia or smell blindness. [Si] Reliance on human sensory panels, widely used in the food, perfume, and flavor industries, have other disadvantages such as a high degree of subjectivity. This results in poor reproducibility due to varying degrees of the health of panel members,

differences in the time of day, the effect of previous odors analyzed by panel members, time requirements for panel testing and high costs. There are also many instances where human sensory panels cannot be used due to the presence of hazardous odors, requirements for continuous operation and sensing needs in remote or difficult locations. There are myriad motivations for developing an "electronic nose".

The Electronic Nose

"Electronic noses" are used in a wide variety of industries and settings to characterize and identify individual odor molecules and complex mixtures of odors. Bartlett et al. (1997) define an electric nose in *Food Technology* magazine as "an array of chemical sensors, where each sensor has only partial specificity to a wide range of odorant molecules, coupled with a suitable pattern recognition system." The operation of an "electronic nose" is based on the way the biological mammalian nose works. In a mammalian nose, there are many chemical receptors known as olfactory receptors which, when combined with signal preprocessing in the olfactory bulb and pattern recognition in the olfactory cortex of the brain, make it possible for the mammal to smell and recognize a particular scent or odor. No single receptor identifies a specific odor. It is the collective effect of the odorant on all or many of the receptors that allows specific identification. [K]

The design is similar in the electronic nose. Equivalent to the olfactory receptors in an electronic nose are chemical sensors designed to react to odorant molecules. These sensors can be made of a variety of materials including organic, conducting, or non-conducting polymers, metal-oxide semi-conductors, surface acoustic wave devices, liquid crystal sensors, fiber optic sensors and others. [K] The sensors react to the odorant

molecules producing a measurable change in the sensor. Each sensor is designed to be slightly different from every other sensor in the unit. This results in a unique and characteristic "fingerprint" for each individual odor. [S-S] After the sensors measure the "fingerprint" of the odor, the pattern recognition phase can be used to identify the odor.

There are several different approaches used for the pattern recognition systems in electronic noses. These include statistical methods, often called "chemometric" methods, artificial neural networks (ANNs or NNs), and neuromorphic models. These approaches can be used singly or in combination to improve the robustness of the pattern recognition over those from individual techniques. [K]

Several chemometrics methods are used in conjunction with electronic noses to identify and classify odors. These methods include principle components analysis (PCA), least partial squares, discriminant analysis, discriminant factorial analysis, and cluster analysis.[K] Both supervised and unsupervised algorithms are used for pattern recognition. PCA is an unsupervised technique that is often used to reduce the dimensionality of a system and identify outliers. Supervised learning techniques such as K-nearest neighbor, K-means, and Canonical Discriminant Analysis are used for building an identification model and predicting unknowns. [L]

Electronic noses incorporating ANNs have been demonstrated in numerous applications. [K], [HKKK], [SHG], [Sa] In many of these applications, the number of detectable chemicals is generally greater than the number of unique sensors and less selective sensors can be used. [K] Some of the ANN algorithms used in conjunction with electronic noses include supervised algorithms such as back-propagation feed-forward

networks, learning vector quantizers, and fuzzy-ART maps. In some cases, fuzzy-neural networks produced considerably better performance than back-propagation networks. [BEG] Unsupervised ANNs include self-organizing maps (SOMs) and adaptive resonance theory networks. [K] Some unsupervised learning algorithms mimic the way the human brain works, as there is no separate learning stage. As of 1997, unsupervised learning neural network algorithms had yet to be applied in conjunction with an artificial nose. [BEG]

Neuromorphic approaches are based on building plausible models of olfaction based on biology and implementing them in electronics. [K] These approaches are not as well developed as other approaches and are not investigated in this paper.

The electronic nose used in this investigation was the Cyranose 320™ manufactured by Cyranose Sciences of Pasadena, CA. The Cyranose electronic nose was originally released in early 2000 and was chosen because of its practicality for use in the food and flavor industry. The unit was the first nose on the market to be portable enough for field use and economically feasible for even smaller companies to purchase. The system includes the sensing unit and associated pattern recognition software to be installed on a PC. The hand-held unit is lightweight, battery operated, and suitable for a variety of environments. There is an easily readable LED display in which instructions can be entered, samples taken, etc.

Goals of this Thesis

The primary aim of this thesis is to develop an unsupervised ANN to cluster odorant data for discrimination between different samples. The pattern recognition or clustering effectiveness of the ANN will be compared to the best capabilities of the Cyranose 320™ electronic nose. Pattern recognition consists of two phases: (1) training and clustering using the sensor data and (2) identification of unknowns. Only the first phase will be compared in this investigation. The data used for both pattern recognition methods was collected using the Cyranose unit and odorant chemicals provided by GLCC Co., a Michigan flavor house. A secondary aim was to investigate strategies to improve on the pattern recognition capabilities of the neural network. The neural network achieved partial success in clustering of the chemicals compared to the built-in chemometrics methods used by the Cyranose unit. This thesis is organized into the following chapters:

Chapter II presents basic background information about the three main subjects covered in the thesis. These include artificial neural networks, chemometrics pattern recognition methods, and the Cyranose 320™ electronic nose. The purpose is to give the reader enough knowledge about unfamiliar subjects and terminology to follow the discussions.

Chapter III describes the collection of data samples using the Cyranose 320™ electronic nose unit and training of the electronic nose using the build-in chemometrics

pattern recognition algorithms. Training consists of the development of a working pattern recognition method for successfully recognizing all of the chemical odorant samples used in the experiments.

Chapter IV introduces the basic framework of the experimental methodology used in the development of the project's neural network pattern recognition system using a known neural network software library. Data pre-processing and training details for the neural network are also included. This neural network evolved gradually as the experiments progressed with each experiment determining the next steps to be taken.

Chapter V is a narrative of the experiments indicating how and why each step was taken. The results of each step or experiment are discussed and these results helped to determine other possibilities for improving the pattern recognition abilities of the neural network system.

Chapter VI is a discussion and analysis of the results and discussion of the clustering capability of the NN compared to the chemometric methods used by the nose. Possible directions for future research in this area are also identified.

CHAPTER II

BACKGROUND

Artificial Neural Networks

The ability of computers to perform complex, sequential, logic-based information processing is immense. Computers have the ability to perform computations once considered beyond the scope of human endeavor. However, there has long been interest in other information processing systems, including artificial neural networks have certain performance characteristics in common with biological neural networks. Neural networks are characterized by (a) the pattern of connections between the neurons (architecture), (2) a method of determining weights on these connections (training or learning algorithm), and (3) a function applied to the net input to determine its output signal (the activation function), (often but not always present).

Warren McCulloch and Walter Pitts are generally recognized as the developers of the first neural network in 1943. They combined simple neurons into a network that had increased computational power. The weights on their simple network were set to perform simple logic functions. Combinations of these neurons could be arranged to perform more complex logic functions. One feature of McCulloch-Pitts networks that is used in many artificial neurons today is the idea of a threshold. If the net input to the neuron is greater than the threshold, the unit fires, or turns on; otherwise, it remains in the 'off' state. [F]

Over the next 30 years, investigations into neural networks continued with

researchers including Hebb (Hebb learning), Rosenblatt (Perceptrons), Widrow & Hoff (Adeline), Kohonen (Kohonen self-organizing networks), and others developing more complex and varied neural networks. In the late 1960's enthusiasm about NNs waned after clear demonstration of the limitations of simple single layer NNs. [MP] As methods for propagating errors from the output units back to the hidden layers and improved methods for training networks gained wide-spread publicity, research on NNs gradually picked up speed. Their complexity and performance improved with more powerful learning algorithms, incorporation of bias components, more complex activation rules, multiple layers, and other advances.

Methods for adjusting weights and training a network can be characterized as either supervised or unsupervised. In supervised training algorithms, training begins with the presentation of a sequence of training vectors to the network, each with an associated target output vector. The propagation of the training vector through the network results in a certain output. The difference between the output and the target determines how the weights are changed in the network as the training proceeds. In other words, supervised training assumes some amount of a priori knowledge about the system. For example, in an alphabet recognition system a particular input vector would refer to a particular letter. The amount the link weights need to be adjusted will be determined by the difference between the calculated output and the target output. Most supervised learning algorithms also incorporate an activation level for each neuron that is a function of the inputs it has received. The activation is sent as a signal to other neurons in multi-layer networks. Unsupervised learning algorithms do not require a priori knowledge of the system but

work to group similar input vectors together. No target vectors are specified. The network modifies the weights so that the most similar input vectors are assigned to the same output unit. Examples of supervised algorithms include back propagation, learning vector quantization and counter propagation. Unsupervised learning algorithms include Kohonen self-organizing maps and adaptive resonance theory.

Neural networks are currently being utilized in many areas. Automatic recognition of handwritten characters or spoken speech recognition and production applications is increasingly common. General-purpose multi-layer neural nets are being used for recognition of zip codes. [F] Easily available software makes it possible to speak into a computer and have the spoken words typed into a word-processing program. Applications for neural nets abound in the medical industry. An application called "Instant Physician" [F] is a neural net designed to organize large numbers of medical records and be able to give a "best" diagnosis and treatment for a new set of symptoms. In business, applications include insurance underwriting networks, mortgage application processing networks, and others.

The basic architecture of a simple neural network can be seen in figure 1. A common notation for describing neural networks is indicated below the figure. This notation, where applicable, will be used here.

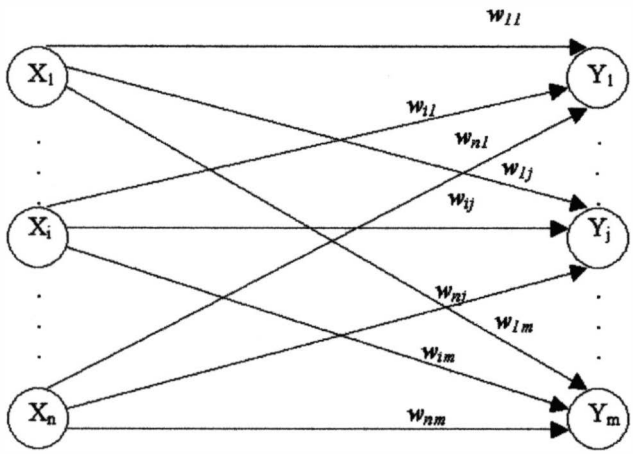


Figure 1: Single Layer Neural Network

Neural Network Notation

X_i Input unit, number of units equals n

Y_i Output unit, number of output units equals m

w_{ij} Weight on connection from unit X_i to unit Y_j

Δw_{ij} Change in weights before and after weight updating

α Learning rate: the rate to control the amount of weight adjustment at each step in training.

$f(x)$ Activation function

The neural network algorithm used for clustering and pattern recognition in the application investigated in this paper is the Kohonen "self-organizing" network, an unsupervised neural network algorithm. Self-organizing maps construct a topological map consisting of clusters of similar data points from the presented data. Interestingly, this property has been observed in the brain but is not found in other artificial neural networks. [F] The architecture of the Kohonen self-organizing map is essentially the

same as that shown in figure 1. The '*m*' output cluster units are generally arranged in one, two, or three-dimensional arrays depending on the purpose of network. For example, a character recognition network could be arranged as a two-dimensional output array (figure 2).

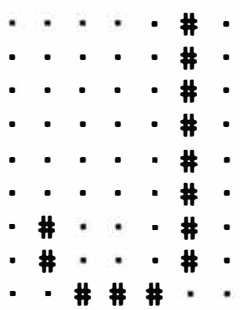


Figure 2: Two Dimensional Output Cluster Array

During the 'self-organization' or training process for a Kohonen network, the training input vectors are sequentially presented to the network. The difference between the input vector and the weights of the cluster units are calculated as in the following equation.

$$[2.1]\ D(j) = \Sigma\ (w_{ij}-x_i)^2$$

The cluster unit with the minimum difference, whose weight vector most closely matches the input pattern, is chosen as the "winner". The weights to the winning unit and neighboring units are then updated. The updating is calculated using the following equation.

$$[2.2]\ w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

This weight updating preserves and strengthens the topology that assigned the input vector the specific winning cluster. Typically, the winning cluster weights are changed by the greatest amount, with neighboring clusters being changed to a gradually lesser degree as the distance from the winner increases. Input vectors continue to be presented to the network until some pre-determined stopping condition is reached. In the fully trained network, the weight vector to a particular output cluster serves as an exemplar of the input patterns assigned to that cluster. Subsequent data vectors presented to the network which have values close to the weight vector of that cluster will be assigned to that cluster. Initial weight values for the network are generally randomly assigned. However, if some information exists concerning the distribution of the clusters that might be pertinent to a particular problem, the initial weights can be chosen in such a way as to reflect this knowledge. [F] (Refer to Appendix A for a more detailed description of the Kohonen neural network algorithm.)

Chemometric Analysis Methods

Chemometric analysis methods are common statistical computational techniques used to extract relevant but often hidden information from data. [BEG] These techniques are used in many other areas of science but the term "chemometrics" is commonly used in the chemical and food industries. In general, chemometrics covers areas of data sample and variable pre-processing, and pattern recognition algorithms.

Pre-processing of the data is an important initial step for data analysis in order to reduce the concentration and environmental effects of the sampling methods. Two types of pre-processing are available: sample and variable pre-processing. Sample pre-

processing is done to reduce systemic variation and works on the entire input sample vector, or row, of data. Sample pre-processing methods include normalization, which scales all samples uniformly; weighting, giving some samples more weight than others; smoothing, which reduces the amount of random variation; and base-line corrections which minimize systemic variation. Variable pre-processing works on each variable, or column, of data and is done to remove any inadvertent weighting that arises due to arbitrary units, as from vastly different samples. [Li] Methods used for variable pre-processing include mean centering, auto-centering, and variable weighting.

Following data pre-processing, a variety of statistical approaches to pattern recognition analysis are included in a chemometrics approach. These include principal component analysis (PCA), partial least squares, discriminant analysis, discriminant factorial analysis (DFA), and cluster analysis. [K] PCA is an unsupervised method that manipulates the feature matrix in order to represent the data using a smaller number of factors, or dimensions, making it possible to view the data in a smaller number of dimensions. This makes it possible for human pattern recognition to be used to identify structures, making PCA an extremely useful first step in multi-dimensional data analysis. PCA analysis is most useful when the dimensionality of the measurement space is large but where the samples reside in a small dimensional space. (i.e., small inherent dimensionality) PCA is also an excellent preliminary data exploration method for examining data vectors for expected or unexpected clusters or for outlier diagnosis. [BPS]

Several supervised statistical pattern recognition algorithms are then used when the goal is to construct a model to be used to classify future samples. These include K-nearest neighbor (KNN), Kmeans, and Canonical Discriminant Analysis (CDA). [Cy1] These methods conduct a cross-validation of the data samples as an initial step in the clustering. Cross-validation is the action of validating the training set by leaving out one data point from each class to build a model and predict the left out data points. [Cy1] All data points are left out once during the process. The overall correct prediction rate will indicate the quality of the training set and the applicability of the model.

K-nearest neighbor classifies unknowns according to a majority vote of the 'K' nearest neighbors in the training set in n-dimensional space. The value of K is determined during cross validation. Kmeans classifies unknowns based on the Euclidean distance between the class centroid and the sample in the sample space. Canonical Discriminant Analysis assigns unknowns to a class based on the Mahalanobis distance between the sample and the cluster centroid in canonical space. The clustering results are viewed on plots in two or three dimensions based on the dimensionality reduction accomplished using PCA. For more details concerning chemometrics methods, refer to Appendix B.

Cyranose 320™ Electronic Nose

Internally, the Cyranose 320™ electronic nose consists of 32 composite polymer odorant sensors. Different types of polymers are used in the nose but they are all nonconductive, absorbing polymers. [S-S] In the middle of each sensor is the non-conducting absorbent polymer. Electrodes bring an electrical current to one side of the

conducting absorbent polymer. Electrodes bring an electrical current to one side of the polymer and take current away from the other side. Embedded in the polymer are conductive carbon chains that bridge the electrical current from the electrode on one side of the polymer to the electrode on the other. (figure 3) When the nose is exposed to an odorant, the polymers absorb differing amounts of the odorant molecules. As molecules are absorbed onto the polymer, they break the circuit made by the carbon chains, increasing the resistance. The resultant 32 resistance values produce a result that is unique to that odorant chemical and is its "fingerprint" and can be compared to other fingerprints to find a match. [S-S]

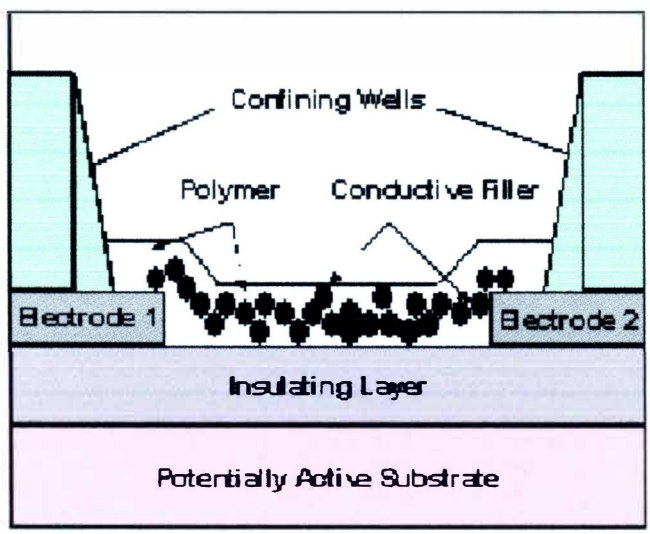


Figure 3: Structure of the Cyranose Sensor
Source: Cyrano Sciences, Inc, *Cyrano Sensor Array.doc* obtained from and used with permission of Cyrano Sciences, Inc. via email, 3/25/03.

The pattern recognition system that comes with the nose is a chemometrics based system. The system includes pre- and post-processing of the signal data and algorithm

utilization for pattern recognition. [Li] The raw data is filtered to eliminate high frequency noise and reduced using a baseline correction method. Finally, the data is normalized and scaled using a choice of several techniques. The pattern recognition algorithms include PCA for outlier detection and supervised algorithms including K-nearest neighbor, K-means and CDA for building models and prediction of unknowns.

The data gathering capacity of the Cyranose 320™ electronic nose is substantial but the terminology for the individual data packages and their interconnectedness can be confusing. A description of the terms used in this thesis is shown below:

1. *Sample Set:* The Cyranose unit and software can hold 5 complete sets of data. These are unfortunately called 'methods' in the built-in software. For clarity, this investigation will refer to these methods as *sample sets*.

2. *Odorant:* Each of the sample sets contains readings from 6 different chemical odorants. These odorants are referred to as chemicals, odors, and odorants interchangeably in this thesis.

3. *Data Vector:* For each chemical in each sample set, there are at most 10 exposures to the odorant. These exposures are called *data vectors*. The term data vector is commonly used with respect to the ordered set of numbers used as the input pattern presented to a neural network.

4. *Resistance values:* Each data vector consists of 32 resistance readings from the 32 sensors in the electronic nose.

An unlimited number of sample sets can be stored on computer disk for future reference. Sample sets are imported for use in the computer software and hand-held unit. This allows for a greatly increased sensing "library" for future sample identification.

CHAPTER III

ELECTRONIC NOSE DATA COLLECTION AND TRAINING

Determination of Chemical Odorant Samples

The odorant samples used for this series of experiments were supplied by GLCC Co., a flavor house located in Paw Paw, MI. The company supplied six odorant samples that were as different from each other as possible. The samples included allyl caproate, a pineapple-like odor; methyl salicylate, a "liniment" odor; isoamyl acetate, candy banana odor; myrcene, tropical fruity, mango-like odor; decanal, a powerful component of the orange peel aroma; and diacetyl, a powerful butter odor. The odorant chemicals were first diluted at 5% by weight in 95% grain alcohol. Those solutions were then diluted with water resulting in odorant concentrations of approximately 150 ppm. These chemicals are extremely strong in their original concentrations and are typically diluted by these amounts in normal use. The samples were put into 2 oz glass bottles, 70 gm per bottle. This left a headspace of about an inch in which the volatile odor chemicals could accumulate prior to sampling with the nose.

Electronic Nose Data Collection

Prior to the collection of data, the sampling and data processing parameters were pre-set through the software supplied with the nose. These parameters included flow settings for the actual sampling pump, digital filtering, substrate temperature control, choice of sensor activity, algorithm choice, preprocessing, normalization type, and identification quality. Choice of algorithm, preprocessing, normalization, and

identification quality can be altered after all sampling has been completed during the fine-tuning of the identification and clustering process.

Flow settings for the sampling pump require fine-tuning in order to achieve the best performance. The Cyranose operating manual includes examples of different sampling experiments for different substrates. An initial starting point was chosen from the example that most closely matched this situation, an experiment to identify a sample that is one of three possible liquid fragrances. The flow settings were adjusted during early sampling tests to optimize the method. Figure 4 shows the flow setting screen where adjustments are made. Table 1 indicates the starting flow setting and data processing settings.

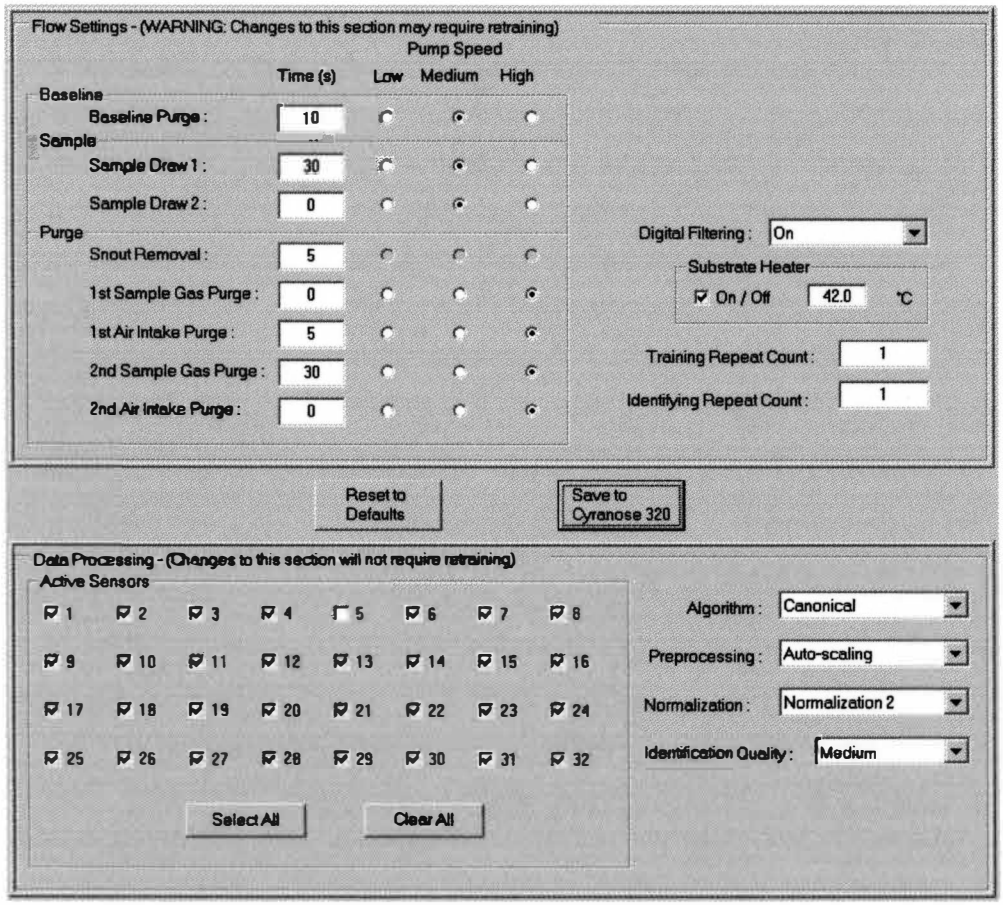


Figure 4: E-nose Method Setting Screen

Flow Settings		
	Time (sec)	Pump Speed
Baseline Purge	10	Medium
Sample Draw	6	Medium
Sample Draw 2	0	Medium
Snout Removal	0	
1st sample gas purge	0	High
1st air intake purge	5	High
2nd sample gas purge	30	High
2nd air intake purge	0	High
Digital Filtering	On	
Substrate heater	On	42
Training repeat count	1	
Identifying repeat count	1	
Data Processing		
Active Sensors	All	
Algorithm	Kmeans	
Preprocessing	Mean-centering	
Normalization	Normalization 1	

Table 1: Initial Flow Settings from Cyranose Operating Manual

Sampling procedures were also developed. These required knowledge of the odorant chemicals and how they behave in closed containers. The sampling procedure adjustments included varying the amount of time allowed for the sample to rest and re-equilibrate between sampling, sampling order, and sample concentration. The sampling was done randomly among the sets to eliminate any sampling bias. Using the odorant

samples described above, the Cyranose 320™ electronic nose was used to collect experimental samples. Five sample sets of sensor data for six chemicals, for a total of 300 data vectors, were collected. Once this was completed, training the nose using the chemometrics software built into the electronic nose could begin. These results became the basis to which the clustering effectiveness of the neural network was compared.

Training the Electronic Nose

Pattern recognition software is part of the Cyranose 320™ system. The software operates through both the hand-held unit and a PC. Pattern recognition consists of two phases: (1) training and clustering using the sensor data and (2) identification of unknowns. Only the first phase was used for this investigation. To run the first step in the pattern recognition operation, the cross-validation operation is selected from the options on the hand-held unit. This operation consists of several steps. The unit performs pre-processing and normalization routines, runs PCA to reduce the dimensionality of the system and detect outliers, and runs the cross-validation check using the modeling algorithm. The internal cross-validation results could be viewed on the hand-held unit. These results indicate the number of correct identifications within the sample set. More extensive results, including details of the cross-validation, Mahalanobis distances (for CDA only), PCA plot, smell prints, distance vectors, and Canonical plots can be viewed on the PC. A flowchart for training the nose can be seen in Figure 5.

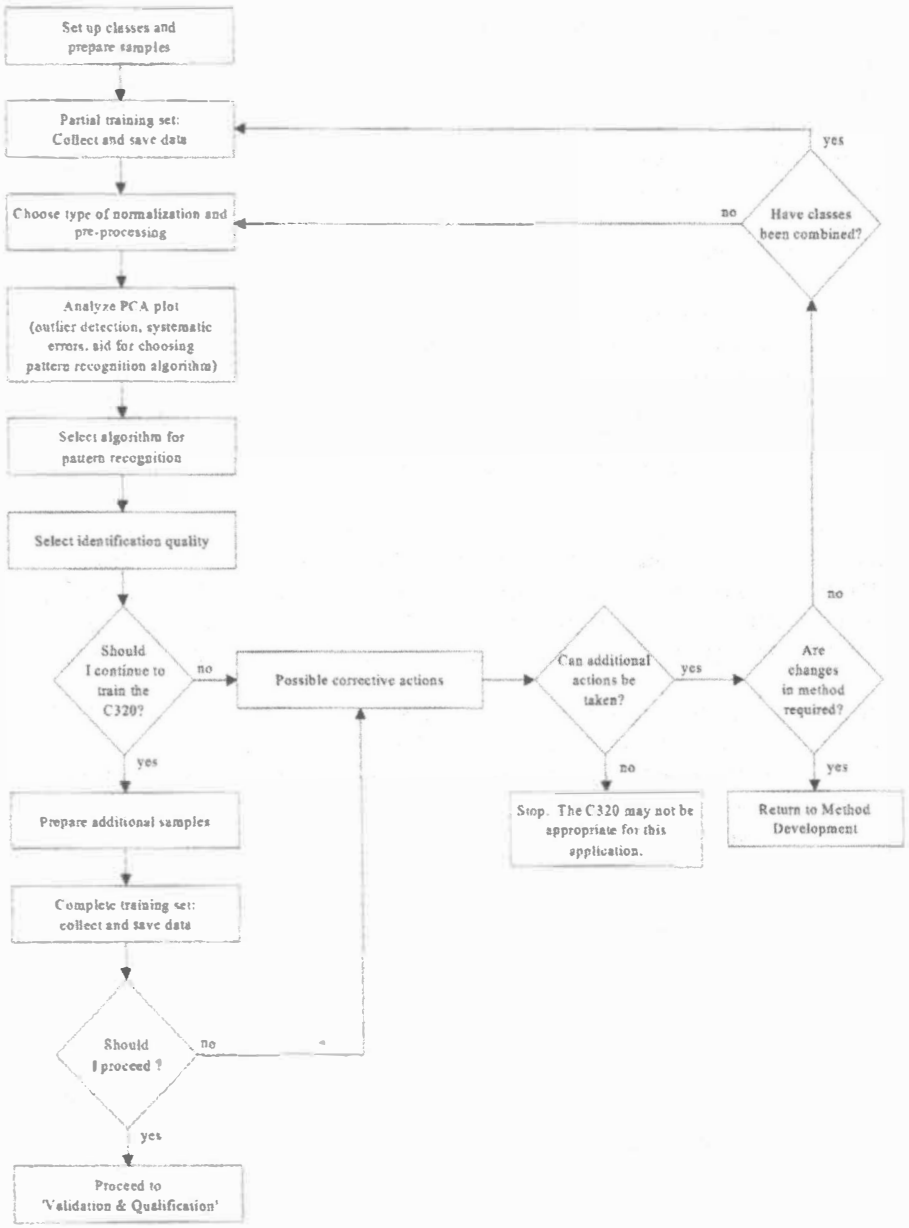


Figure 5: Flow Chart for Training the Cyranose 320™, [Cy2: p56]

Training the nose required considerable re-sampling after the initial 60 data vectors for each sample set were obtained. Each time the pattern recognition algorithm was run, the cross-validation was viewed. If the internal identification of the samples was not 100%, outliers were identified, removed, and re-sampling done. Outliers could be identified several ways. There may have been one or more points that were significantly different from the others. These could be seen as obvious outliers on the PCA plot or as points that had Euclidian distances much larger than the other samples in the class. This testing, resampling, and retesting continued until a 100% cross-validation was achieved for all five sample sets.

During the pattern recognition phase, fine-tuning of the pre-processing, normalization, and algorithm was also performed. The final parameters are shown in figure 3, previously mentioned. Possible options for normalization were no normalization, 1-norm, and 2-norm. After much testing using the different normalization options, the best results for this application were obtained using 2-norm. It is calculated by dividing each value in the data vector by the 2-norm of the set, shown below.

$$[3.1]\quad \text{2-norm} = \sqrt{\sum_{j=1}^{n} x_j^2} \quad \text{where } x_j = \text{input variable value}$$

Variable pre-processing was the second data preparation step. Options included no pre-processing, mean centering, and auto-centering. As with normalization, testing using all pre-processing methods showed that auto-centering worked best in this situation. Auto-centered values were calculated by dividing each value in the data column by the standard deviation of that column shown below.

$$[3.2]\ x_j' = \frac{x_j}{\sqrt{\sum(x_i - \mathbf{x})^2/n}}$$

where x_j = the initial variable value
x_j' = the auto-centered value
$\mathbf{x}$ = mean of the variables
n = total number of variables
x_i = each variable value in the set

The three algorithms available to build the pattern recognition model included KNN (K Nearest Neighbor), K-means, and Canonical (Canonical Discriminate Analysis-CDA). During the sampling, training, and analysis period, all algorithms were tested in order to determine the most effective clustering algorithm for the data. This algorithm turned out to be the Canonical algorithm. In CDA, "the total variation between the objects can be partitioned into 1) the variation due to the differences between the groups or 2) the variation within the groups, due to the differences between individuals." [BEG] In the CDA, an unknown sample is assigned to the class with the shortest Mahalanobis distance between its centroid (the point equidistant from the points already assigned to that cluster) and the sample in canonical space. [Cyl] (Refer to Appendix B for detailed chemometric information.)

Once the training was complete, the results could be viewed in several formats. The cross-validation screen showed the results of the internal cross-validation and measurement of Interclass Mahalanobis distances (M-distances). The cross-validation table shows the number of samples that were correctly identified in the internal comparison. The Mahalanobis distances look at the variance between the responses of the data vectors for the same sensor but also the inter-sensor variations (co-variance). [Th] The PCA plot indicates whether outliers exist. Once a 100% cross-validation was

achieved, further fine-tuning could still be done by improving the interclass M-distances through additional re-sampling. The CDA plot showed the final clustering results for the pattern recognition. The closer together the individual data vectors were for each chemical, the better the clustering capability was. Refer to figures 6 and 7 for examples of the Cross-validation screen and Canonical plot provided by the Cyranose software.

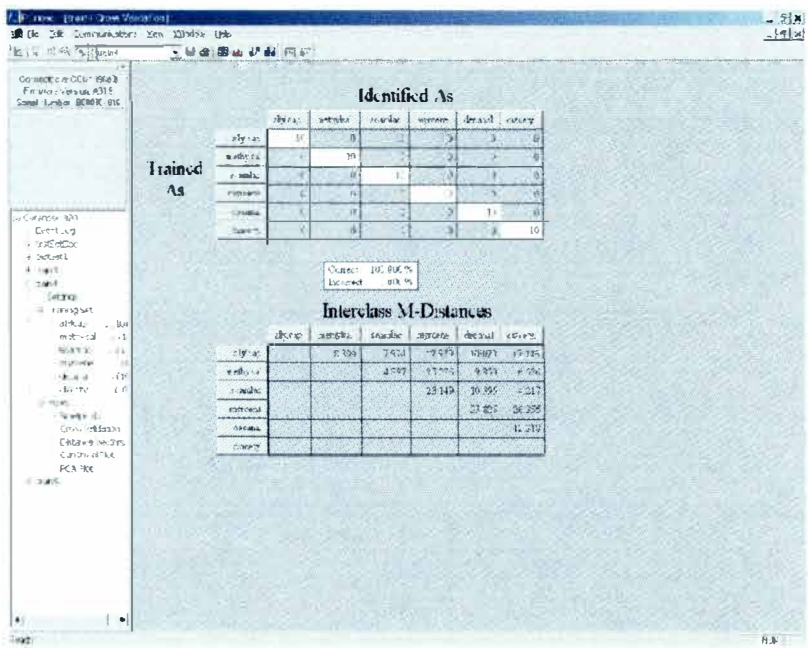


Figure 6: Cross Validation Table

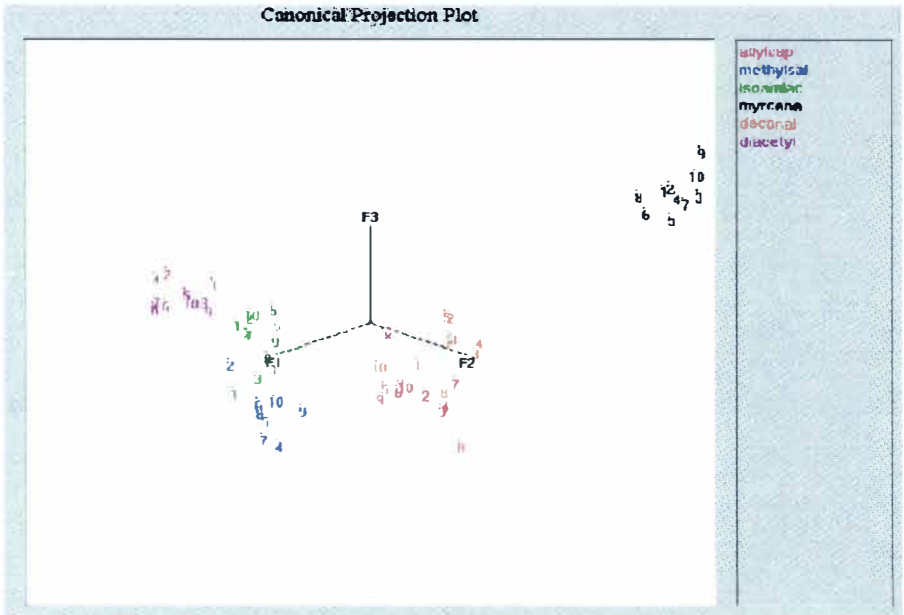


Figure 7: Canonical Projection Plot

The initial step in pattern recognition by the Cyranose unit was now complete. The resultant plots indicate the best clustering capabilities of the built-in pattern recognition algorithms of the unit based on the data collected. The aim of the pattern recognition step was to separate the data from the six chemicals into six distinct clusters such that identification of unknowns would be possible. These results for the baseline against which the subsequent neural network pattern recognition approach could be compared.

CHAPTER IV

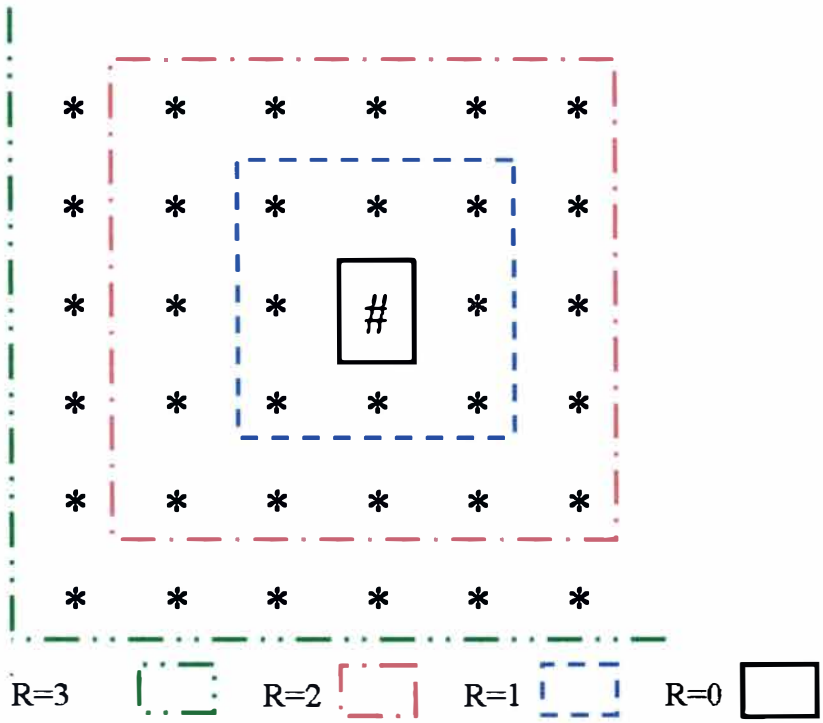
OVERVIEW OF KOHONEN ALGORITHM ADAPTATION

A neural network library called "Monarch", (available in the Computer Science Department at Western Michigan University as a result of a master's thesis [Fr]) was used as the base neural network library for this project. After considerable experimentation and adaptation, the final neural network program was run on the data collected using the Cyranose unit.

Initially, the Kohonen algorithm was implemented at a very elementary level with only two output clusters available and a neighborhood consisting only of the winning cluster. For this application, it was decided that the network would have better discriminating power if the neighborhood weights were updated as well as the winner. During the first several tests, the output cluster array was set up as a 6x6 two-dimensional array.

Numerous data preparation techniques were tested. The pre-processing technique initially used was a simple sample normalization procedure. As experimentation progressed variable pre-processing was added. The most effective pre-processing turned out to be the same as those used for the chemometrics-based system connected with the nose. These were 2-norm normalization sample pre-processing and auto-centering variable pre-processing.

As previously noted, the data samples collected with the Cyranose unit were used as input data to the neural network. Five sample sets each containing 60 data vectors were available for training the network. For a majority of the tests, sample set 2 was used as the input neural network data sample. The concept of neighborhoods was incorporated into the network. With each presentation of data, the winning cluster unit was updated with output cluster units in the neighborhood around the winner updated to a lesser degree. The largest neighborhood in the network (3 in this case) had a radius of the ceiling of the square root of the number of chemicals to be identified. As training progressed, the maximum size of the neighborhood was gradually decreased until, as the stopping condition was approached, only the weight matrix of the winning cluster was updated. (Refer to figure 8). Because the initial radius of the neighborhood is half the size of a side of the array, the outer neighborhood is incomplete. Neighborhoods do not "wrap around" the array. Missing units are simply ignored. As experimentation continued, it was decided to change the cluster array to a three-dimensional array to make the comparison between the two pattern recognition techniques more accurate. As a result, the neighborhood around the winning cluster changed from a two dimensional ring of neighborhoods to a more complex three-dimensional neighborhood.



Winning cluster

* Other clusters in output array

R Rings around the winning cluster, number indicates distance from winner

Figure 8: Two-dimensional Neighborhoods for E-nose Kohonen Network

The stopping condition for network training is chosen to be when the learning rate has decreased to a predetermined level. Each time the network completes an epoch (presentation of the entire data sample); the learning rate is decreased geometrically. In this case, the decrease ranged between 25-50%. Initially, the network weights were set randomly and were in the same numerical range as the normalized and pre-processed data. Ultimately, in a final attempt to improve the clustering ability, some of the weights were set equal to the averages of the known data from the electronic nose sensors, creating a semi-supervised neural network environment. Different arrangements of

designated winning clusters were tested to see if greater discriminating ability could be obtained. Different weight update rates were also tested.

All the data collected from the electronic nose was run through the semi-supervised neural network. The results were plotted using Maple™ and compared to the clustering results from the electronic nose software.

In order to ensure that the network was accomplishing real clustering and not merely accidentally making some data points appear to be clustered together, completely random data was also run through the network.

CHAPTER V

DETAILS OF NEURAL NETWORK TRAINING

Preparation of NN Input Data Files

Prior to training the Kohonen network, the input data sample files from the electronic nose were converted into a file format convenient for use as input to the neural network. The input data files for the electronic nose are called "Method setting files" (.met) and included all the information needed to re-import the data into the nose and run sample identification testing using that sample set. An example of part of a .met data file can be viewed in Figure 9. The data stored in the method files had already been filtered and reduced by the Cyranose unit. The data was in the form of the response of the electronic nose sensors, defined in the following equation and was used as input to the pattern recognition algorithms:

$$[5.1]\ \Delta R/R_o = (R_{max} - R_o)/R_o$$

where ΔR= change in sensor resistance

R_o= average of base resistance calculated by taking 5 data points before sample exposure and 5 from the end of the purge step in the sampling
R_{max}= average maximum resistance

```
Method name=      train1

Class 1=    allylcap
Class 2=    methylsal
Class 3=    isoamylac
Class 4=    myrcene
Class 5=    decanal
Class 6=    diacetyl

Baseline purge=   10s   medium
Sample draw 1=    30s   medium
Sample draw 2=    0s    medium
Snout removal=    5s    low
1st sample gas purge=   0s    high
1st air intake purge=   5s    high
2nd sample gas purge=   30s   high
2nd air intake purge=   0s    high

Digital filtering=      On
Substrate heater= On    42.0
Training repeat count=  1
Identifying repeat count=     1

Active sensors=   FFFFFFEF

Algorithm=  Canonical
Preprocessing=    Auto-scaling
Normalization=    Normalization 2
Identification quality= Medium

;  33x60
; ClassName S1    S2    S3    S4    S5*   S6    S7    S8    S9    S10   S11   S12   S13   S14   S15   S16   S17

Exposure=allylcap  0.003548    0.0034814   0.0030381   0.005742    0.0413538   0.0119697   0.0025728   0.0031912
Exposure=allylcap  0.0036885   0.0035311   0.0032782   0.0061794   0.0430255   0.0111514   0.002909    0.0036934
Exposure=allylcap  0.0029366   0.0026928   0.0024806   0.0046683   0.0335063   0.008302    0.0022241   0.0025074
Exposure=allylcap  0.0031367   0.0029554   0.0027131   0.0052553   0.037347    0.0093797   0.0025239   0.0029711
Exposure=allylcap  0.0029374   0.0029061   0.0024645   0.0048585   0.0339404   0.0070408   0.0022781   0.002576
Exposure=allylcap  0.003317    0.0030373   0.0026327   0.0054184   0.0328921   0.0068582   0.0024116   0.002995
Exposure=allylcap  0.0027332   0.0026241   0.0021719   0.0044553   0.0307178   0.0069942   0.0021882   0.0023862
Exposure=allylcap  0.0025085   0.0023145   0.0019654   0.0042303   0.0304681   0.006934    0.0020886   0.0021862
Exposure=allylcap  0.0040523   0.0038051   0.0034272   0.0066715   0.0408487   0.0080966   0.0023467   0.0032803
```

Figure 9: Example Method File from the Cyranose Unit

The conversion programs extracted the sensor response data from the '.met' files, performed various normalization and pre-processing routines and output the data to the neural network input data files. The final data conversion and pre-processing programs can be viewed in Appendix C.

Evolution of the Kohonen Neural Network Application

A series of experiments drove the development of the Kohonen neural network to cluster the electronic nose data. The aim of each succeeding experiment was determined based on the results of the preceding one. Sometimes, more than one avenue was investigated based on result of a particular experiment.

Test 1- Artificial Data

The first neural network experiments ran artificial data through a 2-dimensional Kohonen network which did not include updating of the neighborhoods. The data sample consisted of eight artificial data vectors each containing four "sensor" outputs. The initial pre-processing was normalization of the samples of data by the equation below:

$$[5.2]\ normx_i = (x_i - x_{min})/(x_{max}-x_{min})$$

where: x_i = each individual data value
x_{min} = the minimum sample data value
x_{max} = the maximum sample data value
$normx_i$ = normalized individual value

The normalized data represented four very different clusters with data points between 0 → 50. The data sample is shown in Table 2.

	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Data vector 1	17.7152	15.5059	17.2873	19.08797
Data vector 2	43.3402	49.954	46.6652	41.793
Data vector 3	0.38741	0.54312	0.36369	0.233522
Data vector 4	8.5227	10.3975	9.49619	7.72074
Data vector 5	50	50	50	50
Data vector 6	18.4488	19.1176	16.2311	17.50617
Data vector 7	10.3857	9.96675	9.26804	8.8177
Data vector 8	0	0	0	0

Table 2: Test 1 Data

Each time a data vector was presented to the network, a "winning" cluster was chosen as the output unit whose weight vector most closely matched the input data vector.

The weights of that unit were then updated to improve the match using the following equation:

[5.3] $w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$

where w_{ij} = weight on the input to output link

x_i = input value

α = the learning rate.

The network successfully clustered this data into four distinct clusters. The results can be seen in figure 10. The output below lists the integer values of the input data and the group in which the input was placed. The numbers indicating the output cluster indicate the row and column of the output cluster unit. The complete output from the program showed the clustering results after each epoch as the network gradually reduced the learning rate. With this artificial data, there were no changes in the clustering from the first to the last epoch indicating the network easily clustered the data from the first epoch.

```
---------- Alpha = 0.000003
18 16 17 19 -> [group: 3, 5]
43 50 47 42 -> [group: 1, 6]
0 1 0 0 -> [group: 1, 4]
9 10 9 8 -> [group: 1, 5]
50 50 50 50 -> [group: 1, 6]
18 19 16 18 -> [group: 3, 5]
10 10 9 9 -> [group: 1, 5]
0 0 0 0 -> [group: 1, 4]
```

Figure 10: Test 1 Results

Test 2- Subset of Actual Nose Data

The second test consisted of a data sample containing 18 sets of data, 3 for each chemical, 10 sensor readings for each data vector. The data was randomly mixed. This data sample came from actual Cyranose data.

At this point, the code for updating the neighborhood around the winner was added to the weight adjusting function. Updating the winner and the neighbors was accomplished in several stages. First, the winning cluster was determined. Then, a loop started which gradually updated clusters from the outer to the inner neighborhoods. The code ensures weights were not changed unless they actually belong to nodes in the neighborhood and are not a "wrap-around" node. The data structure of the output clusters is in the form of an array with output cluster units on different levels of the 2-dimensional arrangement being numbered consecutively. If a neighborhood is not complete, the code has to ensure that the next array index is not assumed to be in the incomplete cluster neighborhood. (Refer to figure 11). Finally, the winning cluster's weights were updated. The distance from the winner determines how much the weights are updated. The winner was updated using the equation [5.3]. The update equation for neighbors at distances of 'ℓ' from the winner is:

$$[5.4]\ w_{ij}(new) = w_{ij}(old) + (\alpha/\ell)[x_i - w_{ij}(old)]$$

where α = learning rate
ℓ = distance from winner

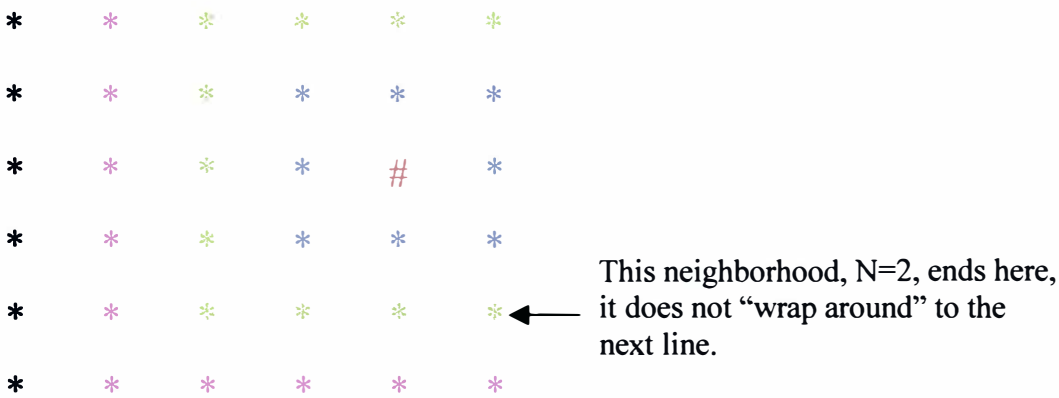


Figure 11: Neighborhoods End at Edge of Cluster Array

The second test included the above-mentioned code changes and corrections. The same sample set, 18 data vectors, three from each of 6 chemicals, was used. After each training epoch, the size of the neighborhood was reduced. As the network became more trained, each presentation of a data vector should have resulted in a winning cluster whose weight vector was closer and closer to the presented data vector. If this were the case, there would be less and less need to alter the weights of the neighboring clusters as well. Test 2 resulted in the classification of the data into 13 clusters. Refer to figure 12 for results. Despite correction of errors in the code, the network was still unable to correctly cluster the data.

---------- Alpha = 0.000003
7114636461271937 -> [group: 1, 4]
3323101653610 -> [group: 5, 6]
3423292873725 -> [group: 3, 3]
47241428641220 -> [group: 5, 2]
89681310981113 -> [group: 6, 1]
3322293072728 -> [group: 3, 5]
0100212620318 -> [group: 5, 1]
50505050505050505050 -> [group: 6, 5]
2211191641416 -> [group: 4, 6]
3322454182837 -> [group: 1, 4]
3513002380 -> [group: 6, 6]
2211233141521 -> [group: 2, 2]
00007140006 -> [group: 5, 6]
33333132142430303425 -> [group: 6, 5]
2211202741422 -> [group: 5, 5]
2311403851535 -> [group: 1, 4]
1211232851520 -> [group: 2, 2]
4323233263723 -> [group: 4, 3]

Cluster	[1,4]	[5,6]	[3,3]	[5,2]	[6,1]	[3,5]	[5,1]	[6,5]	[4,6]	[6,6]	[2,2]	[5,5]	[4,3]
# in group	3	2	1	1	1	1	1	2	1	1	2	1	1

Figure 12: Test 2 Results

When updating neighborhoods, it is possible to change not only the size of the neighborhood but also the magnitude of the changes to the weights in the neighborhoods. It was hoped this approach would improve the clustering. Several combinations were tested. None improved on the clustering capability. Therefore, it was decided to return to the original multipliers. Refer to table 3 for the multipliers tested.

Distance from winner	Initial Multiplier	Multiplier for Test 2a	Multiplier for Test 2b
R=0	α	α	α*100
R=1	α	$\alpha/2$	$\alpha/2$
R=2	$\alpha/2$	$\alpha/4$	$\alpha/4$
R=3	$\alpha/3$	$\alpha/6$	$\alpha/6$
Clusters in Result	13	15	13

Table 3: Test 2 Learning Rate Multiplier Table

Test 3- Additions to Aid Visualization of Clustering Results

Even though at this stage the network was not clustering well, it was decided to convert all five sample sets to the normalized format for running on the network. This normalization was the simple single normalization referred to previously. This resulted in six data files to run through the network. Several different normalization ranges were investigated to see which range worked best. It was decided to use a normalization of 0→1 at this stage. The data files were named method*x*.dat where *x* was a number 1→5 for the different sample sets.

Prior to running test 3, several other additions were made to the code. The initial clustering code indicated in which cluster a particular data vector was placed. If this were plotted, it would result in numerous data points at each cluster, making it impossible to see how many points were assigned to each cluster. In order to separate the data points, the error values calculated by the network were incorporated into the x and y coordinates of the winning cluster resulting in a new data point which was located a distance equal to the error value away from the winning cluster. A random angle was chosen to locate the

point around the output cluster. The resultant *x* and *y* coordinates artificially separate the data points around the winning cluster but greatly aid in visualizing the clustering results. See figure 13 for code to convert data points.

```
for(int j=0;j<N_TESTS;j++){
//calc normed err
err[j] = (err[j]-min)/(max-min);
//choose angle in radians
z = rand()%360;
a_rad[j] = z*PI/180;
// Calculation of plottable winning cluster coordinates
x[j] = x[j] + cos(a_rad[j])*err[j];
y[j] = y[j] + sin(a_rad[j]) * err[j];
//Output to data file
final_data<<x[j]<<" "<<y[j]<<" "<<chem[j]<<endl;
}
```

Figure 13: Output Cluster Data Point Conversion

Prior to this stage, no cross-referencing was done to determine if the clusters created by the network actually held data points from the same chemical samples. In order to make this comparison possible the program imported a file containing a list of the six chemical odorants used to create each data vector. This information was incorporated into the final data files which contained the x and y coordinates of each clustered data point and a number indicating to which chemical the point refers.

Test series 3 was the first run using the above-mentioned alterations. This series consisted of three runs of the neural network. Tests 3a and 3b used sample sets 1 and 2. Test 3c used all 300 data vectors from the five data samples.

The network clustered test3a (data sample 1) into 13 clusters. There did not appear to be a logical way to group the clusters into a smaller number of mega-clusters. There

was a large concentration of data points around [4,0] and [4,1], many more than would be expected if this consisted of two chemicals that were similar according to the network. Refer to figure 14 for a plot of the clustering results.

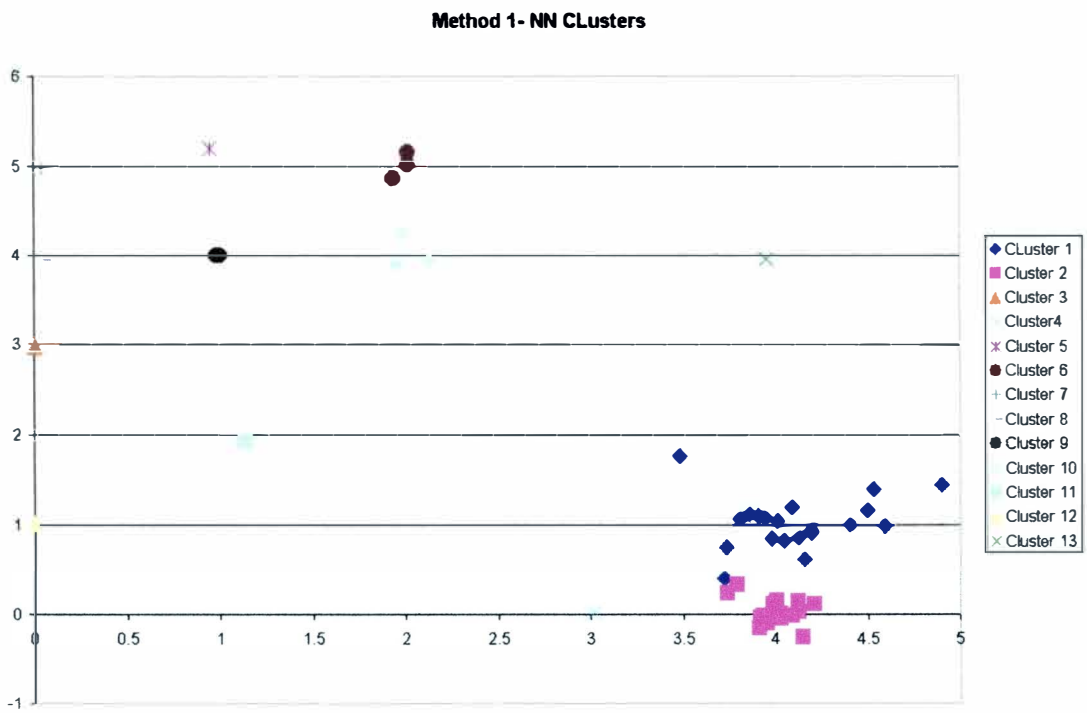


Figure 14: Test 3a Results by Cluster

When the same plot was examined with respect to the chemical odorants, there appeared to be little if any clustering of the individual chemicals. (See figure 15) Two chemicals, myrcene and decanal, appear loosely clustered away from the mass of data points in the lower right corner of the plot. It is unlikely that a statistical analysis of the results would confirm definite clusters. Test 3b produced similar results.

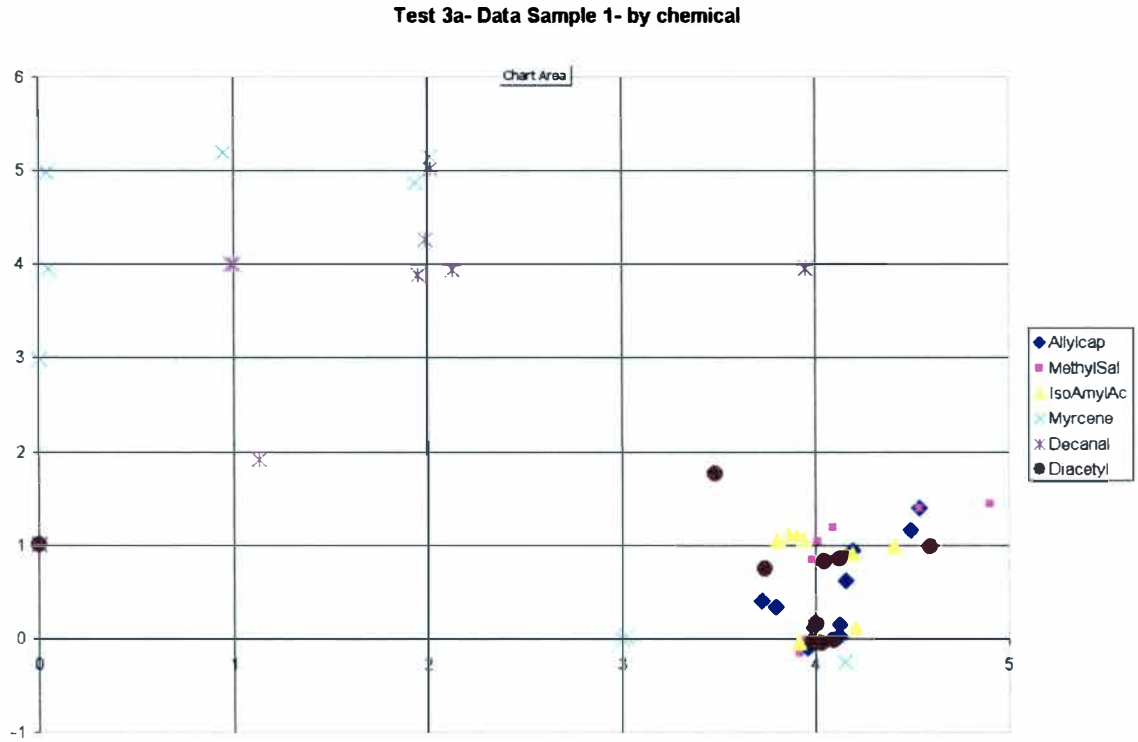


Figure 15: Test 3a- Results by Chemical Odorant

Test 3c, using the entire 300 set data sample, showed little improvement. While the imaginative eye might see approximately seven mega-clusters in the plot, the realistic clustering capabilities of the network were still marginal at best. There is simply too much overlapping of the clusters for any definite identification to occur. (Refer to figure 16)

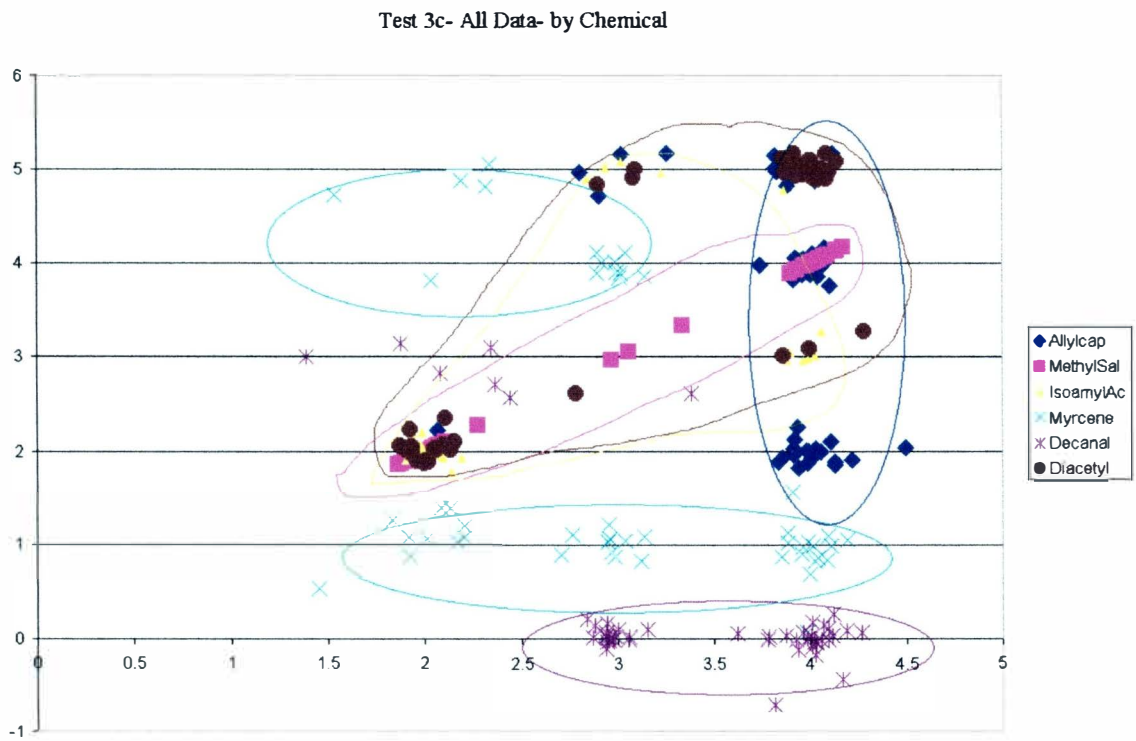


Figure 16: Test 3c- Clustering of 300 Data Vectors

Test 4- More Visualization Improvements

At this stage, an additional factor was added to make the visualization more realistic in the hope of improving the apparent clustering of the network. When calculating the 'x' and 'y' coordinate values for each data point, the cluster with the second lowest error value was used to determine the directional angle of the point from the center of the cluster. In other words, if the winning cluster for data point 1 was (4,1) and the cluster with the second lowest error was (5,3), then the new 'x' and 'y' values would be located at a distance of the minimum error towards (5,3) from (4,1). These code changes are shown in figure 17. Whilst it was not thought that this would improve

the actual clustering capabilities of the network, it was hoped it might improve the visualization of the results.

```
for(int j=0;j<N_TESTS;j++){
        //calc normed err
        err[j] = (err[j]-min)/(max-min);
        //determined directed angle
        rad[j] = angle_det(x[j],y[j],x2[j],y2[j]);
        x[j] = x[j] + cos(a_rad[j])*err[j];
        y[j] = y[j] + sin(a_rad[j]) * err[j];
        //Output to data file
        final_data<<x[j]<<" "<<y[j]<<" "<<chem[j]<<endl;
}
```

where (x[j],y[j]) = 'winning' cluster
(x2[j],y2[j]) = second place cluster.

Figure 17: Calculation of Directional Plotable Data Points

Test series 4 incorporated these directional adjustments to the resultant coordinate points of the winning clusters for each of the data vectors. The directed clustering resulted in tighter concentration of data points around the cluster centroid, but no real improvement in the apparent clustering capabilities. Plots of test 4a and 4b using data sample 1 can be seen in Figure 18: "Randomly Assigned Clustering" and Figure 19: "Directed Clustering". While directed clustering appears to improve the appearance of the graph, it doesn't improve the clustering done by the network. It will be retained, as it does appear to reduce the randomness of the resultant graphs.

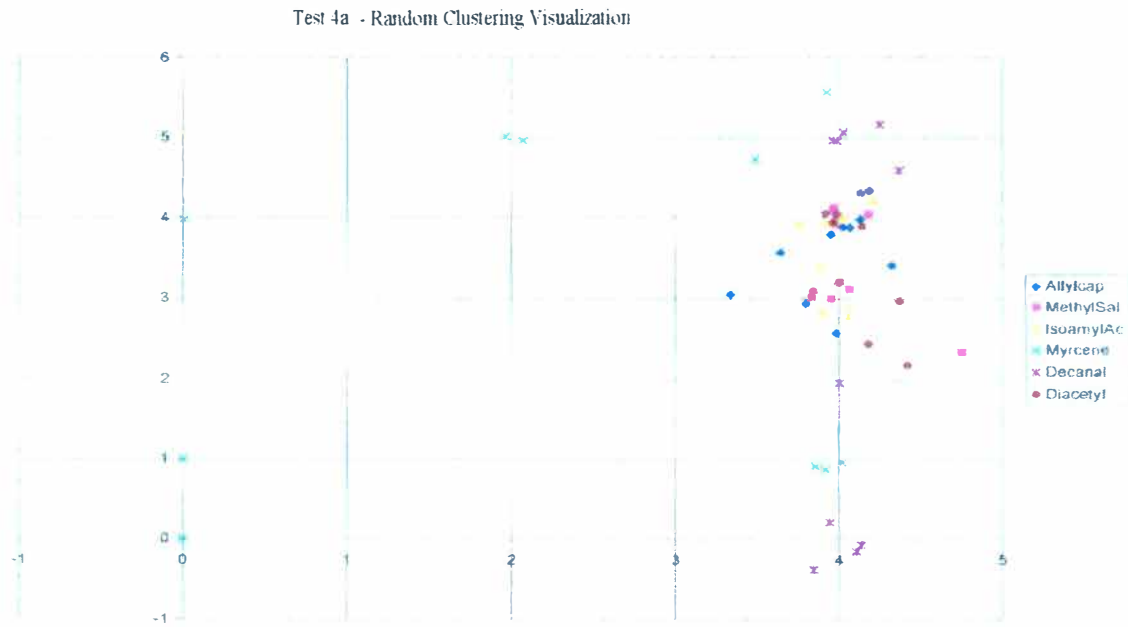


Figure 18: Test 4a Results- Randomly Assigned Clustering

Figure 19: Test 4b Results- Directed Clustering

Test 5- Higher Learning Rate

Thus far, none of the fine-tuning changes had improved the clustering to any great extent. Another avenue to investigate for potential improvement was to increase the value of alpha, the learning rate, and slow the speed at which the learning rate was decreased. If the learning rate were higher, the network would go through more epochs before it quit. Slowing the rate of the decline of alpha would also increase the amount of time the network has to learn to classify the data. In test 5, alpha was increased from 0.9 to 1.9 and the rate of decrease was cut in half. Thus after each epoch, the learning rate was decreased by one quarter. Both changes resulted in the network working longer on the training stage before the stopping condition was reached. The results, which are shown in Figure 20, indicated little improvement. The data points appeared more closely grouped around the cluster centroids, indicating a tighter error range, but the clustering of the different chemical odorants was no better than before.

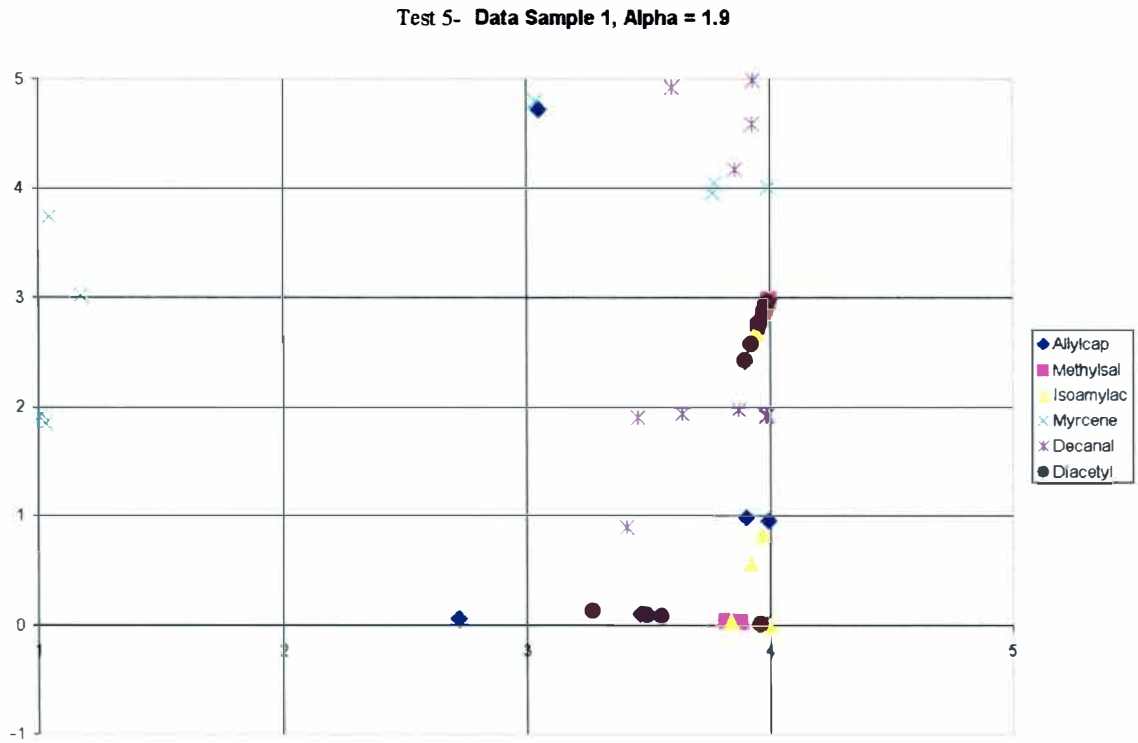


Figure 20: Test 5 Results- Higher Learning Rate

Test 6- 3-Dimensional Output Clustering

At this stage, it was decided to try to more closely mimic the plotting used by the Cyranose unit. Specifically, the Cyranose clustering algorithms displayed the clustering results in a 3-D format. It was hoped that with a 3-D format, the capabilities of the neural network would be improved. Major changes were made to the Kohonen code as well as additions to the main neural network library to allow incorporation of the three dimensional cubical output matrix where the length of the side equaled the number of different odorant chemicals used in the data. The weight updating step was changed to take into account the 3-dimensional neighborhood. (Refer to Appendix D: "Kohonen

Source Code" and Appendix E: "Neural Network Driver Code" to see source code.) The output from the network now included a 3-D coordinate (x,y,z) and the chemical identifier for each data vector presented to the network.

Test 6 was a 3-D neural network run. Whereas the change to 3-D did appear to spread out the data points, there was no improvement in the clustering results. For test 6, the plotted results were spread over three 'z' layers. Two of the chemicals, myrcene and decanal, were located on the z=0 and z=1 planes. The other four chemicals were widely scattered on z=2. There appeared to be no pattern to the arrangement. (Refer to Figure 21)

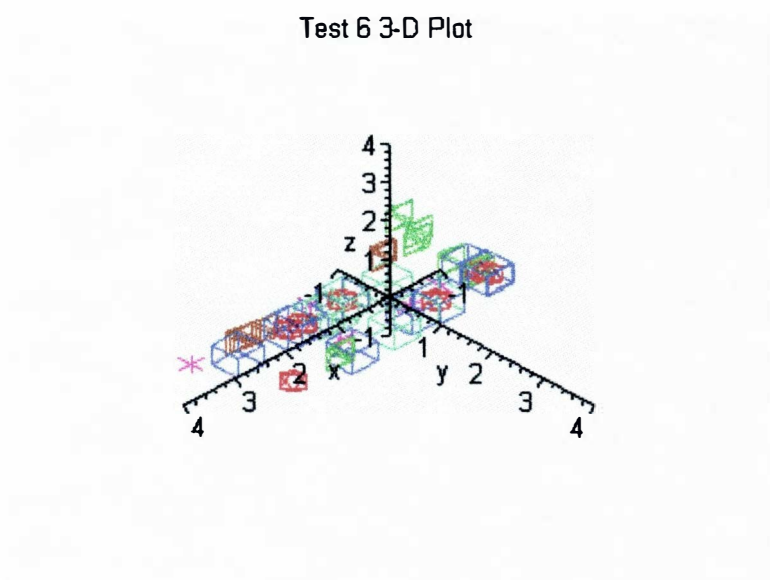


Figure 21: Test 6, Three-Dimensional Clustering

Test 7- Semi-supervision of Kohonen Algorithm

Unfortunately, up to this point the unsupervised Kohonen network simply was not able to cluster the data to any useful degree. A review of the clustering algorithms used by the chemometrics system on the Cyranose unit indicated that all were supervised algorithms. The question now was to determine if there were any ways to make the Kohonen algorithm semi-supervised, using the sensor data as a training reference. It was mentioned in Fausett [p172] that, whereas link weights are often assigned randomly, if knowledge of the distribution of the clusters in the specific problem is known, it may be appropriate to use this knowledge in setting the weights. It may appear that the network is being given "the answers". On the contrary, the purpose is that same as with a supervised neural network, supervised chemometrics pattern recognition algorithm, or many forms of human learning. The a priori incorporated into the weights guides the learning but does not control it. Ideally, the result will be a network with weights set such that unknown samples can be properly identified.

In this case, the data from the nose contained information about the identity of all data vectors. If the input data samples were separated for each chemical and averages of each sensor reading for each chemical were calculated, those averages could be assigned as the initial weights to specific clusters in the output matrix. Six specific output clusters were designated as the "winners" and roughly corresponded to the locations in the three-dimensional clustering diagrams produced by the Cyranose clustering diagrams. The Cyranose clustering diagrams produced clusters for all the chemicals that were in roughly similar locations for all tests. It was thought that by choosing similar winning cluster

locations, the comparison between the methods could be more precise. This method was termed "semi-supervised Kohonen clustering". In the figure below (Figure 22), the calculations of average sensor values for one chemical are shown. These averages correspond to the weight variable names listed below each average. Allyl Caproate was designated as chemical one and 'j' corresponds to a particular output cluster in the network.

		S1	S2	S3	S4	S5	...	S31	S32
Allylcaproate	Data vector 1	2.02114	2.27287	1.22585	1.56243	4.66837		4.57156	1.87565
	Data vector 2	2.20283	2.42379	1.23985	1.71518	4.90609		4.35008	2.75209
	Data vector 3	2.18628	2.4789	1.29925	1.7273	5.00605		4.35627	2.27012
	Data vector 4	2.0699	2.18876	1.1487	1.66736	4.98214		4.38528	2.58351
	Data vector 5	2.52311	2.71523	1.5115	1.98418	5.05256		3.93106	2.15884
	Data vector 6	2.45053	2.57163	1.37765	1.9121	4.81552		3.88237	3.22489
	Data vector 7	2.02982	2.22701	1.2778	1.62434	4.69212		4.35678	2.20155
	Data vector 8	2.02668	2.1906	1.24287	1.62353	4.78665		4.54783	2.4672
	Data vector 9	2.14477	2.25396	1.28326	1.62861	4.84952		4.67096	1.92899
	Data vector 10	2.23785	2.47987	1.33289	1.84262	5.00597		3.82943	2.71731
	Average	2.189491	2.380262	1.293962	1.728765	4.876499		4.288162	2.418015
		w_{1j}	w_{2j}	w_{3j}	w_{4j}	w_{5j}	...	w_{31j}	w_{32j}

Figure 22: Calculation of Semi-Supervised Cluster Weights for 1 Chemical

By forcing the weights of six clusters to correspond to the averages of the six input data subsets (one subset for each chemical), it was hoped that this a priori knowledge would aid in the clustering ability of the network.

In addition to forcing the weights in the network to specific values, it was decided to change the pre-processing of the data so that the same pre-processing methods were used for the neural network data as were used by the Cyranose chemometrics system.

This required incorporating functions for a 2-normalization procedure and an auto-centering procedure to be applied to all the data samples.

In the driver program, the Kohonen network was instantiated with random weights. To incorporate the "semi-supervision", a file was imported into the program containing the average weights for the data vectors from each chemical odorant. The average weight files were calculated for each of the five data samples collected from the electronic nose. These values were then incorporated into the weight matrix in positions corresponding to six pre-determined output clusters. The network was then ready for training. This use of a priori knowledge in the training of the network is similar to the internal cross-validation used in the Cyranose pattern recognition in that knowledge of the identity of each data vector allows a determination if the clustering is effective.

Test series 7 incorporated semi-supervision of the Kohonen algorithm and 2-normalization but not auto-scaling of incoming data, and a random assignment of target clusters. The target clusters in the network were: allyl caproate- [0,0,0]; methyl salicylate- [4,4,1]; isoamyl acetate- [2,3,2]; myrcene- [3,2,3]; decanal- [3,1,4] ; and diacetyl- [5,5,5]. The results from the best of these runs are shown in figure 23.

Clustering results for network run
tally[0][64] = 10
Network clustered 10 input vectors of Allylcaproate to cluster (4,4,1)

tally[1][153] = 7
Network clustered 7 input vectors of Methylsalicilate to cluster (3,1,4)

tally[2][153] = 5
Network clustered 5 input vectors of Isoamylacetate to cluster (3,1,4)

tally[3][73] = 8
Network clustered 8 input vectors of Myrcene to cluster (1,3,2)

tally[4][126] = 10
Network clustered 10 input vectors of Decanal to cluster (2,2,3)

tally[5][152] = 5
Network clustered 7 input vectors of Diacetyl to cluster (2,1,4)

Figure 23: Test 7 Results

As can be seen from the above results, there was some improvement in the clustering capabilities of the network. The network assigned at least half of the data vectors of three of the chemical odorants to clusters that had been designated as target clusters. The target clusters were not always the clusters to which the chemicals had been pre-assigned. For example, all the allyl caproate samples were assigned to cluster [4,4,1] which had initial weight values equal to the averages of the methyl salicylate samples. This could indicate a greater similarity between the weights of all the data vectors than to any of the randomly assigned weight sets. That the assignments no longer appeared completely random appeared to be progress but the network was still not working in the desired manner.

Test 8- Incorporation of Auto-scaling of Input Data

The next series of test runs incorporated auto-scaling of the data samples. At this point, the preparation of the data was identical to that used by the Cyranose system. The results of test run 8 are shown below in figure 24.

```
Clustering results for network run
tally[0][0] = 7
tally[0][153] = 1
tally[0][154] = 2
max = 7 index = 0 i = 0
Network clustered 7 input vectors of Allylcaproate to cluster (0,0,0)

tally[1][64] = 1
tally[1][65] = 4
tally[1][122] = 3
tally[1][153] = 1
max = 4 index = 65 i = 1
Network clustered 4 input vectors of Methylsalicilate to cluster (5,4,1)

tally[2][64] = 3
tally[2][65] = 6
tally[2][122] = 1
max = 6 index = 65 i = 2
Network clustered 6 input vectors of Isoamylacetate to cluster (5,4,1)

tally[3][214] = 6
tally[3][215] = 4
max = 6 index = 214 i = 3
Network clustered 6 input vectors of Myrcene to cluster (4,5,5)

tally[4][92] = 3
tally[4][93] = 6
max = 6 index = 93 i = 4
Network clustered 6 input vectors of Decanal to cluster (3,3,2)

tally[5][64] = 5
tally[5][65] = 5
max = 5 index = 64 i = 5
Network clustered 5 input vectors of Diacetyl to cluster (4,4,1)
```

Figure 24: Test 8 Results

The above results showed slight improvement. Clearly, the data pre-processing steps were helpful. In this test, the network assigned seven data vectors of allyl caproate to cluster [0,0,0], which was the cluster assigned to that chemical. The remaining 3 data

points were assigned to points far from the [0,0,0] cluster and were clearly incorrectly assigned. Half of the diacetyl data vectors were assigned to cluster [4,4,1] which, although not the designated diacetyl cluster was one of the designated clusters. The remaining 5 data points were assigned to cluster [5,4,1], right next to [4,4,1]. This essentially creates one large cluster. Six of each myrcene and decanal data vectors were assigned to clusters within one space of a designated cluster with their remaining data points assigned right next to the major cluster, thus creating two more multiple point clusters. Two chemicals, methyl calculate and isoamyl acetate, had a majority of their data vectors assigned to the same cluster, which was also only one value away from a designated cluster, cluster [4,4,1], to which the diacetyl was assigned. In effect, the network created 4 clusters, three of which contained individual chemicals. The fourth cluster, that consisting of points [4,4,1] and [5,4,1] grouped three chemicals together, essentially not being able to differentiate between the chemicals. Using these results, the network successfully clustered 36 out of 60 data points, a validation percentage of 60%, with some difficulty differentiating two of the chemicals from a third one. These were the best results thus far.

Test 9- Corners Designated as Winners

Clearly, whilst improvements were being made at increasing the clustering capabilities of the network, it was significantly less able to differentiate between the samples than was the Cyranose chemometrics system. Several others avenues were pursued in an attempt to improve the neural network capabilities. The first avenue was to

try to separate the designated clusters as much as possible in the 3-dimensional output arrangement. Six corners of the cluster cube were designated as "winners", each assigned to a different chemical odorant. The assignments can be seen in the following figure.

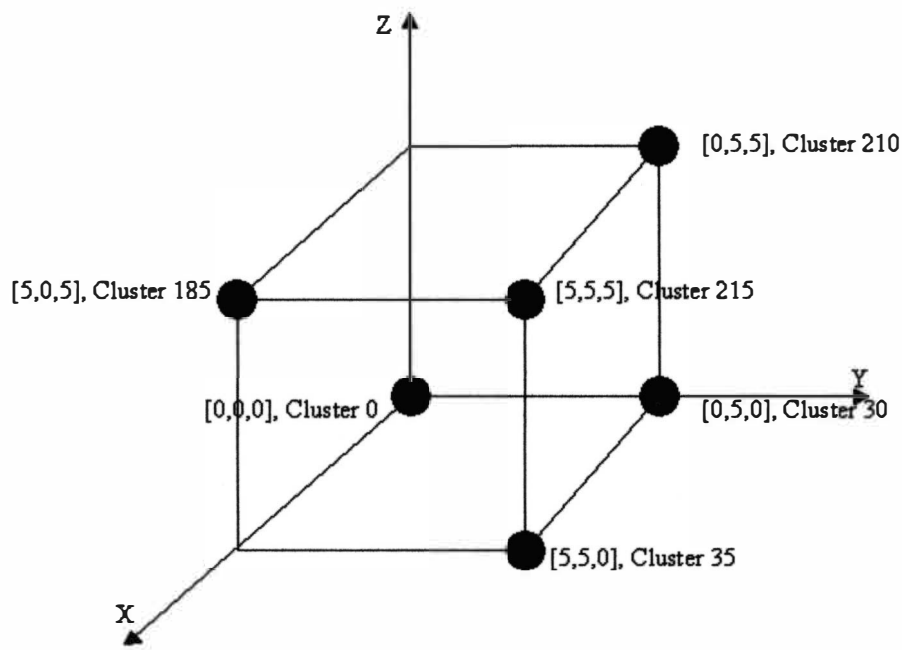


Figure 25: Designated Clusters for Test Run 9

Test 9 was run 10 times. The results are listed in table 4. Results indicate the winning cluster index and in parenthesis, the number of data points assigned to that cluster. A winning cluster is designated as the first cluster to which the greatest number of data points for a chemical has been assigned. The efficiency value in the table is a measure of the total number of data points assigned to a distinct cluster. Assigning the corners to be the designated winning clusters seemed to have the effect of pulling the

actual winning clusters toward the corners. However this still did not seem to either cause a majority of the winning clusters to end up at the corners, nor spread out the different chemical clusters to allow better differentiation.

Chemical	Run 9a	9b	9c	9d	9e	9f	9g	9h	9i	9j
AllylCaproate	33(4)	209(6)	184(9)	184(5)	209(9)	29(9)	184(7)	31(4)	184(10)	184(9)
MethylSal	212(4)	26(5)	28(6)	208(5)	28(6)	208(7)	25(4)	29(5)	184(9)	34(6)
Iso Amyl	208(4)	26(7)	28(9)	208(9)	28(6)	209(8)	27(6)	29(9)	184(7)	34(9)
Myrcene	28(6)	182(6)	213(10	200(6)	214(10)	181(4)	206(6)	33(10)	207(6)	28(10)
Decanal	184(5)	34(8)	208(8)	202(5)	213(6)	206(6)	208(9)	33(8)	34(7)	28(9)
Diacetyl	207(5)	27(9)	28(10)	208(10)	183(8)	209(10)	27(9)	29(10)	209(9)	34(10)
Efficiency	46%	60%	62%	43%	65%	60%	58%	40%	53%	48%

Table 4: Test 9 Results

Test 10- Fix Designated Output Cluster Weights

The next possible area for improvement was that of fixing the link weights of the six designated winning units at the start of the neural network run, in other words, not updating them at all throughout the run. The theory with this test was that the ideal weights of the winning units should be very close to the averages of the input sets that belong in that cluster. It was thought that holding the weights to the average input values would force the correct inputs to the appropriate clusters. These tests were run with the designated winning clusters being assigned to the original units from test runs seven and eight. (Refer to figures 23 and 24) Whilst the winning weights were not altered, the

weights in the neighborhood were updated using the equation [5.5]. (Refer to Figure 26 for Test Run 10 results.)

	Run 10a		Run 10b		Run 10c	
	Cluster	# in cluster	Cluster	# in cluster	Cluster	# in cluster
Allyl Caproate	0	6	0	6	0	6
	153	4	153	4	92	4
Methyl Salicylate	0	3	0	3	0	3
	153	6	153	5	92	6
			152	1		
Isoamyl Acetate	0	1	0	1	0	1
	153	9	153	6	92	9
			152	3		
Myrcene	0	5	0	2	0	5
	153	5	153	4	92	5
			152	4		
Decanal	0	3	0	3	0	3
	153	6	153	6	92	6
Diacetyl	0	0	0	0	0	0
	153	10	153	2	92	10
			152	8		

Figure 26: Test 10 Results

As can be seen from the results above, when the neighborhood was updated but not the winner, the network was unable to discriminate between any of the chemical odorants. At most, data vectors were assigned to three clusters with significant overlap between the clusters. This avenue of investigation appeared to be a dead end.

Test 11- Neighborhood Weight Adjustment Multiplier

The results from test 10 did suggest that the degree to which the winner and its neighborhood were updated might not have been sufficiently investigated previously. In test 11, it was decided to try a different multiplier from those previously investigated in

test 2. For this test, the amount the weights were adjusted was proportional to the distance from the winner with the winner being assigned the distance of 1 instead of 0, which was the previous practice. This avoided division by zero and the possibility of using the same multiplier for the winner as for the closest neighborhood. Therefore, weight update equation for all units, including the winner, became:

$$[5.6]\ w_{ij}(new) = w_{ij}(old) + (\alpha/\ell)[x_i - w_{ij}(old)]$$

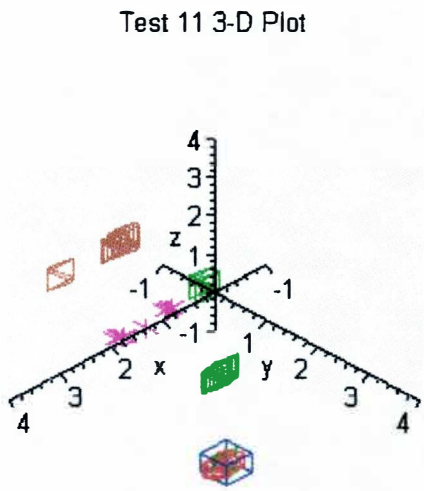
where α = learning rate

ℓ = distance from winner starting at $\ell_{winner} = 1$ to $\ell_{furthest} = 4$ (third ring out from the center)

Test 11 uses the same target nodes for the output clusters as have been used previously. This test was run several times. As the aim of this pattern recognition algorithm is to separate the six chemicals into 6 distinct clusters such that identification of unknown samples is possible, there was some improvement due to the changes made to the weight updating noted above, over the results from test 9. The results from the best of the test 11 runs are shown below in Table 5. It can be seen that the network was able to separate three of the chemicals widely enough to discriminate between them. If the network were run using only data for isoamyl acetate, methyl salicylate, myrcene, and diacetyl, there might be very little overlapping of clusters. The results were plotted in Figure 27.

Test 11 Results		
Chemical	Cluster	#
AlCap	[0,0,0]	2
	[4,4,1]	1
	[3,3,2]	7
MethSal	[4,4,1]	6
	[5,4,1]	3
IsoAmyl	[5,4,1]	9
	[4,4,1]	1
Myrcene	[3,2,3]	4
	[4,2,3]	6
Decanal	[3,1,4]	8
	[4,1,4]	1
Diacetyl	[5,4,1]	10
Clustering Efficiency		78%

Table 5: Test 11 Results



Legend: Allyl Caproate Green Diamond
Methyl Salicylate: Red Circle
iso Amyl Acetate: Blue Box
Myrcene: Magenta Cross
Diacetyl: Gold Diamond
Decanal: Aquamarine Box

Figure 27: Test 11 Results

Test 12- Removal of Outliers

During the initial training of the Cyranose unit using the built-in software, after the initial 60 data vectors were collected, the pattern recognition algorithm was run to determine the effectiveness of the clustering. If the clustering, as indicated by the internal cross-validation, was not 100% effective, outliers were removed and new sets obtained to complete the sample. The choice of which specific samples needed to be removed was different depending on which pattern recognition algorithm was run. The data used in the NN experiments had been selected to obtain a 100% cross-validation result using the

CDA algorithm. Therefore, the data vectors removed may not all have been outliers when the initial data was run through the Kohonen semi-supervised NN algorithm. In the next test, several of the data points which appear to be outliers in test 11 were removed from the data sample and the sample was rerun through the network.

Test 12 started with the results from test 11. As can be seen in Table 5, 7 out of 10 allyl caproate samples were grouped together at point [3,3,2] with a single outlier at [4,4,1] and a pair at [0,0,0]. In test 12, the single outlier was removed and the data was run through the network again. The results can be seen in figure 28. It was noted that the locations of the clusters in this plot were different from those in figure 27. This was due to the random presentation of data to the network during the training phase, leading to a different arrangement of final clustering. However, it was also noted that the overall clustering seen in the previous test remains. The main difference is that allyl caproate (green diamonds) appear in only two positions instead of three, and the single green diamond that had been in the middle of the magenta and blue figures is now gone.

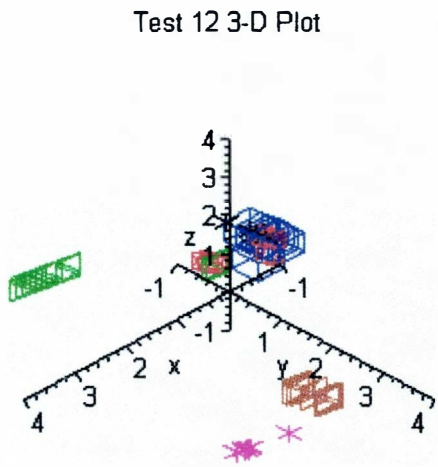
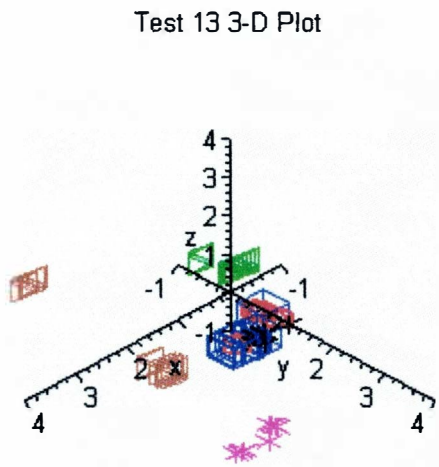


Figure 28: Test 12 Results- Removal of One Outlier

Test 13- Further Outlier Removals

The following three tests investigated the removal of different data vectors. In test 13, the two additional outliers of allyl caproate were removed which resulted in a more distinct allyl caproate cluster than had been seen previously. Evidence of the interaction between the data vectors can be seen in these results. (Refer to figure 29). Previously, the diacetyl data points were reasonably well clustered or more loosely clustered but significantly away from the other data points. With the removal of the two additional allyl caproate points, the diacetyl cluster split into two much more separated clusters. This occurrence was also noticed with the CDA algorithm on the Cyranose system and is therefore not unique to the NN application. It was found previously that in this greatly reduced dimensional representation, the apparent location of a particular data point could

be misleading and not adequately indicate its importance to the system. With test 14, it may have been that one of the outliers could have been removed without causing a reduction in the clustering effectiveness but removing both caused a different set of interactions to come into play.



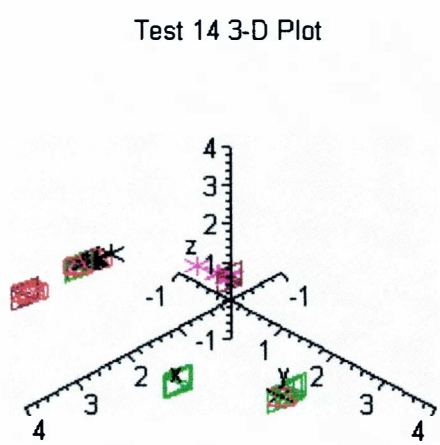
Legend: Allyl Caproate Green Diamond
Methyl Salicylate: Red Circle
iso Amyl Acetate: Blue Box
Myrcene: Magenta Cross
Diacetyl: Gold Diamond
Decanal: Black Cross

Figure 29: Test 13 Results: Removal of Remaining Allyl Caproate Outliers

Test 14- Complete Removal of One Chemical

The NN appeared to have difficulty discriminating between methyl salicylate, isoamyl acetate, and decanal, with all being clustered together as the red, blue, and black symbols above. Test 14 involved removal of one chemical sets completely in an effort to

allow better discrimination between the remaining data vectors. In this test, all of the iso amyl acetate data vectors were removed and the NN was rerun. (Refer to figure 30). These changes made the results worse than before. The diacetyl and allyl caproate have ceased to be distinct clusters. Diacetyl has merged with the decanal and methyl salicylate grouping. This may be because this test was run using the results of test 13 where removal of the two apparent outliers of allyl caproate changed the data interactions. Different results may have been seen if this test had been run with those two points included.

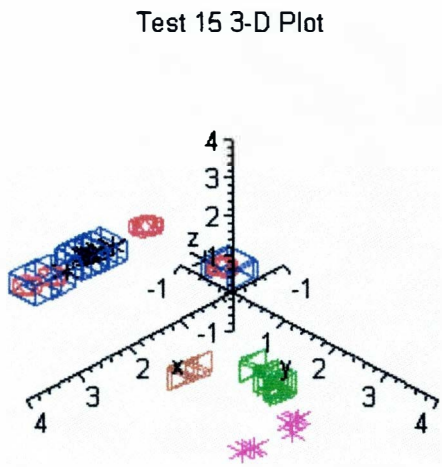


Legend: Allyl Caproate Green Diamond
Methyl Salicylate: Red Circle
Myrcene: Magenta Cross
Diacetyl: Gold Diamond
Decanal: Black Cross

Figure 30: Test 14 Results: Removal of All iso Amylacetate

Test 15- Removal of Three Samples from Test 13

The final test where data vectors were removed was test 15. Preparation for this test began with the results from test 13, after the removal of the single allyl caproate point. Two points from methyl salicylate and one of myrcene were removed. (Refer to figure 31) This test was an improvement on all the tests 12-14. Distinct clusters are apparent for allyl caproate, myrcene, and diacetyl. Decanal consists of a tight cluster of black crosses within the concentration of methyl salicylate and iso amylacetate data points showing no discrimination.



Legend: Allyl Caproate Green Diamond
Methyl Salicylate: Red Circle
iso Amyl Acetate: Blue Box
Myrcene: Magenta Cross
Diacetyl: Gold Diamond
Decanal: Black Cross

Figure 31: Test 15 Results- Removal of Three Additional Outliers

Test 16- New Designated Clusters

There were three more series of investigations to close out this series of experiments. The first was an idea that if the designated clusters were chosen to roughly coincide with the approximate locations of the clusters created using the CDA algorithm on the Cyranose, which were similar for all data samples, the NN might have an easier time assigning the data vectors to clusters. This was test 16. (Refer to figure 32). For comparison, the plot from the CDA Cyranose run using the same data sample as was used for the previous tests can be seen in figure 33. The clusters from this test were used to assign the designated clusters for test 16.

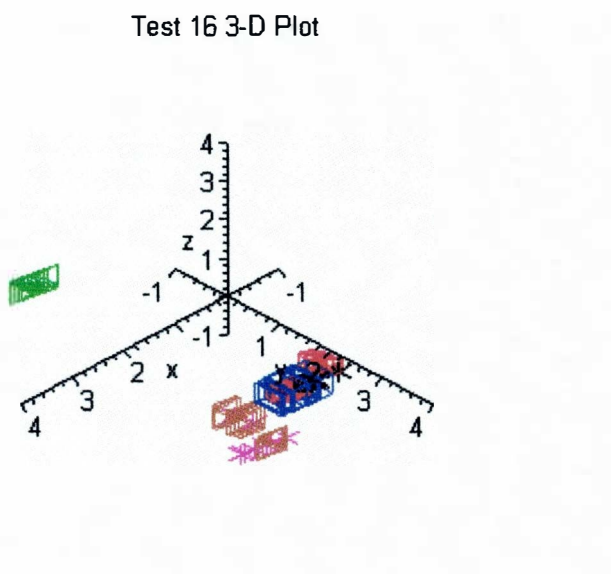


Figure 32: Test 16 Results

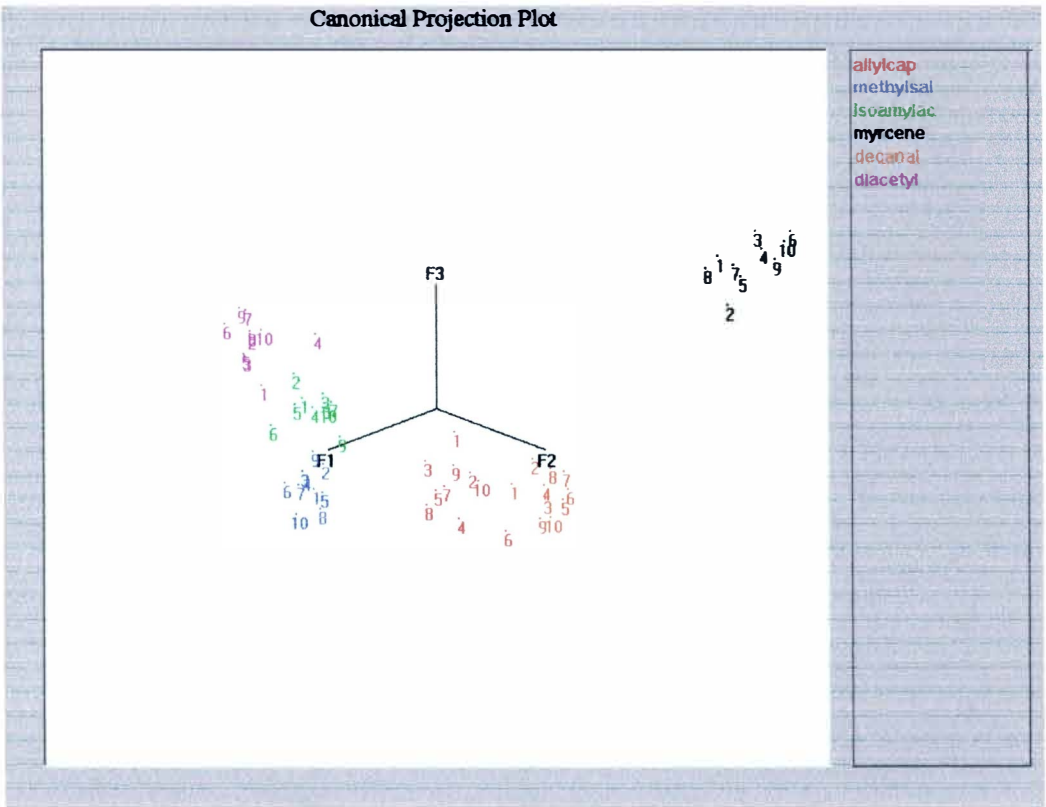


Figure 33: Comparable Results from Cyranose Unit

Test 17- Sequential Presentation of Input Data

The neural network program is designed to present the data to the network in a random order. This results in the data vectors having a varying degree of effect on each other and the clustering produced. In test 17, the data was presented in a sorted order, with the data vectors for each chemical presented sequentially. There was no improvement in the clustering. In fact, the order of the presentation of data to the network does not appear to significantly affect the clustering capabilities of the network. (See figure 34.)

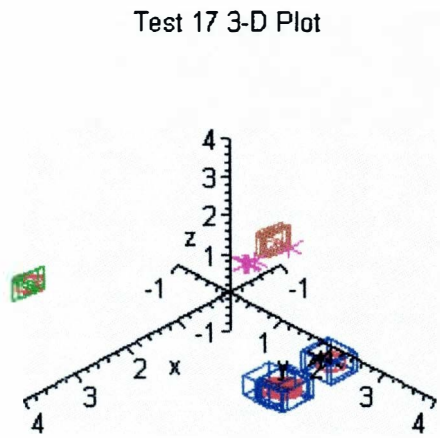


Figure 34: Test 17 Results: Non-Random Data Presentation

Test 18- Random Data

Finally, in order to ensure that the apparent clustering seen in these tests was not due to random chance, a new set of data was created made entirely from randomly chosen numbers in the same range as the actual data. Test 18 shows absolutely no clustering. (Refer to figure 35) This indicates that the clustering seen in the NN plots does indicate ability to group similar data into clusters.

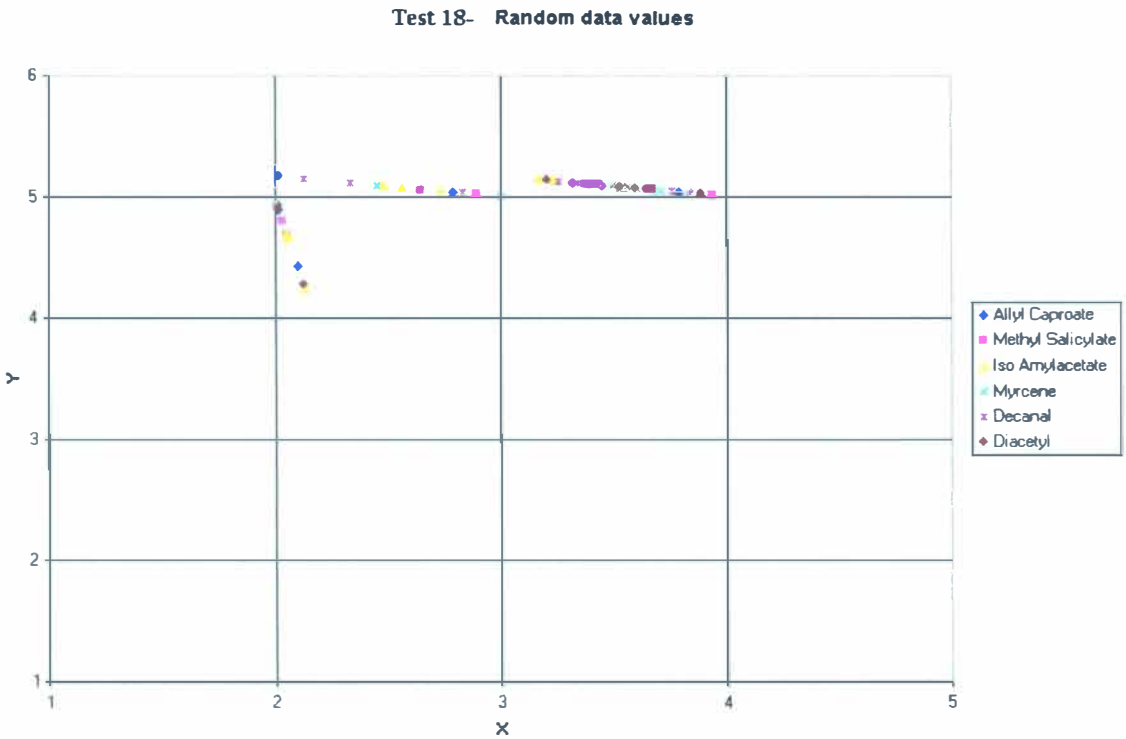


Figure 35: Test 18 Results: Random Data

CHAPTER VI

CONCLUSIONS

Kohonen Neural Network Clustering Ability

The unsupervised Kohonen neural network was unable to cluster the electronic nose data obtained from the Cyranose 320™ Electronic nose. As previously mentioned, as of 1997, unsupervised neural network algorithms had not been applied to electronic nose data. [BEG] It has also been noted by Singh et al. [SHG] that real-world electronic nose data tends not to be well behaved and is often noisy and distorted. This may partially explain why unsupervised neural network algorithms had not been applied to electronic nose data. There did not appear to be any ability to group the input vectors together based on chemical type. It is also interesting to note that according to Sun et al. [SCK], self-organizing neural networks tend to perform worse than K-means clustering even though unsupervised neural networks appear to better suit clustering strategies.

Numerous changes were made to the program in an attempt to improve the clustering ability of the network. The data pre-processing was improved such that it mimicked the pre-processing performed in the chemometrics based pattern recognition system in the Cyranose unit. While this was an important change, making an accurate comparison between the results of the two systems possible, it brought only incremental improvements to the system. Numerous learning rates and neighborhood multiplier update schemes were tested yielding some positive results. The network configuration was changed from a two dimensional cluster array to a three dimensional cluster array

making the neural network results match the configuration of the Cyranose unit, although this did not improve the clustering ability.

Reasonable results in clustering were seen only after the clustering algorithm was changed from an unsupervised to a semi-supervised Kohonen algorithm. Network weights in an unsupervised Kohonen network are usually assigned randomly. If knowledge exists about the ultimate distribution of clusters for a particular problem, this knowledge can be incorporated into the architecture of the network. [F] The input data from the electronic nose contained information about the identity of each input data vector. This knowledge was used in training the neural network to give specific clusters in the network weight values equal to the average values of the input data vectors. This a priori knowledge resulted in a network described in this thesis as 'semi-supervised'. With this change, the network was able to cluster the inputs at an efficiency rate of about 50-70%. This compares to a clustering capability, based on internal cross-validation results, of 100% for the Cyranose unit's build-in chemometrics pattern recognition system. Whilst this level of efficiency from the Kohonen semi-supervised network is not considered sufficient to identify unknown samples with any degree of accuracy, it is a major improvement over the unsupervised network results. The semi-supervised neural network algorithm could be combined with the Cyranose electronic nose by adding the data manipulation and neural network code into the built-in software available with the Cyranose 320™ Electronic nose.

The 3-D clustering graphs indicate that the semi-supervised Kohonen network was able to successfully separate three of the six distinct chemical odorant samples: allyl

caproate, myrcene, and diacetyl. The remaining three samples were assigned to the same or very closely positioned clusters. Although the electronic nose's built-in chemometrics system successfully separated all six chemicals, three of those chemicals were not as well separated as would be ideal. The chemometrics system consistently separated myrcene, decanal, and allyl caproate. The other three chemicals: methyl salicylate, isoamyl acetate, and diacetyl, were more closely concentrated. When choosing the chemicals to use for the initial testing, several were tried and discarded because there were difficulties getting sufficient discrimination from Cyranose built-in identification system. It is highly likely that different combinations of chemicals, concentrations, solvents, or sampling methods would significantly affect the clustering capabilities of the neural network, although possibly in a different way from the effects on the Cyranose pattern recognition system. It is also a possibility that, as was found with the three built-in chemometrics algorithms, a neural network system might work well in cases where the chemometric algorithms did not. This possibility was not tested here.

During the training phase of the Cyranose unit, the pattern recognition algorithm was run many times as data vectors were collected, discarded as outliers, and new sets collected. The data samples ultimately used to achieve the 100% cross-validation on the Cyranose pattern recognition system were then applied directly to the neural network. There was no ability to do any re-sampling once the neural network evaluation commenced. The comparison between the two systems was thus not a fair evaluation in the sense of a scientific comparison and thus gave only an indication of the neural network's capabilities. Experience from training the Cyranose unit on the pattern

recognition algorithms available indicated that the data sample that resulted in a successful clustering for one algorithm did not always result in successful 100% clustering with the other algorithms. It is highly likely that the same situation exists with respect to the neural network algorithm. If the same degree of human supervision of the data sampling had been available for the neural network system, it is highly likely that its performance would have greatly improved.

Avenues for Future Work

There are several avenues for future research into improving the clustering abilities of this semi-supervised Kohonen neural network. The assignment of average input data to weights of the network gave the network some direction in clustering. In the literature, there are references to the use of a combination of PCA analysis to reduce the dimensionality of the incoming data with a neural network as a promising way of clustering data. [K] The Cyranose chemometrics system utilized both PCA for reducing the dimensionality and for identifying outliers and one of three pattern recognition algorithms to successfully cluster the data and make identification of unknowns possible.

Linking the data sampling with the neural network program in order to improve the selection of a workable data sample and avoid the necessity of using a data sample selected for a different pattern recognition system could greatly improve the ultimate performance. This would require close cooperation between a researcher and Cyrano Sciences, Inc. A true comparison of the two systems would start with a random set of data

for six chemicals, with no additional sampling, and control of all other variables. The efficiency rates of the two systems could then be compared.

Finally, other neural network algorithms could be applied to electronic nose data. Algorithms such as fuzzy ART maps and learning vector quantization are both algorithms for clustering input data.

APPENDIX A

KOHONEN NEURAL NETWORK ALGORITHM

KOHONEN NEURAL NETWORK ALGORITHM

The Kohonen self-organizing neural network was developed by Teuvo Kohonen. His original work on self-organizing maps was conducted in the early 1980's and was further developed into a more formal neural network algorithm in 1989. [F] The development of the algorithm followed discoveries that detailed maps of interrelated signals can be formed in a one or two-dimensional array of processing units which had no structure initially. [Ko2] Self-organizing neural networks are also called topology-preserving maps.

During the self-organization process, each input pattern is presented to the network. The cluster unit whose weight vector most closely matches the input pattern is chosen as the winning cluster. In this way, it can be seen that the weight vector serves as a typical input pattern for any particular cluster. As the distance from the winning cluster increases, the similarity between its weight vector and those of the neighbors increases. Each time a winning cluster is chosen for a particular input pattern, the weight vector is updated to more closely match the input vector, improving the chances that the next time a similar input pattern is presented to the network, it will also be assigned to the same winning cluster.

Architecture

The processing units that execute these topological maps are similar to perceptrons. [K] There is a set of input units linked to the output or cluster units. Each link is given a distinct weight value. The methods for choosing weight values is explained later in this appendix. There are m cluster units and they are typically arranged

in a one- or two-dimensional array. A one-dimensional topology is shown in figure A1 below.

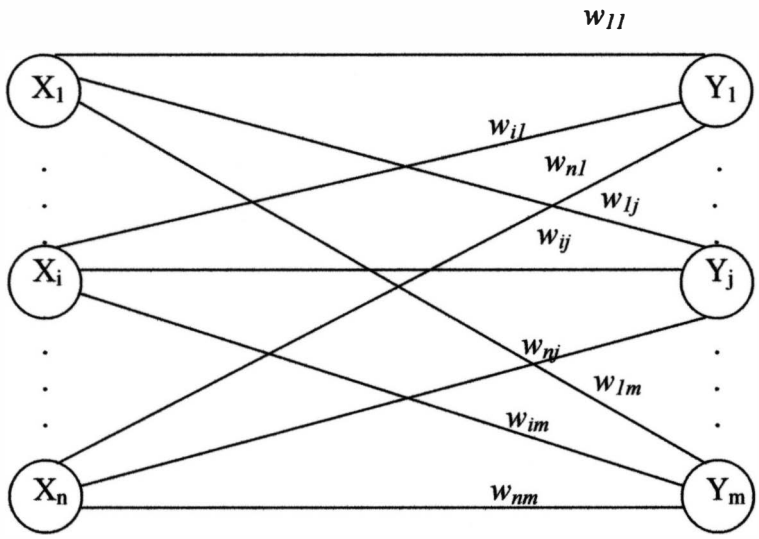


Figure A1: Single Layer Kohonen Network

Each cluster unit has a designated 'neighborhood' around it. The neighborhoods are designated by a radius (R) indicating the distance from the specific cluster unit. The size and shape of the neighborhoods can be varied and is dependent upon the characteristics of the particular problem and on the shape of the cluster array. Figures A2 and A3 show two possible neighborhood arrangements for two-dimensional cluster arrays. Each time the winning cluster's weights are updated, the neighboring cluster unit weights are also updated. The degree to which the neighboring units are updated is dependent upon the distance of the unit from the winning unit.

Note that if the winning unit is near the edge of the grid, some neighborhoods will have fewer units than if the winning cluster were in the center of the array. The neighborhoods do not wrap around from one side of the grid. Missing units are ignored.

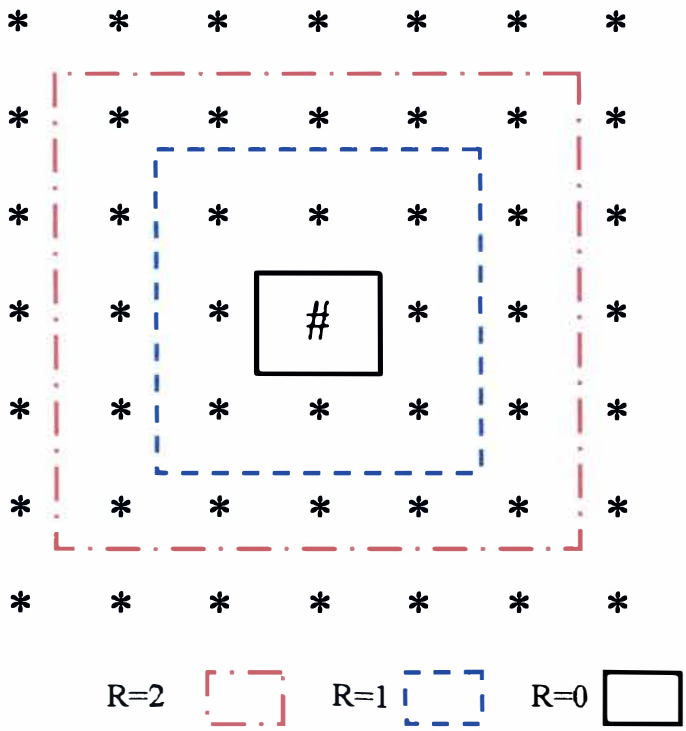


Figure A2: Neighborhoods for Rectangular Cluster Array

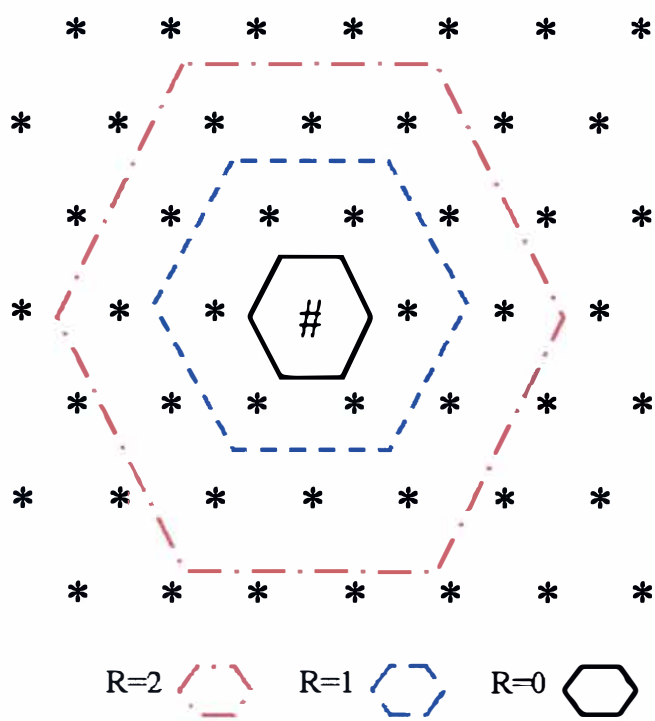


Figure A3: Neighbrhoods for Hexagonal Cluster Array

Kohonen described more complex systems for incorporating neighborhoods in a self-organizing network in a paper in 1982. [Ko2] In this example, the units in the neighborhoods immediately around the winning cluster were updated to make their weight vectors more similar to the winner as before. Additional neighborhoods further from the winner were given an inhibitory update, decreasing any likelihood of similar input vectors being assigned in this region. There is both anatomical and physiological evidence that this type of activity takes place between biological neurons. [K]

Algorithm

Step 0: Initialize weights w_{ij}.

Set neighborhood parameters.

Set learning rate parameters.

Step 1: While stopping condition is false, do steps 2-8.

Step 2: For each input vector **x**, do steps 3-5.

Step 3: For each j, compute:

$$D(j) = \sum_i (w_{ij} - x_i)^2$$

Step 4: Find the index J such that $D(J)$ is a minimum.

Step 5: For all units j within a specific neighborhood of J, and for all i:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha[x_i - w_{ij}(\text{old})].$$

Step 6: Update learning rate.

Step 7: Reduce radius of topological neighborhood at specified times.

Step 8: Test stopping condition.

Weight Assignment, Learning Rate and Neighborhood Reducing Alternatives

Weights in the network are generally assigned random values. If there is some information known about the system or the distribution of the cluster for a particular problem, it may be appropriate that the choice of weights in some way reflect that a priori knowledge.

The learning rate is a slowly decreasing function of time or training epochs. It has been shown that a linearly decreasing function works well for most practical computations [Ko]. It is also possibile to use a geometrically decreasing function to reduce the learning rate.

The maximum size of the neighborhood also decreases as learning progresses. As the network approached the stopping condition, the maximum size of the neighborhood decreases in size. The neighborhood can be reduced one unit each time the learning rate is reduced, one unit for every epoch or several epochs, or any method that fits the situation.

APPENDIX B

A BRIEF INTRODUCTION TO CHEMOMETRICS

CHEMOMETRICS

Chemometrics is defined as the use of multivariate mathematical and statistical techniques to extract valuable but often hidden information from data. [BEG] [Li] These methods have proven essential in areas such as the chemical and flavor industries due to their ability to analyze many variables simultaneously.

Chemometrics includes many methods of data analysis techniques which are applicable to sensor array data analysis. These include data pre-processing techniques, principal components analysis, and pattern recognition algorithms.

Data Pre-processing

Data pre-processing is an important first step in data analysis. Pre-processing of the data is performed on data for a number of reasons. Pre-processing can remove or reduce the effects of systemic and random variations in the data, reduce random and low frequency noise, reduce baseline effects, account for data intercepts, and give more weight to some samples rather than others. The choice of which pre-processing techniques to apply to different data depends on the specific details of the data and the problem being investigated. Often, the exact choice of techniques to apply cannot be determined without testing various techniques on the data.

In general, there are two types of data pre-processing: variable and sample pre-processing. [BPS] Variable pre-processing contains tools that operate on each variable of the data, in other words, on the data columns. Sample pre-processing operates on the individual samples of data, operating on the rows of data.

Variable pre-processing methods include mean-centering, auto-centering, and variable weighting. Mean centering is a common tool that is applied to account for an intercept in the data. Performing the mean centering calculation on the variable data results in the removal of the mean sample vector from all vectors in the data set. This operation generally does not hurt the data analysis and often is beneficial. [BPS] Mean centering is calculated by subtracting the mean of a particular variable vector from each of its elements. The calculation is shown below in equation B1.

$$[B1] \quad x_{imc} = x_i - (\sum_{j=1}^{n} x_j)/n$$

where x_{imc} = mean-centered variable element
n = total number of elements in data column
x_i = variable element prior to mean-centering

Auto-centering to unit variance removes any inadvertent weighting from the data arising from arbitrary units such as from vastly differing samples. [Li] Auto-centering captures information from sensor responses that are very repeatable. Auto-centering is usually not the best variable pre-processing method if the data contains sensors with relatively small responses containing little meaningful information. [Cy2] Auto-centering is calculated by dividing each element of the variable vector by the standard deviation of the vector. This calculation is shown in equation below.

$$[B2] \; x_{iac} = x_i / \text{sqrt}(\Sigma (x_j - \boldsymbol{x})^2/\text{n})$$

where x_{iac} = auto-centered variable element
$\boldsymbol{x}$ = mean of variable elements

A variation on the previous two centering methods is a combination of the methods. It is possible to standardize each variable by subtracting the mean of the variables from each individual value then dividing by the standard deviation. [B]

Variable weighting is the last variable pre-processing method. Variable weighting is used to emphasize some variables over others to increase the influence of those variables on the model. Variable weighting would consist of multiplying all entries for a particular variable by some constant to increase its effect on the data analysis.

There are four types of sample pre-processing used to prepare data for analysis. These are normalization, of which there are several methods, weighting, data smoothing, and baseline corrections. [BPS]

Normalization of a sample vector is accomplished by dividing each variable in the sample set by a constant. There are three types of normalization: '1-norm', '2-norm', and max intensity, or 'infinity-norm'. The '1-norm' normalization involves dividing the sum of the absolute values of all entries for a particular sample vector from each entry. (See equation B3.)

$$[B3]\quad x_j' = x_j / \left(\sum_{j=1}^{nvars} | x_j | \right)$$

The 2-norm is the normalization of each value to unit length and is calculated by dividing each value by the square root of the sum of the squared values of the sample vector as indicated in equation [B4].

$$[B4]\quad x_j' = x_j / \mathrm{sqrt}\left(\sum_{j=0}^{nvars} x_j^2 \right)$$

There are potential difficulties with 2-norm as it is possible to lose the variability in some of the variables. A compromise approach would be to selectively normalize certain variables. With this method, the summation would include only those variables

that are being normalized instead of all variables. The third normalization method, ∞-norm, is normalization to a maximum intensity. If the maximum intensity equals one, this is calculated by dividing each element by the ∞-norm, which is the maximum, in absolute value, of the sample vector. Normalization is applied specifically to remove systemic variations.

$$[B5]\ x'_j = x_j / \left(\max_{j=1}^{nvars} | x_j |\right)$$

Sample weighting, like variable weighting, involves multiplying each element in the sample vector by some constant. Weighting should only be applied when very reliable information is available about the relative importance of some samples over others.

Sample smoothing is used to mathematically reduce the random noise in a sample with the goal of increasing the signal to noise ratio.

Baseline corrections remove low-frequency sources of variation, which are not related to the chemistry of the system being investigated. [BPS] These variations can be large relative to the changes in the signal of interest. An explicit modeling approach is used to remove the baseline effect.

If a sample vector is r = f(x), and

$$[B6]\ r = r' + \alpha + \beta x + \gamma x^2 + \delta x^3 + \ldots$$

where r' is the signal of interest and the remainder is the baseline effect. By postulating a model for the baseline (offset, linear, polynomial, etc.), the offset can be accounted for through subtraction.

Principal Components Analysis

Principal Components Analysis (PCA) is an unsupervised mathematical manipulation of a data matrix where the goal is to represent the variation present in many variables using a smaller number of 'factors' or dimensions. [BPS] This allows the investigator to view the true multivariate nature of the data in a relatively small number of dimensions, allowing human pattern recognition to be used to identify structures within the data. PCA is an excellent tool for preliminary data exploration. It is useful for examining data sets for expected or unexpected clusters and for the presence of outliers. PCA can be used to filter out noise in a system and can provide the amount of variation contained by each measurement variable.

Supervised Pattern Recognition Algorithms

There are several supervised pattern recognition algorithms available for in-depth data analysis. These methods are used when the goal is to construct a model to be used to classify future samples. Supervised learning is accomplished using a set of data with known classifications to "train" the system to distinguish between classes. The algorithms include K-nearest neighbor, K-means, and Canonical Discriminate Analysis.

K-nearest neighbor is a general approach for classifying unknown samples. The assignment of an unknown is accomplished according to the majority vote of its K-nearest neighbors in the training set in the multi-dimensional space. To classify an unknown, the distance is calculated between it and a set of samples of known class. The closest k samples are then used to make the classification. The choice of the value of k is

determined by a cross-validation procedure and is often equal to the maximum number of samples in the class with the fewest members. [Cy1]

K-means is similar to K-nearest neighbor. This prediction algorithm assigns unknowns to a class based on the Euclidian distance between the class centroid and the sample in the multi-dimensional space. [Cy1] The predicted class of an unknown is assigned to the class of the sample(s) lying nearest to it in multi-dimensional space. The Euclidian distance is commonly used to measure the nearness between samples. The Euclidian distance is calculated by the equation below [B7].

[B7] Euclidian Distance = $\text{sqrt}((x_1-y_1)^2 + (x_2-y_2)^2 + \ldots + (x_{nvars} - y_{nvars})^2)$

Canonical Discriminant Analysis is a supervised learning algorithm, which assigns unknown samples to classes based on the criterion of the shortest Mahalanobis distance between its centroid and the sample in canonical space. [Cy1] The Mahalanobis distance is measured in terms of the standard deviation from the mean of the training samples. It differs from the Euclidian distance in that it takes into account sample variability. It weights the differences by the range of variability in the direction of the sample point. [Th]

In chemometrics data analysis, the aforementioned methods are often used in combination to improve the data analysis. [K] Extensive testing is done to determine which combinations work best with a particular data set.

APPENDIX C

DATA CONVERSION AND PREPROCESSING PROGRAMS

DATA CONVERSION AND PREPROCESSING PROGRAMS

Program Name: 'dataConvert.cxx'

```
/* This program is designed to convert enose data
 * from the file format from the cyranose unit into
 * a data file usable with the NN for the enose.
 * The program will take several data files and
 * convert and combine them so that adequate data
 * samples are available to train the NN
 */

/* Created 1/18/01
 * Program works for old and new data files
 *Will use ANSI std include files and notation
 * 2/12/01 including randomizing of file
 * works for randomizing 2 data files of 60 datasets each 2/14/01
 * 2/12/01 including normalization of data
*/
/* Edited for final data file format 11/01
 * Will take data file directly from nose
 * Files will be named 'train1'... 'train5'
 * Works 11/13/01
 * Determined that randomization of training data is required, 11/14/01
 * Code included 11/14/01
 * Testing started
*/

#include <iostream>
#include <stdio.h>
#include <fstream>
#include <stdlib.h>
#include <string>
#include <ctime>

using namespace std;
using std::fstream;
using std::ostream;
using std::ofstream;

const int SIZE = 480;

void random_array(string[], int);
```

```
int main()
{
	//read in name of file
	char firstLine[80];
	string classNumber;
	char fileNameIn[30];
	char fileNameOut[30];
	int i=0, j=0;
	char convert = 'n';
	double data;
	//temp var to hold data read from file
	char ans = 'y';
	int dataset = 0, row = 0;

	do{
	//This allows conversion of multiple files
	cout<<"Enter the name of the file to convert: ";
	cin>>fileNameIn;
	cout<<"fileName = "<<fileNameIn<<endl; //this works
	ifstream infile(fileNameIn , ios::in);

	if(! infile ) {
		cerr<<"Input file could not be opened."<<endl;
		exit(1);
	}

	//open output file
	//Make this general- ie add line to enter outfile name, 11/12/01
	cout<<"Enter the name of the output file:  ";
	cin>>fileNameOut;
	ofstream outfile(fileNameOut, ios::app);
	if(! outfile ) {
		cerr<<"Output file could not be opened."<<endl;
		exit(1);
	}

	infile>>firstLine;
	//get to first line of valuable information
	while(firstLine[0]!='E'){
		//Get next line
		infile>>firstLine;
	}
	cout<<"\n"<<firstLine<<"\n";
	infile>>data;
	cout<<data<<endl;
	//Put first data point into outfile
	outfile<<data<<" ";
	row=0;
	while (!infile.eof()){
		j=0;
		infile>>data;
		outfile<<data<<" ";
		if(data >=1){
			outfile<<endl;
			dataset++;
			row++;
			infile>>firstLine;
		}  //end if
		if(row == 60)
			break;
```

```
        } // end while
    //ask if user wants to convert another file
    do {
        cout<<"Do you have another file to convert? ";
        cin>>ans;
        } while(ans != 'y' && ans != 'n');
    outfile.close();
    infile.close();
    } while(ans == 'y');

    //start of randomization portion of program
    //Use same file for input as used for output above
    ifstream infile(fileNameOut , ios::in);
    if(! infile ) {
        cerr<<"Input file could not be opened."<<endl;
        exit(1);
    }
    cout<<"Enter the name of the final output file:  ";
    cin>>fileNameOut;
    ofstream outfile(fileNameOut, ios::app);
    if(! outfile ) {
        cerr<<"Output file could not be opened."<<endl;
        exit(1);
    }
    char firstline[SIZE];  //grabs each line of data file
    string dataLine[300];  //pointers to strings holding each line of
data
    int fill[300] = {0};
    int k = 0;
    int sets=0;
    cout<<"How many data sets in infile?\n";
    cin>>sets;

    //read in data from file
    while (!infile.eof()){
        cout<<"In while\n";
        j = rand()%sets;
        cout<<"j = "<<j<<endl;
        while(fill[j] == -1){
            //find another j
            j = rand()%sets;
        }
        infile.getline(firstline,SIZE,'\n');
        cout<<"firstline: "<<firstline<<endl;
        dataLine[j] = firstline;
        fill[j]=-1;
        cout<<"Data line "<<j<<": "<<dataLine[j]<<endl;
        k++;
        cout<<"k = "<<k<<endl;
        if(k == sets)
            break;
    }
    cout<<"k = "<<k<<endl;

    //randomize data lines
    random_array(dataLine,sets);

    //Now put lines into output file
    for(i=0;i<sets;i++){
```

```
            outfile<<dataLine[i]<<endl;
            cout<<"Line "<<i<<" : "<<dataLine[i]<<endl;
      }

      return 0;
}

/* each line in the incoming data file should be put into
 * strings before being sent to this function
 * Function: random_array()
 * Input: string[], an ordered array of strings
 * Output: string[], a randomized array of strings
 */

void random_array(string B[], int size)
      {
      //Initialize variables
      int i,j;
      string R[size];  //string to hold randomized array

      //random seed
      srand(time(0));
      cout<<"In random_array()\n";
      for(i=0;i<size;i++){
               j=rand()%size;
                //correct the comparison below
               while(B[j] == ""){
                        //find another value
                        j = rand()%size;
                }
            cout<<"in for, j = "<<j<<endl;
                R[i] = B[j];
                //set B[j] to 0 so not used again
                B[j] = "";
        }
        //put mixed up values back into B[]
        for(i=0;i<size;i++){
                B[i]=R[i];
        }
        for(i=0;i<5;i++)
                cout<<"B["<<i<<"]: "<<B[i]<<endl;
} // end random_array()
```

Program Name: 'enose_fullnorm_k2.cxx'

```
/*This program takes raw enose data and normalizes it for inclusion into
a neural net testing program using the Monarch library.
*/
//This program creates enosetrain.dat as of 2/14/01
/* This program was altered 11/16/01 to create data to use for Kohonen
networks

 * Data normalized to between 0 and 1

 * 3/7/02 Normalize up to 300 data sets of entire data from enose
 * 3/25/02 Change normalization to 0-1.0
 * 9/3/02  Change normalization to match that done by Cyranose unit
 *          3:10 pm test on 2-norm only
 *    twonorm works @3:30 pm
 *   10/21/02   alt preprocessing added- auto-scaling
*/


#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctime>

void twonormal(double [300][35], int, int, double[300][35]);
//double[rows][columns] rows == 300, cols == 35
      // two norm operates on rows
void meancenter(double [300][35], int, int, double[300][35]);
//preprocessing on columns

void autocenter(double [300][35], int, int, double[300][35]); //another
preprocessing on columns

int main(){
      //int seed;
      srand(time(0));
      //Internal variables
   ifstream in_file;

   char fileNameIn[30];

   char fileNameOut[30];

   ofstream out_file;
   double value;
   ofstream middata;

   //double subscript array: row, column
   //enose kohonen up to 300 data sets, 35 inputs
   double raw[300][35], train1[300][35], train2[300][35];
   int i=0;
   int j=0,k = 0, m=0;
   int cols;
   int rows;
```

```
//Open files
//open raw data file and neural net data file

cout<<"Enter the name of the file to normalize: ";
cin>>fileNameIn;
//cout<<"fileName = "<<fileNameIn<<endl;
in_file.open(fileNameIn , ios::in);

cout<<"Enter the name of the output file:  ";
cin>>fileNameOut;
out_file.open(fileNameOut, ios::out);

if(!in_file){
   cout<<"Error opening a data file.\n";
   exit(-1);
   }

if(!out_file){
   cout<<"Error opening secondary output file.\n";
   exit(-1);
   }

//middata.open("intermed.dat", ios::out);
//input number of rows of datasets in input file

cout<<"Enter the number of datasets in input file.\n";
cin>>rows;

//Bring in all data from raw data file for testing data
while(k<rows){
   if(k == 0)
         in_file>>value;

   j=0;
   while(j<=32){//End of characteristic array not reached
         //read in each array

         raw[k][j] = value; //change raw- need different array
         j++;
         in_file>>value;

   }// end inner while

   cols = j;
   k++;

} //end of read in while loops

//Normalize train data, i = # of train sets
// For Kohonen, normalization between 0 and 1
// 9/3/02; to match Cyranose, use 2-norm
// 2norm = sqrt(SUM(xj^2)), j=1 to nvars
// Send whole array to function
// Only normalize cols 0-31, not 32- chem col
int n;
for(k=0; k<rows;k++){
   twonormal(raw, k, cols,train1);
```

```
   } //end for loop

  /*
  //mean centering of columns, except for chem column
  for(j=0;j<cols-1; j++)
     {
     //send column to function
     meancenter(train1, j, rows, train2);
  }
  */

  // 10/21/02 auto-scaling for preprocessing
  for(j=0;j<cols-1; j++)
     {
     //send column to function
     autocenter(train1, j, rows, train2);
  }

   // Add chem number to train2 martix
  for(i=0;i<rows;i++)
  {
        train2[i][32]=raw[i][32];
  }

   //Then output characteristic sets to NN data file
   k=n=0;
   //Fill train data file, 300 data sets
   // No randomizing of order
   for(k=0;k<rows;k++)
      {
      n=0;
      for(n=0;n<cols;n++)
         {
            //middata<<train1[k][n]<<" ";
            out_file<<train2[k][n]<<" ";
         }
      out_file<<endl<<endl;
      //middata<<endl;
      }

in_file.close();
   //in_fileTrain.close();
   out_file.close();
   //middata.close();
} //end main
```

```
void twonormal(double b[300][35], int row, int cols, double
twonorm[300][35])
      {
      //operates on rows
      int i=0;
```

```
        double sum = 0;
        double sqvalue; // square of value in row
        double srvalue; //square root of sqvalue
        //sum the squares of all values in row
        for (i=0;i<cols-1; i++)
              {
              sqvalue = pow(b[row][i],2);
              sum = sum + sqvalue;
              }
        //cout<<"sum = "<<sum<<"\n";
        srvalue = sqrt(sum);
        for (i=0;i<cols-1; i++)
              {
              twonorm[row][i] = b[row][i]/srvalue;
              }

        // Enter chem value in last column into new matrix
        twonorm[row][cols]= b[row][cols];
        } //end twonormal()


void meancenter(double c[300][35], int col, int rows, double
meancntr[300][35])
{
        //mean centering operates on columns
        // subtract the mean of that variable vector from each element
        int i=0;
        double mean =0;
        double sum = 0;
        //calculate mean of variable vector
        for (i=0; i< rows; i++)
           {
           sum = sum + c[i][col];
           }
        mean = sum/rows;

        //calculate new element value
        for (i=0; i<rows; i++)
           {
           meancntr[i][col]= c[i][col] - mean;
           }

        }

void autocenter(double c[300][35], int col, int rows, double
autocntr[300][35])
{
        //auto centering operates on columns
        // mult each value in column by inv of std dev
        int i=0;
        double mean =0;
        double sum = 0;
        double difsum = 0;
        double stddev;

        //calculate mean of variable vector
        for (i=0; i< rows; i++)
           {
           cout<<c[i][col]<<" ";
```

```
   sum = sum + c[i][col];
   }
cout<<"\nsum = "<<sum<<endl;
mean = sum/rows;
cout<<"Mean = "<<mean<<endl;
for (i=0; i<rows; i++)
{
      difsum = difsum + pow((c[i][col]-mean),2);
}
cout<<"difsum = "<<difsum<<endl;
// calculate standard deviation
stddev = sqrt(difsum/rows);
cout<<"stddev = "<<stddev<<endl;
//calculate new element value
for (i=0; i<rows; i++)
   {
   autocntr[i][col]= c[i][col]/stddev;
   }
}
```

APPENDIX D

KOHONEN NEURAL NETWORK SOURCE CODE

KOHONEN NEURAL NETWORK
SOURCE CODE

File Name: "setup.h"

```
#define N_INPUTS 32
#define N_OUTPUTS 6
// enose has 32 sensors and 6 "outputs" or clusters
// 11/21/01 smaller test set of data to work out bugs

#define DATA_SIZE n_inputs
#define N_TESTS 60

real Layer :: f(real x)     { return purelin(x);     }
real Layer :: f_dot(real x) { return purelin_dot(x); }

static real Gain=DEFAULT_GAIN, Eta=DEFAULT_ETA, Alpha=DEFAULT_ALPHA;
static real Epsilon=DEFAULT_EPSILON;
static integer n_inputs=N_INPUTS, n_outputs=N_OUTPUTS;

void _alpha(char *s) { Alpha = atof(s); }
void _eta(char   *s) { Eta   = atof(s); }
void _gain(char  *s) { Gain  = atof(s); }

#define N_PARMS 5

parameters test_ini[N_PARMS] = {
    { "echo",  (function) _echo    },
    { "*",     (function) _comment },
    { "alpha", (function) _alpha   },
    { "gain",  (function) _gain    },
    { "eta",   (function) _eta     },
};
```

File Name: "App.h"

```
// Altered 3/6/02 for neighborhoods

#include <kohonen.h>

// adjust for neighborhoods, 2/25/02

class App : public MonarchApp {
    private:
        array *dummy_target;
    public:
        InputLayer *il;
        OutputLayer *ol;
        KoNet *net;

        void Simulate(array&, matrix3&, int);

        App(int argc, char *argv[]);
        ~App(void);
};

// out changed to matrix, 2/25/02
void App::Simulate(array& in, matrix3& out, int r) {
    il->from(in);
    net->simulate(*dummy_target, r);  // Kohonen nets are self
orgainizing... no target.
    ol->to(out);
}

App::App(int argc, char *argv[]) :
  MonarchApp(argc, argv, "enose_k.ini", N_PARMS, test_ini) {
    il = new InputLayer("Inputs", n_inputs);
      //output layer a matrix of n_outputs^2, changed 2/25/02
      //output layer for a 3D matrix n_outputs^3, changed 4/26/02
    ol = new OutputLayer("Outputs", n_outputs*n_outputs*n_outputs);

    net = new KoNet("Ko", 2, Alpha, Epsilon);
    net->add(new KoLink(il, ol));

    dummy_target = new array(0);
}

App::~App(void) {
    delete net;
}
```

File Name: "kohonen.h"

```
/*
**  This was originally written by John and Jet (jettero@voltar-
confed.org)
**  This is known to not work quite right.  We think it's pretty
**  close though.
**   11/01  Additions and corections added by Lori Evesque
**    11/27/01    Add function to reduce size of weight matrix to get
rid of biases
**    2/25/02           Adding changes for incorporation of
neighborhoods
**                      May need to add class KoLayer
**    3/5/02            Don't need KoLayer due to inheritance but do
need KoNet simulate fn
**                      to get size of neighborhood data to simulate fn
*/
#ifndef KoHoH__
#define KoHoH__

#include "monarch.h"

#define DEFAULT_EPSILON 0.0001
#define DEFAULT_K_ALPHA 2

class KoNet;

class KoLink : public Link {
    protected:
        KoNet *net;
        array *activation;
    public:
      KoLink(Layer *from, Layer *to);
      virtual ~KoLink(void);
      KoNet *netOf(void) { return net; };
      void netIs(KoNet *n) { net = n; };
      virtual void propagate(void);
      void adjustk(int&);
      void update(int,int);
      void update_wts(double[], int) ;
};

class KoNet : public Net {
    protected:
      real alpha, epsilon;
    public:
      KoNet(char *n, integer l,
            real e=DEFAULT_K_ALPHA, real ep=DEFAULT_EPSILON
        );
      ~KoNet(void) { };
      void add(KoLink *l) {
            l->netIs(this);
            links[next++] = l;
        };
      void addAt(integer i, KoLink* l) {
```

```
            l->netIs(this);
            links[i] = l;
        };
      real alphaOf(void) { return alpha; };
        void setAlpha(real alin) { alpha = alin; }
      real epsilonOf(void) { return epsilon; };
      virtual void simulate(array&, int);
      virtual void adjustk(int&);
      void update_wts(double[], int);
};

#endif
```

File Name: "kohonen.cpp"

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <iostream.h>

#include "kohonen.h"
//#define DEFAULT_ALPHA 2
#define ALPHA (net->alphaOf())
//#define ALPHA 2
#define EPSILON (net->epsilonOf())

KoLink :: ~KoLink(void) { delete activation; }

KoLink :: KoLink(Layer *from, Layer *to) : Link(from, to) { }

void KoLink :: propagate(void) {
    real sum;
    //cout<<"In KoLink propogate\n";
    for (integer i = 1; i <= upper->numberUnits(); i++) {
        sum = 0.0;
        for (integer j = 1; j <= lower->numberUnits(); j++) {
            sum += sqr(((*weights)[i][j]) - (*(lower->output))[j]) ;
        }
        upper->output->addAt(i, lower->f(sum));
    }
}

//Streamlines adjustk function
//Performs weight updating
// 5/2/02
//

void KoLink :: update(int l, int m){
      real wOld;

      if(m>= 0 && m < upper->numberUnits()){
            //index m is within array
            for(int i=1; i<= lower->numberUnits(); i++){
                  wOld = weights->getAt(m,i);
                  weights->addAt(m,i,wOld + ALPHA/(l+1) * ((*(lower-
>output))[i]- wOld ));
            }
      }
} // end update


// Change for neighborhoods
// 2/26/02 adjust nearest neighbors by half amount as winner
// will try runnng 3/6/02
// Change to incorporate 3D., 5/2/02
// 11/5/02  a)Adjust neighbors only-- not winner
```

```
// 11/05/02 b) adjust neighbors by ALPHA/(l+1) and winner by ALPHA/l
//    therefore, update() above also edited 11/5/02

void KoLink :: adjustk(int& r){
      real sum;
      real wOld;
      real least = MAX_REAL;
      integer J;
      integer i,j;
      int l=r; //l and k determine neighbor
      int k,m; // m is neighbor index
      int n;
      int nout = pow(upper->numberUnits(), 1/3); //N_OUTPUTS
      int nout2 = pow(nout,2);

      // find 'winner'
      for (j=0; j<= upper->numberUnits(); j++) {
            sum = 0.0;
            for(i=0; i<= lower->numberUnits(); i++) {
                  sum += sqr( ((*weights)[j][i]) - (*(lower->output))[i]
);
            }

            if(sum < least) {
                  least = sum;
                  J = j;
            }
      }

      int z = floor(J/pow(nout,2)); //level winner is on
      //cout<<"in adjustk winner = "<<J<<endl;
      // what is J?  take out after debugging
      //cout<<"J = "<<J<<"   l="<<l<<endl;

      //change entire neighborhood
      //start changing weights furthest from winner to winner

      while (l>0){
            for (k = -l; k<=l; k++){
                  //check for neighbors on left side of array
                  if(J%nout-l >= 0){
                        //there are neighbors on left side
                        m = J-l-(k*nout);
                        if(m>= (z*pow(nout,2))){
                              // m on same level as J
                              update(l,m);
                        }
                  }

                  //cout<<"wOld = "<<wOld<<"  Alpha/l/2 =
"<<100*ALPHA/(l*2)<<endl;
                  //check for neighbors on right side of array

                  if (J%nout+l < nout){
                        //neighbors on right side exist
                        n=J+l+(k*nout);
                        //make sure n is on same z level
```

```
                        if(n> (z+1)*pow(nout,2)){
                              //n on same level
                              update(l,n);
                        }
                  }
            } // end outer for loop for left and right sides

            // top and bottom neighbors
            k = l - 1;
            // determine 1 node right of leftmost top node
            m= J + l*nout + k;

            //assign one right of bottom left node
            n = J - l*nout + k;

            //move across top and bottom rows left to right
            while(k<l){
                  if((J+l*nout)< ((z+1)*(pow(nout,2)))){
                        //top row exists
                        update(l,m);
                  }
                  if((J-l*nout)>= z*pow(nout,2)){
                        //bottom row exists
                        update(l,n);
                  }
                  k++;
                  m = J + l*nout + k;
                  n = J - l*nout + k;
            }
            //Adjust for third dimension
            //Does upper boundary exist?
            if(z+l < nout){
                  //top plane exists
                  //adjust nodes on plane within neighborhood
                  for(int i = -1; i<=1; i++){
                       for(int j = -1; j<=1; j++){
                        //calculate index
                        m = J + i*nout + j + l*int(pow(nout,2));
                        //determine if index on correct level
                        if(m>=(z+l)*pow(nout,2)&&
m<(z+l+1)*pow(nout,2)){
                              if(m>= (J-l+i*nout+l*pow(nout,2)) && m <
(J+l+i*nout+l*pow(nout,2))){
                                    update(l,m);
                              }
                        }
                     }
                  }
            }

            if(z-l >= 0){
                //bottom plane exists
                //adjust nodes on plane within neighborhood
                for(int i = -1; i<=1; i++){
                  for(int j = -1; j<=1; j++){
                     //calculate index
                     n = J + i * nout + j - l*int(pow(nout,2));
```

```
                        //determine if index on correct level
                        if(n>=(z-l)*pow(nout,2) && n < (z-
l+1)*pow(nout,2)){
                           if(n>= (J-l+i*nout+l*pow(nout,2)) && n<
(J+l+i*nout+l*pow(nout,2))){
                              update(l,n);
                           }
                        }
                     }
                  }
               }

               //Adjust square of nodes on levels between top and bottom
               for(int s=1-l; s<l; s++){
                  //adjust lower levels to upper
                  for(k = -l; k<=l; k++){
                     //check for neighbors on left side of array
                     if(J%nout-l >= 0 && ((z+s)<nout && (z+s) >= 0)){
                        //there are neighbors on left side and on level z+s
                        m = J-l-(k*nout)+ s*int(pow(nout,2));
                        cout<<"m = "<<m<<endl;
                        if(m>= ((z+s)*pow(nout,2))){
                           //m is s levels from j
                           update(l,m);
                        }
                     }
                     //check for neighbors on right side of array
                     if(J%nout + l < nout && ((z+s)< nout && (z+s) >=0)){
                        //neighbors on right side and on level (z+s) exist
                        n = J+l+(k*nout) + s*int(pow(nout,2));
                        //make sure n on same z level
                        if(n< (z+s+1)*pow(nout,2)){
                           //n is levels from J
                           update(l,n);
                        }
                     }
                  } // end outer 'for' loop for left and right sides
                  //top and bottom neighbors
                  k = 1-l;
                  //determine 1 node right of leftmost top node on level
z+s
                  m = J + l*nout + k + (s*int(pow(nout,2)));

                  //assign one right of bottom left node on level z+s
                  n = J - l*nout + k + (s*int(pow(nout,2)));

                  //move across top and bottom rows left to right

                  while(k<l){
                     if(J+l*nout+(s*pow(nout,2))< (z+s+1)*pow(nout,2)){
                           //top row exists
                           update(l,m);
                     }
                     if((J-l*nout + (s*pow(nout,2)))>= (z+s)*pow(nout,2)){
                           //bottom row exists
                           update(l,n);
                     }
```

```
                    k++;
                    m= J + l*nout + k+ (s*int(pow(nout,2)));
                    n = J - l*nout + k + (s*int(pow(nout,2)));
                }
            } // end for loop to adj btw levels
            l--;
      } // end l>0 while

      // A) Don't change winner at all, 11/06/02
      // B) Change winner by ALPHA,  11/06/02
      //change winner separately
      for(i=1; i<= lower->numberUnits(); i++) {
            wOld = weights->getAt(J,i);
            weights->addAt(J, i,
                  wOld + ALPHA * ((*(lower->output))[i] - wOld));
      }


} //end KoLink::adjustk()


/*
 * update_weights() -- updates weight matrix for specific weights
 *  Input- weight values (array) and 3-D matrix location of output,
single variable
 */

void KoLink :: update_wts(double a[], int out)
      {

      // go through all inputs and change weights for specified output
      for(int i=1; i<= lower->numberUnits(); i++){
            weights->addAt(out,i,a[i-1]);
            }
      }



KoNet :: KoNet(char *s, integer n, real a, real ep) : Net(s, n) {
    alpha = a;
    epsilon = ep;
}

/*
 * adjust() -- adjust weights
 *   This goes forward through the network.
 *   {*} Use a forward loop contruct macro
 */
void KoNet :: adjustk(int& r)
      {
// cout << "\n" ; // {*} just for debugging
      for (integer i = 0; i < n_layers-1; i++)
            {
            links[i]->adjustk(r);
            }
```

```
}


/*
 * simulate() -- run (and possibly) train the network
 *   Runs a single epoch
 */
void KoNet :: simulate(array& target, int r)
      {
      if (phase != setup)
            {
            //cout<<"In KoNet simulate\n";
            ++epoch;
            propagate();
            computeError(target);
            if (phase == training)
                  {
                  backpropagate();
                  adjustk(r);
                  }
            }
      }


/*
 *  update_wts() -- allows for the updating of individual weight arrays
 *  This function updates all the weights connected to a specific output
node
 *  Inputs: weight array, output node indicated by a number
 *  Outputs: none, alters  the pre-existing weight matrix only, nothing
returned
 *  Function created 10/1/02, LLE
 *
 */

void KoNet :: update_wts(double wts[], int node)
{
      //send parameters to link function
      for (int i=0; i< n_layers-1; i++)
         {
         links[i]->update_wts(wts, node);
         }
}  // end KoNet::update_wts()
```

APPENDIX E

KOHONEN NEURAL NETWORK DRIVER

KOHONEN NEURAL NETWORK DRIVER

Neural Network Driver Program-
File Name: 'enose_k.cpp'

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <kohonen.h>
#include "setup.h"
#include "App.h"
#include <time.h>

// 2/25/02 begin alterations for 'neighborhoods'
// 4/16/02 begin alterations for 3D clustering
// 5/13/02 add function to display clustering efficiency
// 9/24/02 start semi-supervised operation
//    import weights for 6 nodes based on averages from data files
// 11/18/02  This version assigns winners to be closest to enose results

const double PI = 3.14159;
real A;
// Size of neighborhood
int rad = ceil(sqrt(N_OUTPUTS));
const int cube = int(pow(N_OUTPUTS,3));
//efficiency measurement
int tally[N_OUTPUTS][216];

// Frequency of changing neighborhood
// also related to number of outputs
int rchange = ceil(sqrt(rad));
array Vectors(N_INPUTS * N_TESTS);
//function prototype, this for 3D
double angle_det(double, double, double, double, double, double,
double);
//function to selectively update weight martix for semi-supervision,
9/26/02
void update_wts(fstream&);
App *theApp;
void dataConvert() {
      //internal variables
      double x[300], y[300],
err[300],a_rad[300],x2[300],y2[300],err2[300];
      double z[300], z2[300], a_rad2[300];
      int chem[300];
      int i=0;
      double max = -1000;
      double min = 1000;
      //Input data file name
      ifstream raw_data("data_out.dat" , ios::in);
      if(! raw_data ) {
```

```
            cerr<<"Input file could not be opened."<<endl;
            exit(1);
      }
      //Output data file name
      char fileNameOut[30];
      cout<<"Enter the final data file: ";
      cin>>fileNameOut;
      ofstream final_data(fileNameOut, ios::out);
      //ofstream final_data("final_data1.dat" , ios::out);
      if(! final_data ) {
            cerr<<"Output file could not be opened."<<endl;
            exit(1);
      }

      //Chemical data file name
      char fileNameChem[30];
      cout<<"Enter the chemical data file: ";
      cin>>fileNameChem;
      //ifstream chemical("chemical1.dat" , ios::in);
      ifstream chemical(fileNameChem, ios::in);
      if(! chemical ) {
            cerr<<"Chemical file could not be opened."<<endl;
            exit(1);
      }
      //set all tally to 0
      for(int i=0; i<N_OUTPUTS; i++){
            //cout<<"zeroing tally\n";
            for(int j=0; j<cube; j++){
                  tally[i][j]=0;
            }
      }

      srand(time(0));
      while(raw_data>>x[i]>>y[i]>>z[i]>>err[i]>>x2[i]>>y2[i]>>z2[i]>>err
2[i]){
            chemical>>chem[i];
            i++;
      }
      //normalize error values
      for(int j=0;j<N_TESTS;j++){
            if(err[j] < min)
                  min = err[j];
            if ( err[j] > max)
                  max = err[j];
      }
      for(int j=0;j<N_TESTS;j++){
            //calc normed err
            err[j] = (err[j]-min)/(max-min);
            int loc = int(z[j]*36+y[j]*6+x[j]);

            //add tally
            tally[chem[j]-1][loc]++;

            //determined directed angle

            a_rad[j] = angle_det(x[j],y[j],z[j],x2[j],y2[j],z2[j],
a_rad2[j]);
```

```
            x[j] = x[j] + cos(a_rad[j])*err[j];
            y[j] = y[j] + sin(a_rad[j]) * err[j];
            z[j] = z[j] + sin(a_rad2[j]) * err[j];
            //Output to data file
            final_data<<x[j]<<" "<<y[j]<<" "<<z[j]<<" "<<chem[j]<<endl;
      }
      raw_data.close();
      final_data.close();
      chemical.close();
      } //end data_convert() function
//function prints NN results of clustering
//determines how good a clustering job the network did
void efficiency(){
      //variables
      int max;
      int index;
      double total = 0; //clustered total

      //find biggest cluster for each chemical
      cout<<"Clustering results for network run\n";

      for(int i=0; i<N_OUTPUTS; i++){
            max = -1;
            for(int j=0; j<cube; j++){
                  if(tally[i][j]>0)
                        cout<<"tally["<<i<<"]["<<j<<"] =
"<<tally[i][j]<<endl;
                        if(tally[i][j] > max){
                              max = tally[i][j];
                              index = j;
                        }
                  }
            cout<<"max = "<<max<<" index = "<<index<<" i = "<<i<<endl;
            cout<<"Network clustered "<<max<<" input vectors of";

            switch(i){
               case 0:
                  cout<<" Allylcaproate ";
                  break;
               case 1:
                  cout<<" Methylsalicilate ";
                  break;
               case 2:
                  cout<<" Isoamylacetate ";
                  break;
               case 3:
                  cout<<" Myrcene ";
                  break;
               case 4:
                  cout<<" Decanal ";
                  break;
               case 5:
                  cout<<" Diacetyl ";
                  break;
               }

            cout<<"to cluster ("<<(index%36)%6<<","<<(index%36)/6
```

```
                <<","<<(index/36)<<")\n"<<endl<<endl;
            total = total + max;
      }

      double percent = total/60*100;
      cout<<"\nHighest clustering is "<<percent<<"% of inputs\n";
} //end efficiency
// 4/15/02 change to add saving of second lowest cluster
// 4/17/02 Adding 3rd dimension of clustering
void printOutVector(matrix3 &out, int total[][N_OUTPUTS][N_OUTPUTS],
fstream &data_out, fstream &errs, int finish = 0) {
      // this function will also save error values for non-winning
clusters
      int yold=0, xold = 0, zold = 0; //variable to hold value of y and
x
      int x=0, y=0, z=0;
      int x2=0,y2=0, z2=0;
      double err2=0, err1=0;
      // need to change getAt()
      double dif = out.getAt(x,y,z); //holds current min diff
    for(int i = 0; i<N_OUTPUTS; i++) {
            for(int j=0; j<N_OUTPUTS; j++)  {
                  for (int k=0; k<N_OUTPUTS; k++) {
                        if (finish == 1){
                        errs<<out.getAt(i,j,k)<<"\t";
                        if (j == N_OUTPUTS-1)
                              errs<<endl;
                  }
                  //original array eqn g=
(out.getAt(i)<out.getAt(g))?i:g;
                  yold = y;
                  xold = x;
                  zold = z;
                  y = ( out.getAt(i,j,k) < out.getAt(y,x,z) ) ? i : y;
                  x = ( out.getAt(i,j,k) < out.getAt(yold,xold, zold) )
? j : x;
                  z = ( out.getAt(i,j,k) < out.getAt(yold,xold,zold) ) ?
k : z;
                  err1 = dif; //stores current value
                  dif = (out.getAt(i,j,k) < dif)? out.getAt(i,j,k) :
dif;
                  if(dif < err1){
                        //old dif now 2nd lowest
                        err2 = err1;
                        x2 = xold;
                        y2 = yold;
                        z2 = zold;
                  }
            }
            }
    }
    //printf("[group: %i, %i], dif = %f\n", x+1, y+1, dif);
    total[y][x][z]++;
      // send coordinates of winner and second to output file
    if (finish == 1){
       errs<<endl<<endl;
```

```
        data_out<<x<<" "<<y<<" "<<z<<" "<<dif<<" "<<x2<<" "<<y2<<"
"<<z2<<" "<<err2<<endl;
    }
}

//print inVector prints data vector to standard output
void printInVector(array &in) {
    for(int i = 0; i<N_INPUTS; i++) {
            //printf statement prints only 1 sign digit- rounds input
vector
        printf("%1.0f", in.getAt(i));
    }
    printf(" -> ");
}

void Test(int final = 0) {
    array in(N_INPUTS);
      // Output a 3D matrix
      matrix3 out(N_OUTPUTS, N_OUTPUTS, N_OUTPUTS);
    theApp->net->setPhase(testing);
    A = theApp->net->alphaOf();
    //fstream clusters("clusters.dat", ios::app);
    fstream data_out("data_out.dat", ios::app);
      fstream errs("error_all.dat", ios::app);

    //need array to keep track of all the sets assigned to cluster nodes
    int totals[N_OUTPUTS][N_OUTPUTS][N_OUTPUTS];
    for(int i=0;i<N_OUTPUTS;i++){
            for(int j=0;j<N_OUTPUTS;j++){
                  for(int k=0;k<N_OUTPUTS;k++){
                  totals[i][j][k]=0;
                  }
            }
    }
    printf("\n-------------- Alpha = %f\n", A);
    for (integer n = 0; n< N_TESTS; n++) {
        in.from(Vectors, n*DATA_SIZE);
            //Simulate- check to see if it's ok with out-matrix
        theApp->Simulate(in, out, rad);
            if (final == 1){
                  errs<<"Test array no.:"<<n<<endl;
                  printOutVector(out,totals,data_out,errs,1);
            }
            else
           printOutVector(out,totals,data_out,errs);
      }
    data_out.close();
}

//tests entire data sample through network
void Train() {
    array in(N_INPUTS);
      matrix3 out(N_OUTPUTS, N_OUTPUTS, N_OUTPUTS);
    theApp->net->setPhase(training);
    A = theApp->net->alphaOf();
    for (integer n = 0; n< N_TESTS; n++) {
        in.from(Vectors, n*DATA_SIZE);
```

```
        theApp->Simulate(in, out, rad);
    }
    theApp->net->setAlpha( 0.25 * A);
}
//Main driver program
int main(integer argc, char *argv[]) {
      //internal variables
    theApp = new App(argc, argv);
    A = theApp->net->alphaOf();
      //network weight files
    fstream initWt("initWt.dat", ios::out);
    fstream midWt("midWt.dat" , ios::out);
    fstream fnWt("finalWt.dat", ios::out);
      //Input data file name
      char fileNameIn[30];
      char wttargetfile[20];
      cout<<"Enter the name of the file to convert: ";
      cin>>fileNameIn;
      cout<<"fileName = "<<fileNameIn<<endl;

      //ifstream from("readydata1.dat" , ios::in);
      ifstream from(fileNameIn, ios::in);
      if(! from ) {
            cerr<<"Input file could not be opened."<<endl;
            exit(1);
      }

    // This file will hold target weights for semi-supervision
    //fstream super("wt_targets1.txt", ios::in);
      cout<<"Enter the name of the weight target file: ";
      cin>>wttargetfile;
      cout<<"fileName = "<<wttargetfile<<endl;
      fstream super(wttargetfile, ios::in);
      //send entire weight matrix to file
    theApp->net->saveToFile(initWt);
    from >> Vectors;
    //Here assign weights to connections between inputs and 6 outputs
    // semi-supervision
    update_wts(super);
    theApp->net->saveToFile(midWt);
    // neighborhood change variable
    int k;
    while(A > 0.00001) {
            k= rchange;
            while(k > 0 ) {
                  Train();
                  Test();
                  k--;
            }
    // reduce size of neighborhood radius
    // only until rad = 0
    while(rad > 0 )
          rad--;
    }
    Test(1);
    theApp->net->saveToFile(fnWt);
    delete theApp;
```

```
    initWt.close();
    fnWt.close();
    midWt.close();
    from.close();
    super.close();
    //convert data to graphable data
    dataConvert();
    //Determine efficiency of clustering
    efficiency();
    return 0;
}

//function to determine directed angle for error visualization
// includes 3D implementation
// first angle depends only on x and y
// second angle depends only on y and z- makes it easier
double angle_det(double x, double y, double z, double x2, double y2,
double z2, double ang2_rad)
      {
      int s;
      double ang, ang_rad, ang2;
      //choose random number once
      s = rand()%1000;
      // first angle
      // nested if statements
      if (x2 < x){
            //not in regions 8,1,3,7,or,2
            if (y2 > y){
                  // region 4 angle picked from btw 100-170
                  ang = (s/1000*(170-100))+100;
            }
            else if (y2 == y){
                  // angle btw 170-190
                  ang = (s/1000*(190-170))+170;
            }
            else {
                  //angle btw 190-260
                  ang = (s/1000*(260-190))+190;
            }
      }
      else if (x2 == x){
            // in region 3 or 7
            if (y2 > y){
                  //region 3, angle 80-100
                  ang = (s/1000*(100-80))+80;
            }
            else {
                  //region 7, angle 260-280
                  ang = (s/1000*(280-260))+260;
            }
      }
      else {
            //regions 8,1,and 2
            if (y2 < y){
                  //region 88, angle 280-350
                  ang = (s/1000*(350-280))+280;
            }
```

```
            else if (y2 == y){
                  //region 1, angle -10-> 10
                  ang = (s/1000*(10+10))-10;
            }
            else {
                  //region 2, angle 10-80
                  ang = (s/1000*(80-10))+10;
            }     }
      ang_rad = ang * PI/180;
      // second angle
      if (z2 < z){
            // direction of error goes down
                  ang2 = (s/1000*(360-180))+180;
            }
      else if (z2 == z){
            // same plane
                  ang2 = 0;
            }
      else {
            //z2>z, error goes up
                  ang = (s/1000*180);
            }
      ang2_rad = ang2 * PI/180;
      return ang_rad;
} //end angle_det()


void update_wts(fstream& infile)
      {
      // update weights for N_OUTPUTS
      // 9/26/02 for this case it's 6 outputs
      // at this point use preselected outputs
      // they will be [2,4,0],[4,1,0],[3,1,2],[0,5,5],[1,4,1],[5,0,3]
      // 11/18/02  winning nodes assos with enose winners
      // nodes numbered 0..215
      int node[6]= {22, 9, 80, 205, 57, 113};
      double inwt;
      double wts[50];
      // read in weights into array for 1st node
      for (int i=0;i<32;i++)
         {
         infile>>inwt;
         wts[i]=inwt;
         }

      //Start loop to change weights of specific nodes
      int j;
      for (j=0; j< 6; j++)
         {
         // send parameters to links
         theApp->net->update_wts(wts, node[j]);

         // get next array of wts from file
         for (int i=0;i<32;i++)
         {
         infile<<wts[i];
         }
```

```
        }
} // end update_wts()
```

BIBLIOGRAPHY

[B] Brereton, R.G., *Chemometrics- Applications of Mathematics & Statistics to Laboratory Systems*, Ellis Horwood, 1990.

[BEG] Bartlett, Philip N., Elliot, Joe M., Gardner, Julian W., "Electronic Noses and Their Application in the Food Industry", *Food Technology*, Vol. 51, No. 12, pp. 44-48, Dec. 1997.

[BPS] Beebe, K.R., Pell, R.J., Seasholz, M.B., *Chemometrics- A Practical Guide*, John Wiley & Sons, Inc. 1998

[CCM] Cohn, D., Caruana, R., McCallum, A., Semi-Supervised Clustering with User Feedback, AAAI 2000, DRAFT submission.

[Cy1] Cyrano Sciences, Inc, *The Cyranose 320 Electronic Nose User's Manual*,Cyrano Sciences, Inc., Edition 3, Revision C, November 2000.

[Cy2] Cyrano Sciences, Inc., *The Practical Guide to the Cyranose 320™*, Cyrano Sciences, Inc., Revision B, November 2000.

[DWH] Dimitriadou, E., Weingessel, Hornik, A., K., "A Mixed Ensemble Approach for the Semi-Supervised Problem", *ICANN 2002*, Madrid Spain, August 2002.

[F] Fausett, Laurene. *Fundamentals of Neural Networks- Architectures, Algorithms, and Applications*, Prentice Hall, 1999

[Faq] www.faqs.org, "What are Cross-Validation and Bootstrapping?", http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-12.html, visited 2/17/03.

[Fr] Franz, M.P., "A Portable, Object-Oriented Library for Neural Network Simulation", Masters Thesis, Computer Science Department, Western Michigan University, 1998.

[HKKK] Hashem, S., Keller, P.E., Kouzes, R.T., Kangas, L.J., "Neural Netork Based Data Analysis for Chemical Sensor Arrays", Proceedings of the SPIE, (Ed.s) Rogers, S.K., Ruck, D.W., Vol. 2492, No. 5, pp. 33-40., 1995, www.emsl.pnl.gov:2080/proj/neuron//papers/hashem.spie95.abs.html

[K] Paul E. Keller, "Mimicking Biology: Applications of Cognitive Systems to Electronic Noses", *IEEE International Symposium in Intelligent Control /Intelligent Systems and Sembiotics (ISIC/ISAS'99)*, Cambridge, MA, USA,

http://www.emsl.pnl.gov:2080/proj/neuron//papers/keller.isic99.html (visited Oct. 1999)

[KKK] Keller, P. E., Kouzes, R.T., Kangas, L.J., "Three Neural Network Bases Sensor Systems for Environmental Monitoring", Electro/94 International Combined Conference Proceedings, Miller Freeman, Inc., Dallas, TX, USA, 1994, pp. 378-382, www.emsp.pnl.gov:2080/proj/neuron/papers/keller.electro94.htm

[Ko] Kohonen, T., *Self-Organization and Associative Memory*, Second Edition, Springer-Verlag, 1988.

[Ko2] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics* 43:59-69, 1982.

[Li] Li, Jing, "On Signal Processing- The Brains of the Cyranose 320", Cyrano Sciences, Inc., June 2001, http://cyranosciences.com/technology/onboard.html (visited Jan 2003)

[MP] Minsky, M.L., Pappert, S.A., *Perceptrons*, Expanded Edition. Cambridge, MA, M.I.T. Press, 1988, Original Edition 1969.

[P] Piatkowski, Thomas F., "Citation and Acknowledgement Guide", 2000, http://www.cs.wmich.edu/~piat/citationAckGuideAbstract.html .

[S] Severin, Erik, "Cyrano Sciences' Sensor Technology- The Heart of the Cyranose 320 Electronic Nose", Cyrano Sciences, Inc., June 2001, http://cyranosciences.com/technology/sensor.html (visited Jan 2003)

[S-S] Sun-Sentinal, "The Electronic Nose Knows", Sunday, August 2, 1998.

[Sa] Sawyer, A., "Electronic Nose Sniffing Out Wine Niche- Perfecting Sensory Evaluation of Smells by Imitating Canine 'Noses with Legs'", Wine Business Online, July 1997. http://winebusiness.com/html/SiteFrameSet.cfm?fn=..Archives/Monthly/1997/9707/bmg9746.htm

[SHG] Singh, S., Hines, E.L., Gardener, J.W., "Fuzzy Neural Computing of Coffee and Tainted Water Data From an Electronic Nose", *Sensors and Actuators B*, Vol. 30 (1996) pgs 185-190.

[SCK] X. Sun, R. Collins, J. Kim, "A Comparison of SOM Neural Networks and K-means clustering using real world data: Chinese Consumer Attitudes Towards Imported Fruit", *Acta Hort.*, 566, ISHS 2001, pgs 185-191.

[Si] Peter Sinton, "The Nose Knows", San Francisco Chronicle, Dec. 2, 1997.

[Th] Thermogalactic.com, "Discriminant Analysis, The Mahalanobis Distance", www.galactic.com/Algorithms/discrim_mahaldist.htm, visited 1/11/03