

Victor José de Almeida e Sousa Lobo

SHIP NOISE CLASSIFICATION

A contribution to prototype based classifier design

Dissertation presented to obtain the degree
of Doctor in Informatics by the New
University of Lisbon, College of Science and
Technology.

Supervisor: Prof.Dr. Fernando Moura Pires

Co-supervisor: Prof.Dr. Roman Swiniarski

Lisbon

2002

to Beatriz

Deo Omnis Gloria

Acknowledgments

Although a PhD program is supposed to be a personal accomplishment, I feel that this one was really a team effort of many people.

I would like to express my great gratitude, admiration, and friendship towards my supervisor, Prof.Dr. Fernando Moura Pires. He was extremely patient, spending long hours discussing and working with me, sometimes showing more faith than myself in the success of the project. I am also in dept towards my co-supervisor Prof.Dr. Roman Swiniarski. Thanks to him I spent several months at San Diego State University, opening my eyes to a New World, that changed me forever. His attention to detail and the many discussions and corrections of the manuscript had an enormous impact on the final text of the thesis.

I must also give a very special acknowledgement to Commander Paulo Mónica de Oliveira and Eng.Nuno Bandeira, who had a very direct impact on my thesis and could be considered my unofficial co-supervisors. Having done his PhD on signal processing, Commander Mónica de Oliveira helped me with all signal processing aspects of my work, and being head of department at the Naval Academy gave me all the support possible. More than that, his advice as a very dear friend changed the way I work and think about many things. Eng. Nuno Bandeira is, besides the author, the person that spent more time working on problems related to this thesis, having written most of the code of DSOM. As such, he was one of the first persons to fully understand the concepts of Q-set simplifications. My many discussions with him were a wonderful experience by themselves and an invaluable help to this thesis.

The support of all my family, both directly and indirectly was the basis for all my work. I must thank my wife Sara for all the extra work she has had so that I could finish this thesis, and my daughter Beatriz for being so understanding. My father has always given me an example of true academic spirit, and personally reviewed my thesis. My mother gave all the support that only a mother can give. And my brothers and sisters-in-law, as well as my parents-in-law always supported my work with their friendship and help. Almost like family I must also thank Carol Bourke, that “took care” of me and gave me an excellent working environment while I stayed in San Diego. Her enthusiasm and support were a wonderful, if unexpected, blessing.

I am in dept to San Diego State University that received me as a visitor, and the faculty from the Computer Science and Mathematics Departments that welcomed and helped me. In particular, I would like to thank with all my heart Marko Vuskovic and his wife Anna, Marie Roch, Roxana Smarandache, Sara Baase and Keith, John Elwin, and Renee Caprice.

During these last 6 years, there were many contributions by colleagues and professors from Universidade Nova de Lisboa, from Instituto Superior Técnico, and from other researchers that I met at conferences. Of these, I must mention Doctor João Paulo Marques da Silva, who introduced me to the satisfiability problem, and gave me most of the references I used in that area.

Most of my work was carried out while working at the Naval Academy. Besides thanking those that have been in charge of this century old school for all their support, I must explicitly thank Professors Cruz Serra, Afonso Barbosa, Alves Moreira and Pedro Girão, Commanders Ferreira Neto, Sadanha Carreira and Miguel Policarpo, Sargents Natario, Gaspar, Nicolau, Mateus, Paulino, and Gertrutes, and more recently Lieutenant Veloso for their support. I must also thank the personnel from the acoustic tank, namely Mr. Rêgo, Brás e Silva, Ferreira da Costa, Victor Santos, and Tito Matias.

I must also thank INVOTAN for giving financial support for my first visits to San Diego State University, Fundação Gulbenkian for financing one of my trips to a conference, and very specially FLAD for it's very generous support, available despite very short notice.

Finally, I would like to thank all my friends, particularly those from "AnTUNiA", from the University Choir, from the Department of Computer Science of FCT/UNL, and from the Naval Academy, for the relaxing moments that I enjoyed with them.

Table of contents

LIST OF FIGURES	XI
LIST OF TABLES	XVII
LIST OF ALGORITHMS.....	XVIII
GLOSSARY	XIX
ABSTRACT	XXIII
PART I – STATE OF THE ART.....	1
1 - GENERAL OVERVIEW.....	3
2 - FEATURE EXTRACTION	9
2.1 – INTRODUCTION.....	9
2.2 - FOURIER TRANSFORMS.....	11
2.3 - OTHER FREQUENCY BASED TECHNIQUES	13
2.4 - PRINCIPAL COMPONENT ANALYSIS AND RELATED TECHNIQUES	15
3 - FEATURE SELECTION	19
3.1 – INTRODUCTION.....	19
3.2 - SCATTER MATRICES	21
3.3 - ROUGH SETS	23
4 - EXPLORATORY DATA ANALYSIS.....	31
4.1 – INTRODUCTION.....	31
4.2 – K-MEANS CLUSTERING.....	34
4.3 – SELF ORGANIZING MAPS (SOM)	37
5 - CLASSIFIER DESIGN	77
5.1 – INTRODUCTION.....	77
5.2 - CLASSIFIERS	79
5.3 - NEAREST NEIGHBOR CLASSIFIERS	80
5.4 – VARIATIONS ON NEAREST NEIGHBOR OR PROTOTYPE BASED SYSTEMS.....	82
5.5 – OTHER RESEARCH ON NEAREST NEIGHBOR RELATED PROBLEMS	88
5.6 - PROTOTYPE MINIMIZATION.....	89
6 - VALIDATION	113

6.1 – INTRODUCTION.....	113
6.2 – KNOWN SETS, TRAINING SETS, VALIDATION SETS, AND TEST SETS.....	114
6.3 – ERROR RATE ESTIMATES	116
6.4 – CONFUSION MATRICES	119
PART II – ORIGINAL CONTRIBUTIONS.....	121
1 - Q-SETS: A BOOLEAN FORMALIZATION FOR MINIMIZING PROTOTYPE-BASED CLASSIFIERS.....	123
1.1 - INTRODUCTION.....	123
1.2 - INFORMAL PRESENTATION OF THE THEORY.....	124
1.3 - THEORETICAL FRAMEWORK.....	127
1.4 - POSITIVE-ONLY Q -FUNCTIONS.....	133
1.5 – GENERAL CASE.....	138
1.6 - COMPARISON WITH OTHER METHODS.....	143
1.7 – EXTENSIONS AND APPLICATIONS OF Q-SET THEORY.....	151
2 - BINARY SELF-ORGANIZING MAP - BSOM	161
2.1 – INTRODUCTION.....	161
2.2 - BINARY SOM ALGORITHM	162
2.3 – RESULTS OBTAINED WITH BSOM	165
2.4 – OTHER WORK	166
3 - PARALLEL IMPLEMENTATION OF SOM OVER PVM.....	169
3.1 – INTRODUCTION.....	169
3.2 - DISTRIBUTED SOM ALGORITHM.....	170
3.3 - EXPERIMENTAL RESULTS	173
3.4 - CONCLUSIONS.....	175
PART III - APPLICATION.....	177
1 - SHIP NOISE AND TARGET IDENTIFICATION.....	179
1.1 – INTRODUCTION.....	179
1.2 – THE BASIC PROBLEM	180
1.3 - SOUND GENERATED BY SHIPS.....	184
1.4 - TRANSMISSION OF SOUND TO THE SONAR EQUIPMENT	190

1.5 – PREVIOUS WORK IN THIS AREA	195
2 - THE SOFTWARE USED	203
2.1 – INTRODUCTION.....	203
2.2 – THE DSOM PROGRAM.....	204
2.3 – MATLAB ROUTINES	214
2.4 - OTHER SOFTWARE	218
3 - THE SUBMARINE DATA	221
3.1 - FIRST EXPERIMENTS.....	222
3.2 – USE OF DSOM AND DISTRIBUTED PROCESSING.....	223
3.3 – BROADBAND VS TONAL IDENTIFICATION	224
3.4 – CLUSTERING ON A LARGE DATASET	228
4 - ACOUSTIC TANK DATA	233
4.1 - INTRODUCTION.....	233
4.2 - DATASETS AND EXPERIMENTS	242
APPENDIX A - EXPERIMENTS WITH HART’S DOUBLE F PROBLEM	253
APPENDIX B - EXPERIMENTS WITH THE STRAIGHT LINE PROBLEM.....	263
APPENDIX C - LIST OF DATA RECORDED IN THE ACOUSTICAL TANK.....	277
APPENDIX D - OVERVIEW OF THE SIGNALS RECORDED IN THE ACOUSTIC TANK.....	283
APPENDIX E - MATLAB ROUTINES	291
REFERENCES.....	339

List of figures

Figure 1 - General overview of the classification problem. Spiked shapes represent data in various forms.5

Figure 2 - Feature extraction/selection that produce data with these distributions will lead to a good neural network classifier, but a bad decision tree classifier, since these have decision boundaries parallel to the axis.6

Figure 3 - Example of a universe U partitioned by a set of attributes Q , and by a subset A of these attributes. The set X (known as concept or class), that was an exact set using Q , becomes a rough set using A26

Figure 4 - Example of possible pitfalls of the k-means algorithm. In the situation presented on the left, the sum of square distances criteria will correctly position the centroids at the center of the clusters. However in the situation presented on the right, that criteria does not provide satisfactory results.36

Figure 5 - Basic structure of a Self-Organizing Map (SOM)38

Figure 6 - Example of a 2-dimensional SOM mapping 3-dimensional patterns. On the top, patterns are represented by "-", and are distributed around some of the vertices of the cube. The SOM units are represented in the input space by black balls, with lines showing their neighbors in the output space. On the bottom, we can see the layout of units in the output space, forming a regular grid. On the left, a 2x2 SOM was used, while a 4x4 was used on the right.39

Figure 7 - Example of the unfolding of a 1-dimensional SOM (a line) (Kohonen 1995), to fit a set of points uniformly distributed within a triangular area. The small numbers represent the number of the iteration at which the snapshot was taken.40

Figure 8 - Example of a 2D to 2D mapping of a uniform distribution of points in a square (Mathworks 2001), (Kohonen 1995). Note that after training the units of the SOM are a faithful representation of the original distribution. This is possible because it was uniform, and its dimensionality was the same as the SOM's.41

Figure 9 - Example of an unfolded SOM. This map represents the same problem as the one in Figure 4, but due to a bad choice of initial radius and learning rate, the map did not unfold smoothly, and got stuck in a local minima.41

Figure 10 - Positions of the SOM units and U-Mat units in the output space. On the left, it is shown how the U-Mat values are computed for the 3 types of units: those that are located between SOM units, on SOM units, and on the diagonals.48

Figure 11 - Example of a 3D representation of a U-Mat, taken from (Guimarães and Urfer 2000). The central cluster is clearly separated from the rest of the map by a high ridge, and the white line represents a succession of states present in a certain patients data.....	48
Figure 12 - Example of cluster identification with a U-Matrix. 360 3-dimensional data points, centered at 6 corners of a unit cube (on the left) are mapped into 6 distinct areas separated by dark dividing lines (on the right).....	49
Figure 13 - A possible taxonomy for temporal SOMs.....	50
Figure 14 - Temporal Sequence processing with a tapped delay as input for a SOM.....	52
Figure 15 - Temporal sequence processing using time-related transformations as pre-processing for the SOM.....	52
Figure 16 - Structure of a trajectory based SOM.....	54
Figure 17 - Structure of the Kangas Map.....	61
Figure 18 - Structure of each unit in a Temporal Kohonen Map (TKM)	64
Figure 19 - Structure of each unit in a recurrent SOM	65
Figure 20 - Structure of the Recursive SOM	66
Figure 21 - Structure of each unit in a SOMTAD based map	67
Figure 22 - Structure of a Hierarchical SOM.....	69
Figure 23 – Hart’s problem: two classes, each with 200 patterns, with uniform distribution in the "F" shapes given.....	95
Figure 24 - Comparison of NN and CNN for Hart’s problem. In the right figure, only 47 of the original 400 patterns were selected as classifiers.....	96
Figure 25 - Comparison of NN and RNN for Hart’s problem. In the right figure, only 29 of the original 400 pattern were selected as classifiers.	97
Figure 26 - Example of a Voronoi tessellation, defined by a set of 2-dimentional prototypes represented by points.....	102
Figure 27 - Different data sets involved in the classification process	115
Figure 28 - Example of a Q-set for a 2-dimensional problem. The center cross represents the patterns for which the Q-set is being calculated. Crosses represent prototypes with the same class as the pattern, and circles represent prototypes with a different class. The white area represents the Q-set, containing 3 prototypes.....	125
Figure 29 - Example of a Q and R sets for a 2-dimensional problem. The center cross represents the patterns for which the sets are being calculated. Crosses represent prototypes with the same class as the pattern, and circles represent prototypes with a different class. The white areas represent the Q -sets, while the gray represent the R -sets.	130

Figure 30 - Hart's Double F problem. Class 1 has a uniform distribution in the rightmost F shape, while class 2 has the same type of distribution in the leftmost, inverted, F shape.	144
Figure 31 - Number of prototypes used for Hart's double F problem.....	145
Figure 32 - Error rate for Hart's double F problem.	145
Figure 33 - Training time required for Hart's double F problem.	146
Figure 34 - Number of prototypes used for the straight line problem.....	150
Figure 35 - Error rate for the straight line problem	150
Figure 36 - Training times for the straight line problem. Note that the time axis uses a logarithmic scale to be able to show very different training times. Due to this, when the training time is too close to zero (as is the case for QSET-BB, CNN, RNN and QSET-P when the training set has fewer than 16 patterns), the value is not represented in this graph.	151
Figure 37 - Same data used for prototypes and for selection.....	152
Figure 38 - Separate data for prototypes and selection.....	153
Figure 39 - Q -sets used a pruning technique	154
Figure 40 - Q -sets as pre-processing.....	155
Figure 41 - The 10x5 unit SOM trained with binary data. Each shade corresponds to a different type of ship	165
Figure 42 - Message exchange in distributed SOM.....	172
Figure 43 - Diagonal distribution of units amongst different processors	174
Figure 44 - Absolute execution times	174
Figure 45 -Relative execution time (1=time on a single machine).....	175
Figure 46 - General description of the process of noise generation, transmission, and capture by a passive sonar.....	181
Figure 47 - Typical frequency ranges of different sources of ship noise (Collier 1998).	184
Figure 48 - Acoustic power radiated by a ship (Urick 1982)	185
Figure 49 - Propulsion and auxiliary systems, and the fundamental frequency of noise generated by them (Collier 1998).....	186
Figure 50 - Typical power spectra of ship generated noise, and its change with speed.	189
Figure 51 - Typical ambient noise levels (NATO 1993)	191
Figure 52 - Typical bathythermic profiles (Apel 1990).....	192
Figure 53 - Main window of the DSOM program.	208
Figure 54 - The main Pattern Window	211
Figure 55 - Spectra visualization from the Pattern Window.....	212

Figure 56 - Training dialog box.....	213
Figure 57 - Example of a SOM obtained with the first experiments with the submarine data. Each name (alfa to echo) corresponds to a different ship or class of ships. Echo and Foxtrot are two very similar types of torpedoes.....	223
Figure 58 - Binary patterns obtained	225
Figure 59 - A 10x5 unit SOM trained with binary data. Each shade corresponds to a different type of ship	227
Figure 60 - The U-Matrix obtained after clustering the 33 ship dataset with a SOM.	229
Figure 61 - Various aspects of the acoustic tank. Note the sliding bridge and the outboard motor fixation on the top photographs, and in the bottom ones, the acoustical isolation visible when the tank was emptied.....	235
Figure 62 - Bruel & Kjaer 8104 passive omnidirectional hydrophone	236
Figure 63 - Bruel 2636 amplifier	236
Figure 64 - Motor 1, a 4.5 hp Mercury	238
Figure 65 - Motor 4, a 3.6 hp Mercury	238
Figure 66 - Motor 3, a 3.6 hp Yamaha.....	238
Figure 67 - Various aspects of the electric model boat (motor 5).....	238
Figure 68 - Transients <i>c</i> and <i>d</i> : hitting metal tubes	240
Figure 69 - Transient <i>a</i> : bursts of compressed air.....	240
Figure 70 - Transient <i>b</i> : splashing water.....	241
Figure 71 - SOM with 20x15 units trained with all the patterns of dataset 1, and the corresponding U-matrix. For the unfolding phase we used $\alpha=0.2$, $r_{init}=18$, and 10 iterations through the dataset. For the second, we used $\alpha=0.05$, $r_{init}=8$, and 100 iterations through the dataset.	244
Figure 72 - SOM with 40x30 units trained with all the patterns of dataset 1, and the corresponding U-matrix. For the unfolding phase we used $a=0.2$, $r_{init}=38$, and 10 iterations through the dataset. For the second, we used $a=0.05$, $r_{init}=12$, and 100 iterations through the dataset.....	244
Figure 73 - SOM with 20x15 units trained with all the patterns of dataset 1 using reduced features, and the corresponding U-matrix. For the unfolding phase we used $\alpha=0.2$, $r_{init}=18$, and 10 iterations through the dataset. For the second, we used $\alpha=0.05$, $r_{init}=8$, and 100 iterations through the dataset.....	247

Figure 74 - SOM with 20x15 units trained with all the patterns of dataset 2, and corresponding U-matrix. For the unfolding phase we used $a=0.2$, $r_{init}=18$, and 10 iterations through the dataset. For the second, we used $a=0.05$, $r_{init}=8$, and 100 iterations through the dataset	248
Figure 75 - SOM with 40x30 units trained with all the patterns of dataset 2, and corresponding U-matrix. For the unfolding phase we used $a=0.2$, $r_{init}=38$, and 10 iterations through the dataset. For the second, we used $a=0.05$, $r_{init}=12$, and 100 iterations through the dataset.....	248
Figure 76 - Hart's Double F problem. Class 1 has a uniform distribution in the rightmost F shape, while class 2 has the same type of distribution in the leftmost, inverted, F shape.....	254
Figure 77 - Borders between classes in the double F problem using 100 training patterns	255
Figure 78 – Borders between classes in the double F problem using 200 training patterns a	256
Figure 79 – Borders between classes in the double F problem using 400 training patterns	257
Figure 80 – Borders between classes in the double F problem using 800 training patterns	258
Figure 81 – Borders between classes in the double F problem using 1600 training patterns	259
Figure 82 - Borders between classes in the double F problem, using 1600 patterns	260
Figure 83 - Borders between classes in the double F problem, using 3200 patterns	261
Figure 84 - The straight line problem. Patterns belonging to class 1 are represented by "+", and patterns belonging to class 2 by "x"	264
Figure 85 – Borders for the straight line problem, with a training set of 4 prototypes.....	265
Figure 86 – Borders for the straight line problem, with a training set of 8 prototypes.....	266
Figure 87 – Borders for the straight line problem, with a training set of 16 prototypes.....	267
Figure 88 – Borders for the straight line problem, with a training set of 24 prototypes.....	268
Figure 89 – Borders for the straight line problem, with a training set of 32 prototypes.....	269
Figure 90 – Borders for the straight line problem, with a training set of 40 prototypes.....	270
Figure 91 – Borders for the straight line problem, with a training set of 48 prototypes.....	271
Figure 92 – Borders for the straight line problem, with a training set of 56 prototypes.....	272
Figure 93 – Borders for the straight line problem, with a training set of 64 prototypes.....	273
Figure 94 – Borders for the straight line problem, with a training set of 96 prototypes.....	274
Figure 95 – Borders for the straight line problem, with a training set of 128 prototypes.....	275
Figure 96 – Borders for the straight line problem, with a training set of 256 prototypes.....	276

Figure 97 - Spectra of Motor 1. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).....	285
Figure 98 -Spectra of Motor 2. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).....	286
Figure 99 -Spectra of Motor 3. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).....	287
Figure 100 - Spectra of Motor 4. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).....	288
Figure 101 - Spectra of background noise (or motor 5). The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).	289

List of tables

Table 1 - General conventions	xix
Table 2 - Names of techniques, sets, and algorithms.....	xix
Table 3- Overview of the approaches used in 68 different papers.....	75
Table 4- List of some papers that compare prototype minimization techniques	111
Table 5 - Smallest size of consistent subsets obtained for the Iris data.....	111
Table 6- Example of a confusion matrix. Numbers on the diagonal correspond to correctly classified patterns. In this case, it clear that class C is correctly classified, but there are errors in distinguishing class A from class B.	120
Table 7 - Leave-one-out cross-validation for the Iris Dataset. Together with the error rate, the actual number of errors is shown in parenthesis	148
Table 8 - Proposed solutions	165
Table 9 - Execution times (in seconds).....	175
Table 10- Error rates in the training sets.....	226
Table 11 - Error rates in the test sets.....	226
Table 12 - General information about the Acoustic Tank data. The number of patterns correspond to 3s segments of the original signal. These will later be subject to different feature extraction techniques, to produce the final patterns.	242
Table 13 - Results of cross-validation on dataset 1.	245
Table 14 - Results of cross-validation on dataset 1, using small training sets.....	246
Table 15 - Features selected from dataset 1, using scatter matrices.	246
Table 16 - Results of cross-validation on dataset 1 with reduced features.	247
Table 17 - Results of cross-validation on dataset 1 with reduced features, using small training sets.	247
Table 18 - Results of cross-validation on dataset 2.	249
Table 19 - Results of cross-validation on dataset 2, using small training sets.....	250
Table 20 - Reducts for dataset 2 produced by Roughsetlab, using 10 levels of discretization.	251
Table 21- Results of cross-validation on the reduced dataset 2.....	251
Table 22- Results of cross-validation on the reduced dataset 2, using small training sets	252
Table 23 - General information about the Acoustic Tank data. The number of patterns correspond to 3 s segments of the original signal. These will later be subject to different feature extraction techniques, to produce the final patterns.	284

List of algorithms

Algorithm 1 - Original k-means clustering	35
Algorithm 2 - Batch k-means clustering	35
Algorithm 3 - SOM training algorithm (for a 2-dimensional map)	42
Algorithm 4 - Nearest Neighbor Classification Rule	80
Algorithm 5 - Building Condensed Nearest Neighbors set (CNN)	94
Algorithm 6 - Building the Reduced Nearest Neighbor set (RNN).....	97
Algorithm 7 – Chang’s Algorithm	99
Algorithm 8 – Selective Nearest Neighbors	101
Algorithm 9- Computing Positive-only Q-sets	136
Algorithm 10 - Qset Heuristic for selecting prototypes	137
Algorithm 11 - Computing the complete Q-sets	141
Algorithm 12 - G2P - General to Positive	142
Algorithm 13 - Algorithm 13 - The distributed SOM algorithm.	171

Glossary

Throughout the thesis we have tried to use a coherent nomenclature that is summarized in Table 1 and

Table 2. Generally, italic is used when referring to variables, and they will be in bold if they are vectors or sets, and normal if they are scalar

Table 1 - General conventions

Sets of patterns are in italic and start with uppercase.	<i>Example_set</i>
Members of Sets of patterns are in italic, and have indexes in parenthesis.	<i>Example_set(3)</i>
Patterns are in bold italic, and start with lowercase.	<i>example_pattern</i>
Components of individual patterns are represented by their index number in superscript.	<i>example_pattern</i> ⁴
Component planes (i.e. the same component of all individual patterns) of a set of patterns, are represented by their index number in superscript over the name of the set.	<i>Example_set</i> ³

Table 2 - Names of techniques, sets, and algorithms

A_set/	Cardinality of “A_set”.
<i>a priori error</i>	Maximum error that might occur when using the positive only Q-set approach.
AMER	Acceptable maximum error rate – Maximum a priori error rate we are willing to accept when using the general case Q-set heuristic.
CB	Cost/Benefit ratio of a pattern used in the general case Q-set heuristic.
CNN	Condensed Nearest Neighbor (Hart 1968) – A prototype minimization technique, , for prototype based classifiers.
CNF	Conjunctive Normal Form - A representation of a Boolean function as product of sums.
DNF	Disjunctive Normal Form – A representation of a Boolean function as sum of products.

DMCNN	Devi modified CNN - Prototype selection method, for prototype based classifiers (Devi and Murty 2002).
DSM	Decision Surface Mapping – A method that finds prototypes close to interclass borders (Geva and Sitte 1991).
DYNAGEN	Prototype selection method, for prototype based classifiers (Laha and Pal).
G2P	General to positive only – algorithm used to convert a general case complete Q-set to a positive only Q-set.
GA	Genetic Algorithm (Fogel 1999) - An optimization technique. It can be used to for prototype minimization, for prototype based classifiers.
GLVQ	Generalized LVQ – A variation on the LVQ neural network.
GLVQ-F	Fuzzy generalized LVQ (Karayiannis, Bezdek <i>et al.</i> 1996) – A variation on the LVQ neural network.
Hastie-Stuetzle Algorithm	Algorithm for finding principal curves (Hastie and Stuetzle 1989) (Chang and Ghosh 2001). Related to PCA and SOM..
ICA	Iterative Condensation Algorithm – Prototype minimization technique (Swonger 1972).
ICA	Independent Component Analysis – Data transformation technique, e.g. (Hyvarinen and Oja)
LVQ	Linear Vector Quantization (Kohonen 2001) – A type of neural network.
LVQ-H	Huang's modified LVQ (Huang, Chiang <i>et al.</i> 2002) – A variation on the LVQ neural network
MCS	Minimal Consistent Subset
MNV	Mutual Neighborhood Value – A prototype minimization technique, for prototype based classifiers (Gowda and Krishna 1979).
MSS	Minimal Selective Subset
MultiEdit	A data editing technique, used to improve the performance of prototype-based classifiers (Devijver and Kittler 1982).
RCNN	Reduced Complexity Nearest Neighbor -An algorithm for building fast nearest neighbor classifiers (Lee and Chae 1998).
RISE	Rule Induction from a Set of Exemplars - A case-based reasoning system that unifies instance based learning with rule induction (Domingos 1995).
RNN	Reduced Nearest Neighbor (Gates 1972) – A prototype minimization

	technique, , for prototype based classifiers.
RS	Random Selection – A technique based on random selection of entities. It can be used to for prototype minimization, for prototype based classifiers (Kuncheva and Bezdek 1998).
SA	Simulated Annealing (Kirkpatrick, Gelatt Jr. <i>et al.</i> 1983) – An optimization technique. It can be used to for prototype minimization, for prototype based classifiers.
SNN	Selective Nearest Neighbors (Ritter, Woodruff <i>et al.</i> 1975) – A prototype minimization technique, , for prototype based classifiers
SOM	Self Organizing Map (Kohonen 2001). A data visualization, quantization, and mapping algorithm.
SVM	Support Vector Machines – A classifier design technique (Vapnik 2000).
TS	Tabu search (Glover and Laguna 1997) – An optimization technique. It can be used to for prototype minimization, for prototype based classifiers.
X_{CNN}	CNN classification set
X_{RNN}	RNN classification Set
X_{train}	Training Set (of patterns)

Abstract

The main objective of this thesis is to construct a system that can identify ships using the noise they produce underwater. In the process, a few contributions are made to prototype based classifier design.

This thesis is divided into three parts. We begin by presenting a state of the art of relevant topics, then we propose a few original contributions, and finish by presenting the results of the application of previously discussed techniques to a specific problem.

Part I contains an overview and state of the art on pattern classification. Particular emphasis is given to techniques that were applied during our experimental phase and to issues more closely related to our contributions. A brief introduction to the global problem is presented in Chapter 1, where relations among various phases of classifier design are laid out. Chapter 2 overviews some of the most common techniques used for feature extraction, such as frequency based transforms and principal component analysis. Chapter 3 deals with feature selection techniques, including the use of scatter matrices, and the use of Rough Sets. In Chapter 4 we review the use of k -means clustering and Self-Organizing Maps for exploratory data analysis. Chapter 5 is the largest as it addresses the main issue which is classifier design. While other classification methods are mentioned, most of the chapter covers nearest neighbor classifiers and a thorough review of prototype minimization techniques is presented. Finally, Chapter 6 reviews the issue of cross-validation and clarifies the meaning of training, test, and validation sets. The review of two very specific issues (the use of Self Organizing Maps for binary patterns, and the parallelization of the Self Organizing Map algorithm) is postponed to the last chapters of Part II.

Part II contains the core of this thesis describing its original contributions. The first chapter deals with the main contribution, which is a method for minimizing the number of prototypes necessary for a nearest neighbor classifier. A framework is proposed, named Q-set theory, which relies on a Boolean function formalization of the classifier minimization problem. Two distinct algorithms are presented, that use Q-set concepts and heuristics to achieve that minimization. An example of how Q-set theory allows the problem to be solved exactly with existing optimization algorithms it is also shown. A comparison with other algorithms is performed, with standard benchmark datasets, and possible extensions proposed. The next two chapters contain two minor

original contributions. Chapter 2 presents an adaptation of the Self-Organizing Map algorithm for clustering binary valued data. The problems associated with that adaptation are presented, solutions are proposed, and a brief description of its performance is done. Chapter 3 presents a parallel implementation of Self-Organizing Maps using common networked PC computers.

Part III describes the application of the classifier techniques developed here to the specific problem of this thesis: the classification of underwater sound. Chapter 1 overviews the issues that, while not directly related to computer science, are relevant to the problem. Chapter 2 describes the software tools developed. Chapter 3 describes the classification of data obtained by operational submarines. Chapter 4 describes the classification of acoustic data obtained under controlled conditions in an acoustic tank. The process of data gathering is described in detail and processing results are presented.

PART I

State of the art

PART I

CHAPTER 1

General Overview

The main problem that originated this thesis was how to enable a submarine to identify the ships that are near it by hearing the underwater sound they produce. This is a crucial problem for submarine operation, and as we progressed in our work we found many other areas of application where the same techniques could be used, both for military and for civilian purposes. In all those applications, underwater sound must be recorded, pre-processed, and classified into one of a series of possible classes.

Classification of underwater sound, also referred to as hydrophonic effects, can be seen as a very particular case of the more general classification problem, or pattern recognition problem. In this

part of the thesis we shall overview the general problem of classification, with particular emphasis on the approaches and techniques that will be improved as original contributions in part II, and those that will be used to process our data in part III.

Classification of data is a very well studied problem in statistics, computer science, and engineering in general, and many excellent textbooks have been written on the subject, e.g., (Fukunaga 1990; Bishop 1995; Duda, Hart *et al.* 2001). As a whole, the problem encompasses much more than the strict classifier design problems that shall be overviewed in Chapter 5, and includes problems such as gathering the data, choosing what aspects of that data are relevant, validating the results, etc.

Generally, the whole classification task can be divided into the following steps, shown graphically in Figure 1:

- a) Obtain the raw data.
- b) Extract features from that data.
- c) Perform some exploratory data analysis, and gain insight on the problem, if necessary.
- d) Select the features most relevant for classification.
- e) Design a classifier.
- f) Validate the classifier to obtain an estimate on its reliability, i.e., on how much confidence should be given to the classification it performs on new data.

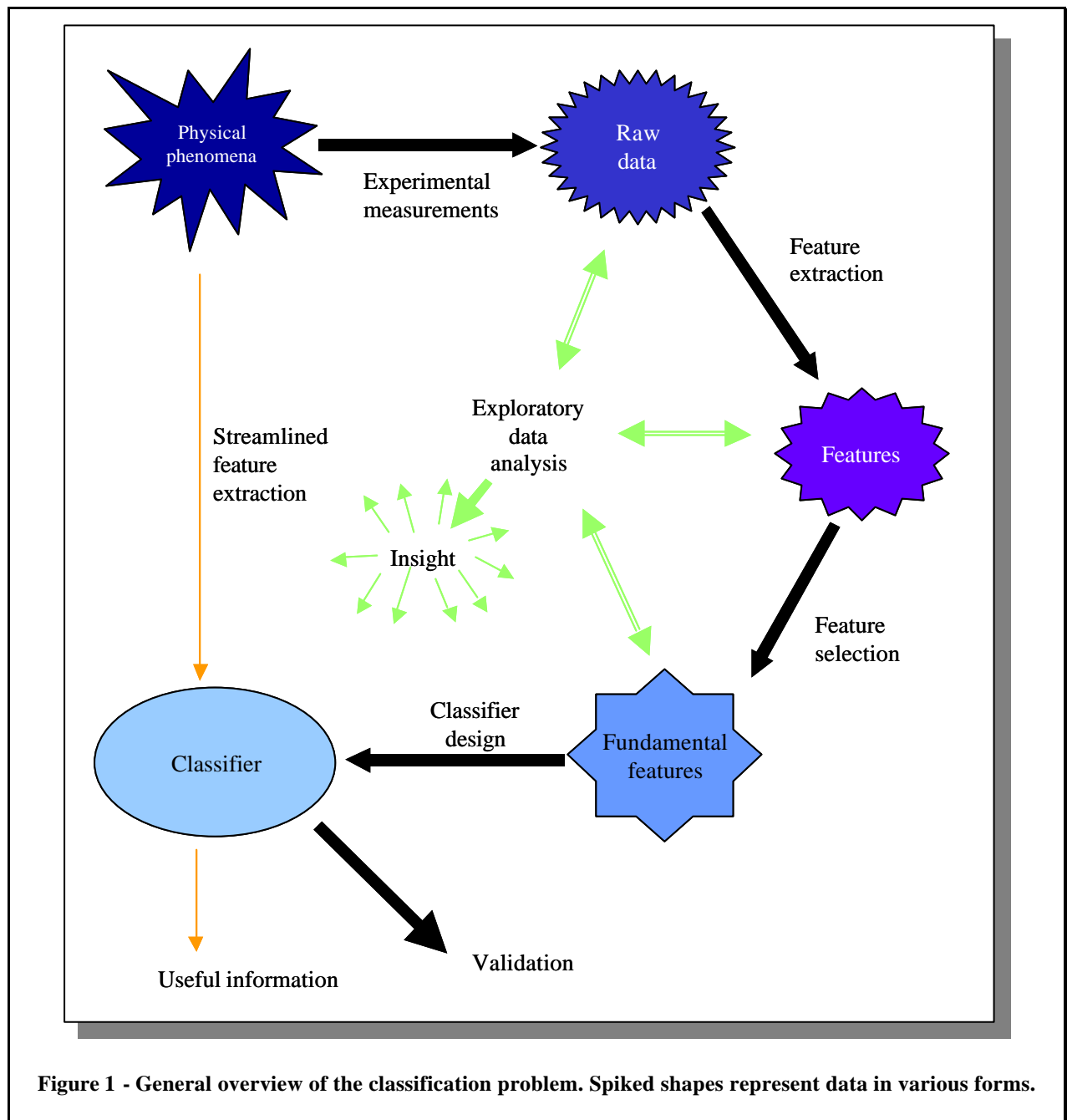
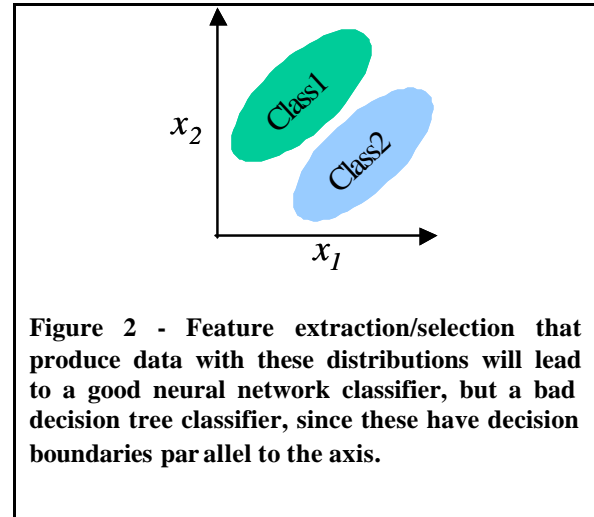


Figure 1 - General overview of the classification problem. Spiked shapes represent data in various forms.

This division into separate and more or less independent tasks is a simplification that is necessary to tackle the problem effectively. In reality all tasks are strongly interconnected. As an example, a feature extraction/selection technique that produces 2-dimensional features with a diagonal distribution as shown in Figure 2, may be optimal for a neural network based classifier (say a MLP (Rumelhart, Hinton *et al.* 1986)), but will certainly lead to a poor Decision Tree based classifier, such as those proposed by (Breiman, Friedman *et al.* 1984).

It can also be argued that if the right parameters are measured and the feature extraction technique is “optimal”, it will be possible to map the raw data directly into the “class space”, rendering all following steps unnecessary. The exact opposite may happen if the raw data available can be used directly to design a good classifier in an efficient manner. The notions of “pre-processing”, “feature extraction” and “classifier” become blurred and almost undistinguishable. Naturally, since these cases constitute trivial problems that are infrequent in practice, no more attention will be given to them. The only useful point to retain is that good feature extraction techniques can lead to simpler classifiers, and powerful classifiers can make up for poor feature extraction. The choice of where to invest time and effort is problem and user dependant.



Although there has been a lot of work choosing the best combinations of techniques for specific problems, a general unifying approach that can be applied to any generic problem is not foreseeable. Therefore, we will follow the traditional approach and look at each step in the classification problem separately.

Some of these steps are very problem dependant, such as the measurements to be made of the phenomena, while others are almost problem independent, such as the design of a classifier given the fundamental features.

The phenomenon to be classified is many times a physical one, such as the noise produced by a ship moving in the ocean studied in this thesis. However, it could be anything, such as a web page that we may want to categorize or information about bank transactions. When attempting to obtain a classifier, the choice of data to use and the process of gathering them is crucial to the success of the process. When the data can be obtained via controlled experiments, the design of those experiments is very important. It should guarantee that the data obtained are representative of the problem at hand and are not biased or contaminated in any way. The design of experiments is one of the most ancient arts of science, perfected by generations of chemists, physicists or biologists. It is extremely problem dependent, requiring careful planning, execution, and note

taking. When experimental data generation is not possible, such as when we want to classify stock exchange fluctuations, care must be taken in selecting representative data, and characterizing the conditions under which they were obtained. Once again, it is an extremely problem dependant issue, for which no general recipes can be given. Thus, we will not attempt to overview this topic, but will characterize our data gathering efforts in part III of this thesis.

Feature extraction is also highly problem dependant, so in chapter 2 we will only overview the techniques used in this thesis.

When overviewing feature selection techniques in chapter 3, we will give particular attention to Rough Sets (Pawlak 1988) since it is a relatively recent technique that we find particularly attractive.

The overview on exploratory data analysis given in chapter 4, although not strictly necessary for the pure classification problem, can be very useful. When dealing with a difficult and possibly ill-understood problem, exploratory data analysis can provide us with clues as to what is happening in the available data, how to improve the various steps of the process, and what to expect from the final classifier.

In chapter 5, we will overview the main research area of this thesis, namely nearest neighbor or prototype based classifier design. We will position this type of classifiers within the much broader scenario of classifier design techniques but will not discuss them individually.

Closing the classifier design cycle, we will overview validation techniques in chapter 6.

Finally, it must be pointed out that in a practical situation, all the steps discussed will probably be iterated in closed loop, until a satisfactory result is obtained.

PART I

CHAPTER 2

Feature Extraction

2.1 – Introduction

By feature extraction, we mean the process that transforms the raw data into data that can be used by a classifier. By feature, we mean a component of the multidimensional vector used to represent those data.

Feature extraction is a separate task from feature selection that will be overviewed in chapter 3, since while feature extraction *generates* new data from operations performed on the raw data,

feature selection will only *choose* some components of this data (the most relevant) to be used by the classifier.

Usually, one generates more features than the ones that will be used. The reason for this is that, when designing a classifier, we usually do not know a priori what features are best for classification. The best solution is then is to generate all features that, for some reason, are felt to be useful, and then choose the best.

The choice of feature extraction techniques is extremely problem dependent. All a priori knowledge about the problem should be used at this stage. For example, if we have the measurements of width and length of wooden boards, and know that their area may be important for the classification task (say their distribution amongst different warehouses), then an obvious feature extraction technique would be to simply multiply those two parameters and obtain the area of the boards.

Most classifier design techniques require the data to be presented in sets of small units called patterns, pattern vectors, samples, examples or simply data vectors. In this thesis we have adopted the term pattern, for it is both more general and less ambiguous than the others. When the raw data are obtained, many times they are already in the form of something to which we can call patterns, but other times they are not. In this latter case, and if required by the classifier technique we want to use, the feature extraction technique must divide the data into patterns. For example, when classifying sound pitch, we may have a continuous recording obtained by a microphone. This continuous sound signal must then be broken down into small fragments, where the pitch is assumed more or less constant, so that the classifier can determine that pitch.

A common concern of feature extraction techniques is to obtain features that are invariant to irrelevant aspects of the data, as far as the classification is concerned. When classifying letters, their orientation may be irrelevant, or when classifying weapons by their sound, the instant in time when the shot occurs may be irrelevant. Without trying to be all encompassing, we can say that it is common to wish for properties such as time-invariance, rotation-invariance, position-invariance, or scale-invariance.

Feature extraction can also be viewed as a data or knowledge representation problem, and is treated as such by some authors, such as (Anzai 1992).

Feature extraction will many times require the use of signal processing. We shall briefly review some of the most popular techniques.

2.2 - Fourier transforms

The original work that led to the Fourier transform is due to Jean Baptiste Joseph Fourier who, in the beginning of the 19th century, used sinusoidal decomposition techniques to solve heat transfer problems. It has been widely used in science and engineering ever since. It decomposes a complex valued signal (of which a real valued signal is just a particular case) into a sum of sine and cosine signals. This effectively maps the original signal into a different domain, called the frequency domain. The representation of the signal in the frequency domain is called the spectrum of the signal. If the phase of the sinusoidal functions is ignored and only their amplitude is used, this mapping is time-invariant, thus achieving one of the usual goals of feature extraction. The frequency domain has a very intuitive physical meaning of “cycles per second” (even if sometimes misleading (Oliveira and Barroso 1998)), that is a useful feature by itself in many problems. The square modulus of the spectrum is usually referred to as the “power spectral density” or “energy spectral density” for power or energy signals respectively, and also has a very useful meaning, since it allows us to determine original signal’s power (or energy) contained in any given frequency band.

By definition, the Fourier transform $X(\omega)$ of a signal $x(t)$ is given by

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t} dt. \quad (1)$$

For discrete signals, such as those available for processing by a digital computer, the Discrete-time Fourier transform is defined as (e.g. (Oppenheim and Shafer 1989))

$$X(\omega) = \sum_{n=-\infty}^{n=+\infty} x(n)e^{-j\omega n}, \quad (2)$$

where ω , known as normalized angular frequency, is given in radians per sample, and is related to the more traditional notion of frequency f in Hertz (cycles per second) by $f = (\omega \times f_s) / 2\pi$, where f_s is the sampling frequency (in samples per second). From the formula given above, it is obvious that the Discrete-time Fourier transform at frequencies that differ by 2π will be exactly the same.

Thus, we only need to compute the transform for a 2π interval. In what follows we will refer to the Discrete-time Fourier transform simply as Fourier transform.

When choosing the time interval between two consecutive values of x , known of sampling period (the inverse of the sampling frequency), care must be taken to guarantee that information is not lost. The minimum sampling frequency must be greater than twice the highest frequency component in the original signal. This is known as Nyquist's theorem (e.g. (Oppenheim and Shafer 1989)), and if care is not taken to respect it, aliasing will occur, i.e., the obtained spectra of the signal will be affected by “phantom components” resulting from high frequency components of the signal. A common way to avoid this is to filter the original (analog) signal before it is sampled. These filters are known as anti-aliasing filters.

The Fourier transform, as defined above, would require complete knowledge of the original signal from $n=-\infty$ to $+\infty$. This is obviously not possible for a finite digital system, so the Short Time Fourier Transform (e.g. (Bendat and Piersol 1993)) is used, defined as

$$X(\mathbf{w}) = \sum_{n=0}^{n=N} x(n)e^{-j\mathbf{w}n}, \quad (3)$$

where N is the number of samples (data points) considered.

Unfortunately, considering just a time-limited portion of the original signal is equivalent to using that original signal multiplied by a square pulse with width equal to the observation time considered. It is well known (Oppenheim and Shafer 1989) that multiplication in the time domain is equivalent to convolution in the frequency domain. The final effect is that the spectrum of the original signal is blurred by the convolution with the spectrum of the square pulse. To minimize this effect, the sampled signal is usually multiplied by a window function (Harris 1978), that has more desirable features than the square pulse function. Each different window function has its own specific advantages and disadvantages, balancing the width of the main lobe (which will reduce the actual frequency resolution), the amplitude of the side lobes, the power contained in those lobes (to minimize power leakage), etc. One of the first windows to be proposed, the Hamming window (Harris 1978) is probably the most used for its balance of characteristics since its highest side lobe is 43 dB lower than the main lobe and the main lobe has an equivalent noise bandwidth of only 1.36 bins (e.g. (Poularikas 1998)).

To be able to reconstruct the original discrete signal from its spectrum, this spectrum must be calculated in at least as many points as were present in the signal. Failure to do that will result in time aliasing (Oppenheim and Shafer 1989).

The computation of the Short Time Discrete Fourier Transform directly from its definition is very time consuming. A computationally very efficient technique was developed and named Fast Fourier Transform (FFT) (Cooley and Tukey 1965). It computes the Discrete Time Fourier Transform in N equally spaced points in the frequency domain, where N is the number of points in the time domain, and required to be a power of 2. Other efficient algorithms to compute the Fourier transform have since been developed (Poularikas 1998). In practice, almost all engineering applications of the Fourier Transform use a FFT procedure to calculate it.

For most practical problems, the original signal is real-valued and, in this case, its Fourier transform will possess Hermitian symmetry (i.e. complex conjugate symmetry) around zero frequency. Thus, for a discrete real signal, we need only keep the values of its Fourier transform from 0 to π .

The signals for which we want to compute the Fourier transform are many times contaminated with noise, which is frequently assumed white and Gaussian. One way of canceling out this noise, is to compute the Fourier transform of different portions of the signal and then compute their averages. If the noise contained in the different portions is not correlated, the averaged transform will be less affected by it. Even if there is some correlation between the noise in the two portions (for example, if we use overlapping portions), there will still be some gains. Averages of successive Fourier transforms of overlapping parts of a signal are known as Welch periodograms. It has been proved that, assuming white Gaussian noise, the unbiased estimator with minimum variance due to noise is obtained using a 50% overlap of the base portions of the signal (e.g. (Kay 1988)).

2.3 - Other frequency based techniques

Despite its wide application, the Fourier Transform has a few drawbacks. One is that when representing the signal in the frequency domain, all information about the location in time is lost. For stationary signals, this presents no problem, but it is not appropriate for analyzing signals that have some time-varying dynamics, such as in the analysis of transients. One common solution is

to use short time Fourier Transforms, assuming that the signal is quasi-stationary in that short time, and then use a sequence of these Fourier Transforms in what is known as a spectrogram. The quasi-stationary assumption is not always verified. Even then, the spectrogram may constitute a very useful and practical tool. For the non-stationary case other tools have been developed, such as bilinear time-frequency distributions and wavelet transforms.

One of the first and best known time-frequency distributions is the Wigner-Ville transform (Wigner 1932; Qian 2002), known as WVD. It is defined as

$$WVD_s(t, \omega) = \int_{-\infty}^{+\infty} s\left(t + \frac{\mathbf{t}}{2}\right) s^*\left(t - \frac{\mathbf{t}}{2}\right) e^{-j\omega\mathbf{t}} d\mathbf{t}, \quad (4)$$

where

- $s(\bullet)$ is the function in the time domain,
- $s^*(\bullet)$ is its conjugate,
- t is an instant in time,
- ω is a given frequency.

This allows us to estimate the spectrum of a non-stationary signal at any point in time. However, there are limits to the time-frequency resolution that is achievable, as described in (Oliveira and Barroso 1998). One of the main disadvantages of WVD is that it produces severe cross-term interferences, which tend to contribute to time-frequency descriptions that are many times difficult to interpret. This has spurred a vast array of alternative distributions (Qian 2002), most of them belonging to what is called Cohen's class. In some of the experiments performed by us, we used one such variant, developed in (Hippenstiel and Oliveira 1988; Hippenstiel and Oliveira 1990) known as IPS, and defined as:

$$IPS(t, f) = \frac{1}{2} \int_{-\infty}^{+\infty} [s(t)s^*(t - \mathbf{t}) + s^*(t)s(t + \mathbf{t})] e^{-j2p\mathbf{t}} d\mathbf{t}. \quad (5)$$

Another major alternative is the use of Wavelet transforms. These transforms have their roots in the work of Gabor (Gabor 1946), but only reached widespread use with the developments of Daubechies (Daubechies 1990). The main idea behind wavelet transforms is to decompose the original signal not into sinusoids, but into other base functions. These functions can be scaled into longer or shorter versions, in a manner that matches frequency variations in sinusoids. They may also be “positioned” anywhere in time, making it possible to localize short duration

transients in signals. Although almost any base function could in theory be used, it is important that the decomposition yield a transformed signal that accurately represents the original function, and that the process can be inverted to reconstruct that signal. This implies that the basis functions constitute what is known as a compact base. The most used base function is due to Daubechies, and efficient implementations of a wavelet transform based on it are widely available (e.g. (Mathworks 2001)). It has been found that wavelet transforms, not only allow good time-frequency localization of transient signals, but can also produce very compact representations of these signals. They are fast becoming as commonplace as Fourier transforms, and being used in everyday applications such as JPEG 2000 image compression (Skodras, Christopoulos *et al.* 2000).

Another family of frequency based techniques for signal analysis comes from using higher-order statistics of the signal. These are very well reviewed in (Nikias and Petropulu 1993). One of the most used is the cepstrum analysis, originally due to (Bogert, Healy *et al.* 1963), and studied in detail in e.g. (Oppenheim and Shafer 1989). The complex cepstrum of a discrete signal is defined as the inverse Z transform of the logarithm of the function's Z transform, but can be obtained using the Fourier transform by using

$$c_s(m) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} \log(S(\omega)) e^{jm\omega} d\omega, \quad (6)$$

where $S(\omega)$ is the Fourier transform of signal s .

The cepstrum has many interesting properties, and is frequently used to detect harmonically related components of a signal.

2.4 - Principal Component Analysis and related techniques

Principal Component Analysis (PCA) is an axis transformation technique that finds orthogonal axes where the covariance between features is zero, and ranks those axes according to their own variance. It was first proposed by Karl Pearson in 1901 (Jolliffe 1986; Flury 1988; Child), and has since been improved and widely used for statistical analysis and dimensionality reduction.

Although not the process followed by the more efficient algorithms, the basic idea is to find the direction where variance is maximum, take it as an axis, also known as principal component or

most relevant feature. This feature, which is a linear combination of the original ones, corresponds to the direction in space along which the data is most spread, and thus must contain more information about what distinguishes one pattern from another. We then repeat the process to find an axis perpendicular to this one that maximizes the remaining variance, and iterate until there is no more variance in the data. If the original data can be represented in a space with lower dimensionality than the original one, there will be fewer principal components than original features. In most applications, the importance of successive principal components, although rarely reaching zero, decreases to very small values that may be ignored without substantial information loss. The PCA can thus be used to reduce the dimensionality of the data.

After PCA has been performed on a given dataset, the covariance matrix obtained can be used as a linear transformation to map any new data into the principal component space, or any of its subspaces. This is known as the Karhunen-Loeve transform¹, and is frequently used in telecommunication problems.

It must be noted that PCA is not necessarily a good preprocessing step for classification, especially if we consider only the most relevant components. An elegant example is given in (Bishop 1995) showing a situation similar to that depicted in Part1 - Chapter 1 of this thesis. In that example, the first principal component, although explaining most of the variance in the data, is irrelevant for classification, while the second principal component is the ideal feature for classification. Therefore, PCA must be used with caution when attempting classification.

It must also be noted that Principal Component Analysis will perform a linear mapping onto straight axis. If the data are distributed along a spherical cap or any other curvilinear surface, it would be more convenient to use some sort of *curved axis*. To a certain extent, this can be achieved with principal curves (Hastie and Stuetzle 1989; Kégl 2000, Krzyzak *et al.*1996) (Chang and Ghosh 2001), namely with the incremental Hastie-Stuetzle Algorithm. The mapping performed by these principal curves resembles the one performed by the SOM discussed in chapter 4. Unfortunately, the methods available require a lot of prior knowledge about the data.

¹ Karhunen published his groundbreaking paper in German, with the name “Uber lineare methoden in der Wahrscheinlichkeitsrechnung”, in *Annales Academiae Scientiarum Fennicae*, Series A1: Mathematica-Physica, vol 37, pages 3-79.

As far as we know, a general purpose algorithm that performs efficiently and reliably for any given data has not yet been developed.

There has been a lot of interest recently in another pre-processing technique, called Independent Component Analysis (Hyvarinen and Oja 2000), but we shall not use it in this thesis.

PART I

CHAPTER 3

Feature Selection

3.1 – Introduction

After obtaining a number of features that form the patterns to classify, we should try to select only those that can indeed improve the performance of the classifier. This process is known as feature selection.

Feature selection will generally lead to loss of information, and is many times based on singular transformations. This would be undesirable if we were attempting to describe data, such as is the goal of principal component or factor analysis. However, when we want to perform supervised

classification, or when we want to focus on a particular aspect of those data, we do want to get rid of any information that would distract us from our goal.

There are a few different reasons why we consider that this step should be taken:

- a) Reduce noise generated by irrelevant features. Many classifiers are sensitive to irrelevant features, and will degrade their performance when these features are included. Distance based classifiers, such as the ones used in this thesis, are particularly sensitive to this. If a random feature is included, it will contribute to the distance measure just as much as any other feature. If the features were not scaled (or whitened), they may contribute even more than a relevant feature. Thus, due to this distortion, a pattern may end up being closer to patterns of a different class, even if originally the classes were well clustered by classes.
- b) Reduce the risk of overfitting the training data. The more features are used, the more detailed the classifier can be. As we shall see later, if a classifier has too many degrees of freedom it may adjust itself perfectly to the training data, but perform poorly when used with other data. Reducing the number of features, and thus the degrees of freedom of the classifier, will usually improve generalization.
- c) Make the classifier computationally feasible. Too many features will require not only a lot of computing power to obtain them, but even more computing power when training and using the classifier. Fewer features will lead to a faster, thus more useful classifier.

As pointed out earlier, the best features for one type of classifier are not necessarily the best features for another. Therefore, it is frequent to perform feature selection and classifier testing at the same time in what is called closed loop (Cios, Pedrycz *et al.* 1998). The basic idea is to choose a given set of features, train the classifier with them, and assess the performance. If the performance is not satisfactory, another set of features will be selected, and the process repeated. Since closed loop feature selection requires training many different classifiers, it can be a lengthy process. To abbreviate it, a simplified version of the classifier design process may be used. Since, in this step, we are mainly concerned with the relative merit of different sets of features, we can try them with a under-trained or over-simplified classifier that has the same basic properties of

the final classifier. When a final set of features is selected, the final classifier can then be fine-tuned. While this procedure cannot guarantee optimality, it generally produces good results.

If we do not want to iterate the classifier design phase in closed loop with feature selection, we may attempt to select features based on their capacity to separate the different classes regardless of the specific classifier used. This is called open loop selection. It can be argued that the criteria used to measure the separability capacity of the set of features is implicitly considering a certain type of classifier, but we shall not consider that effect. A number of different techniques have been proposed and used to perform open loop feature extraction. We shall now overview a few of them.

3.2 - Scatter Matrices

Intuitively, the best features for classification are those that have similar values within each class and different values between classes. This can be measured using scatter matrices (Fukunaga 1990).

For feature selection 3 scatter matrices are considered: the within-class scatter matrix S_w , the between-class scatter matrix S_b , and the mixture scatter matrix S_m .

The within-class scatter matrix S_w measures the dispersion of each class of pattern vectors around that class's expected value, and is defined as

$$S_w = \sum_{i=1}^C P_i E((\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T | \mathbf{x} \in c_i) = \sum_{i=1}^C P_i \Sigma_i, \quad (7)$$

where

- \mathbf{m}_i is the expected value of patterns of class i ,
- P_i is the prior probability of class i ,
- c_i is the class i ,
- Σ_i is covariance matrix for class i .

As shown, S_w is simply a weighted average of the covariance matrices of each class.

The between-class scatter matrix S_b measures the dispersion of the class's expected values around the global expected value, and is defined as

$$S_b = \sum_{i=1}^C P_i (\mathbf{m}_i - \mathbf{m}_0)(\mathbf{m}_i - \mathbf{m}_0)^T, \quad (8)$$

where \mathbf{m}_0 is the expected value of patterns of all classes.

The mixture scatter matrix measures the dispersion of all patterns around the global expected value, and is simply the sum of S_w and S_b :

$$S_m = E(\mathbf{x} - \mathbf{m}_0)(\mathbf{x} - \mathbf{m}_0)^T = S_w + S_b. \quad (9)$$

These matrices contain a lot of information about the discriminatory power of each feature, of relations between those features, and discriminatory power of groups of features. Unfortunately, it is not easy to make use of that information. The most common technique relies on considering each feature independently, and selecting those that have greatest ratio of within-class variance to between-class variance. This can be done by calculating a diagonal matrix J that is the quotient of S_b and S_w , and choosing the features that have greatest value in that matrix:

$$J = \text{tr}(S_w^{-1} S_b). \quad (10)$$

As discussed in detail in (Fukunaga 1990), many other choices for J are possible, and can be summarized as follows.

$$J_1 = \text{tr}(S_2^{-1} S_1), \quad (11)$$

$$J_2 = \log(S_2^{-1} S_1) = \log |S_1| - \log |S_2|, \quad (12)$$

$$J_3 = \text{tr}(S_1) - \mathbf{m}(\text{tr}(S_2) - c), \quad (13)$$

$$J_4 = \text{tr}(S_1) / \text{tr}(S_2), \quad (14)$$

where S_1 and S_2 can be $\{S_b, S_w\}$, $\{S_b, S_m\}$, or $\{S_w, S_m\}$.

These scatter matrices can also be used to help design feature extraction techniques that maximize their values.

3.3 - Rough Sets

Rough set theory was originally developed by Zdzisław Pawlak, in articles published within the Institute of Computer Science of the Polish Academy of Science, and was presented in English in (Pawlak 1982), as an alternative to Fuzzy Set theory and tolerance theory.

In a broad overview, it describes sets, which correspond to classes or concepts, based on their *upper* and *lower approximations*. These upper and lower approximations are obtained using the available features, which are called attributes in Rough Set literature, and available patterns, here called objects. Contrary to fuzzy sets, nothing is said about the membership of patterns that lie between the lower approximation (under which we are sure the object belongs to the given set), and the upper approximation (above which we are sure the object does not belong to the given class).

Rough set theory has proved to be particularly useful when dealing with imprecise data. It can be used to find relationships in those data, remove redundancies, generate decision rules, reduce databases, and select features for classification, which is our purpose.

Extensive work has been done in this area. For the basic foundations of Rough Set theory we would recommend (Pawlak and Slowinski 1994). A good collection of papers and other resourced related to Rough Set theory can found at “<http://www.roughsets.org>”. It must however be noted that Rough Set theory is only a framework for the description and resolution of problems. In that framework, goals and cost functions are defined, but Rough Set theory relies on traditional optimization techniques to achieve many of its goals, namely finding the best features for classification (known as finding the *relative reducts*).

Since Rough set theory is not yet widely known, and the concepts used are important to understand some of the software used in this thesis, we shall provide a short introduction to the main concepts of roughest theory. It must be pointed out that this introduction, while enabling the reader to understand the language used by the Rough set community, does not cover many of the aspects of Rough set theory, namely the actual techniques used to solve the problems stated.

3.3.1. Basic concepts

The framework of Rough Set theory assumes an Information System, composed of a 4-tuple as follows

$$S = \langle U, Q, V, f \rangle, \quad (15)$$

where

- S is the information system,
- U the universe, defined as a nonempty finite set of objects $\{x_1, x_2, x_3, \dots, x_n\}$,
- Q a nonempty finite set of attributes,
- V the domain of values $V_q \hat{=} V_q$ for each attribute,
- F the decision function, also called information function, defined as

$$f: U \times Q \rightarrow V: f(x, q) \in V_q, \quad "q \in Q, \quad "x \in U. \quad (16)$$

The information system may be represented by a finite data table, in which the columns are labeled by attributes q (or features), the rows by objects x (or patterns), and each entry in the table has the value of $f(x, q)$.

3.3.1.1 – The indiscernability relation

Two objects are said to be *indiscernible* by a set of attributes A if and only if the values of those attributes are the same:

$$x \tilde{A} y \text{ (read "x is indiscernible to y by A")} \iff f(x, a) = f(y, a) \quad "a \in A. \quad (17)$$

Any subset $A \subset Q$ will lead to an equivalence relation on the universe U , called the *indiscernability relation*, denoted $IND(A)$, that can be defined as follows:

$$IND(A) = \{ (x, y) \in U \times U : f(x, a) = f(y, a) \quad "a \in A \}. \quad (18)$$

The indiscernability relation $IND(A)$, as an equivalence relation, splits the universe into a family of equivalence classes $\{X_1, X_2, \dots, X_r\}$. The family of all equivalence classes defined generates a partition of U , and is denoted by A^* . Alternatively, this partition is also referred to as *classification*, and denoted by $U/IND(A)$. Each of the equivalence classes X_i is thus seen as a certain type of *class*, called an *A-elementary set*. Each of these sets can be defined from an object x as

$$[x]_A = \{ y \in U : f(x, a) = f(y, a) \quad "a \in A \}. \quad (19)$$

These *A-elementary sets* form the smallest discernible groups of objects, and thus the maximum granularity achievable. These *A-elementary sets* constitute the *A-basic knowledge*, that is the maximum amount of knowledge we may get using the set of attributes *A*. If we consider all the attributes of the information system *S*, we will obtain the *Q-elementary sets*, which are called *atoms*, since there is no way of distinguishing objects within them, whatever attributes are considered. A union of one or more *Q-elementary sets* constitute a *concept*, *X*, definable in the information system, which corresponds to the usual notion of class in classification problems.

3.3.1.2 – Decision tables

For classification problems, each object will have an assigned label, or class. Within Rough set theory this is done by dividing the attribute set *Q* into two disjoint sets, called the *condition attribute set C*, and the *decision attribute set D*, so that $C \cup D = Q \wedge C \cap D = \emptyset$.

In this case, instead of an information system *S*, we consider a decision table defined as

$$DT = \langle U, C \dot{\cup} D, V, f \rangle. \quad (20)$$

Like the information system, the decision table can be represented by a table, but now the columns are separated into condition attributes (corresponding to features), and decision attributes (corresponding to class labels).

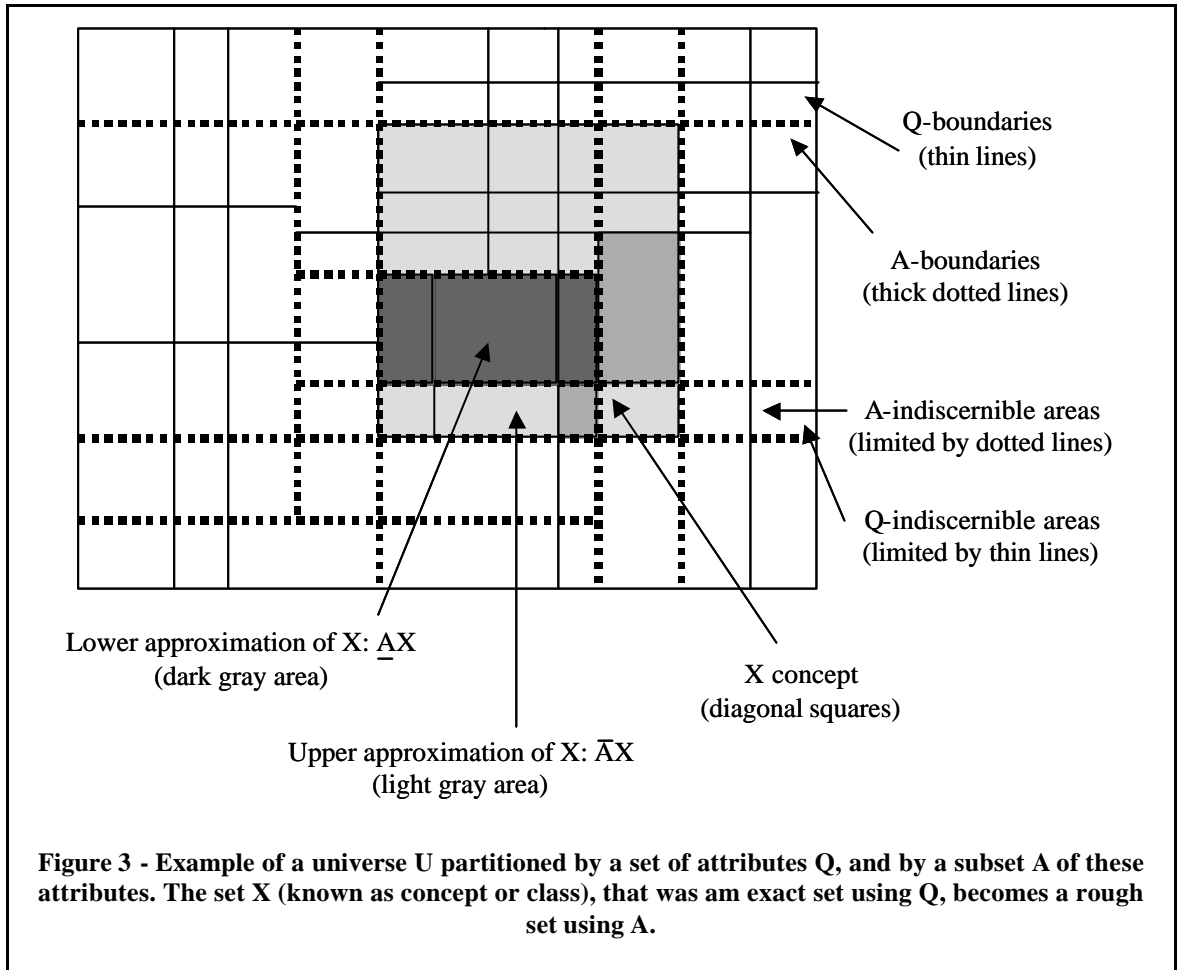
3.3.1.3 – Upper and Lower Approximation of sets

Once we select a given set of attributes, we no longer have the original space with its fine granularity, but an approximation space, denoted $AS = (U, IND(A))$, which will have a coarser granularity. Concepts, unions of *Q-elementary sets* defined by the original information system, may no longer correspond to unions of the *A-elementary sets* of the approximation space. Instead, in the new approximation space we may define upper (\overline{AX}) and lower (\underline{AX}) approximations to the concept *X* as

$$\underline{AX} = \{x \in U : [x]_A \subseteq X\} = \bigcup \{Y \in A^* : Y \subseteq X\}, \quad (21)$$

$$\overline{AX} = \{x \in U : [x]_A \cap X \neq \emptyset\} = \bigcup \{Y \in A^* : Y \cap X \neq \emptyset\}. \quad (22)$$

In plain English, the lower approximation contains only the objects that certainly belong to the concept, but not necessarily all, while the upper approximation contains all the objects that belong to the concept, and possibly some more that do not. A graphical example of upper and lower approximations is given in Figure 3.



The lower approximation of a set, $\underline{A}X$, is also known as the *A-positive region of X in S*, denoted $POS_A(X)$. The area outside the upper approximation, $U - \bar{A}X$, known as the *A-negative region of X in S*, is denoted $NEG_A(X)$. The region between the two approximations, $\bar{A}X - \underline{A}X$, known as the *A-boundary region*, is denoted $BN_A(X)$. Contrary to fuzzy set theory, that assigns a degree of membership to the objects in this area, rough set theory will just consider that it is undecidable whether these objects belong to X or not, given only the attributes A . This stems from a major philosophical difference between rough set theory and fuzzy set theory. While the latter assumes that the uncertainty about the class is an inherent characteristic of the object, the former assumes that that uncertainty is only due to our incomplete knowledge about its attributes.

The relation between the upper and lower approximations of a set will determine its roughness. If both sets are equal, $\overline{AX} - \underline{AX} = \emptyset$, the set X is said to be **A-definable**, and no uncertainty exists. Otherwise it is set do be **A-non-definable**, and may fall into one of 4 categories:

- a) A set is **roughly A-definable** if and only if $\overline{AX} \neq U \wedge \underline{AX} \neq \emptyset$. This will be the most common case, where given a set of attributes, we know for certain that some objects do belong to X and some do not.
- b) A set is **externally A-non-definable** if and only if $\overline{AX} = U \wedge \underline{AX} \neq \emptyset$. This is the case when we cannot be sure that a given object does not belong to X.
- c) A set is **internally A-non-definable** if and only if $\overline{AX} \neq U \wedge \underline{AX} = \emptyset$. This is the case when we cannot be sure that a given object does belong to X.
- d) A set is **totally A-non-definable** if and only if $\overline{AX} = U \wedge \underline{AX} = \emptyset$. In this case, the attributes A are completely useless in defining X, for we cannot be sure of anything.

When a set is roughly A-definable, it is important to have an idea “how rough” it is. To that end, the notion of accuracy of an approximation and quality of an approximation are used.

The accuracy of an approximation is defined as:

$$\mathbf{a}_A(X) = \frac{\text{card}(\underline{AX})}{\text{card}(\overline{AX})}. \quad (23)$$

This is a value between 0 and 1, and gives a good idea about how well we can define a concept with a given set of attributes. A value of 1 would mean that the selected attributes could define the concept perfectly, while a value of 0 would mean that those attributes were a poor choice.

The notion of accuracy can easily be extended to a group of concepts, forming what is known as the **accuracy of the approximate classification g**, defined as:

$$\mathbf{a}_A(\mathbf{g}) = \frac{\sum_{i=1}^n \text{card}(\underline{AX}_i)}{\sum_{i=1}^n \text{card}(\overline{AX}_i)}. \quad (24)$$

A similar notion, called **quality of the approximation classification g** is defined as:

$$\mathbf{r}(\mathbf{g}) = \frac{\sum_{i=1}^n \text{card}(AX_i)}{\text{card}(U)}. \quad (25)$$

If the concepts are disjoint, the quality will have a value between 0 and 1, but otherwise it may have a higher value.

3.3.1.4 - Classification and reduction of an information system

Some of the attributes of an information system may be redundant, i.e., the information they contain is also contained by other attributes. These attributes are said to be *dispensable*. In Rough set theory, the process of finding and eliminating these attributes is called attribute reduction, and it is tightly correlated with our notion of feature selection.

Formally, an attribute a is dispensable from a set of attributes A if and only if $IND(A) = IND(A - \{a\})$, i.e., if the original indiscernability relations generated by the set of all available attributes are the same as the indiscernability relations generated without it.

When deciding if a given attribute is redundant or not, one must have in mind what goal is sought from the information system. Absolute redundancy of an attribute is a rare occurrence, but if want to define a concept (perform a classification), then some attributes may be redundant relatively to that objective. If we remove all redundant attributes, we will have a set of attributes called a *reduct*. If we remove all attributes that are redundant relative to a given classification, we will have a set of attributes called a *relative reduct*.

There may be (and usually are) many different relative reducts for any given problem. If there are any attributes that are part of all the relative reducts, they form what is called the *relative core*. Attributes that belong to the core cannot be discarded without losing discernability. Attributes which are part of a reduct but not of its core can be “traded” by other attributes.

3.3.1.5 – Using Rough sets for practical classification problems

Rough set theory does not require that the domain for each attribute be finite. However, if it is not, then the granularity of the partitions defines will be infinitesimal, upper and lower

approximations will tend to converge, it will very hard to find reducts, and when used for classification, the results will probably overfit the training data and generalize poorly.

Thus, in practical applications, it is necessary to discretize the data, obtaining finite domains for each attribute. This discretization process can be critical, and many methods for doing it are possible (Stockdale 1998). While not discussing here the details on how to perform this step, we just want to mention that it is a necessary step, and most rough set programs provide means for doing so.

After obtaining a discretized representation of our problem, we must define which features are available, and designate them as *condition attributes*, and which are our classes or labels, and designate them *decision attributes*. We may then proceed to compute the *relative reducts*, which will be sets of indispensable features. We will usually choose the relative reduct with smallest cardinality as the features to use in our classifier. However, we may be interested in finding the core, i.e., the most important features, and then go back to the feature extraction process and obtain better features. Rough sets can thus be an important part of the interactive feature extraction/selection/exploratory data analysis process.

PART I

CHAPTER 4

Exploratory data analysis

4.1 – Introduction

Before choosing and designing a classifier, it is useful to have some insight on the data available. This insight is important both to validate the data gathering/feature extraction/feature selection process, and to decide which classifier is more appropriate. As has been mentioned before for other steps in the classification process, isolating this step is an artificial contraption, since it can be omitted, merged with the feature extraction/selection, or merged with the classifier itself. In any case, it will be more efficient if it is iterated in close loop with the other steps. This insight can be given by what are generally known as exploratory data analysis techniques. In recent

years, exploratory data analysis techniques have been the subject of intense research for data mining and knowledge discovery, and a lot of bibliography is available on that subject, e.g. (Sarker, Abbass *et al.* 2002).

The purpose of exploratory data analysis, as the name suggests, is to find relationships within the data, estimate its probability density distribution, and gain insight into the classification problem. Many different techniques may be used to this end, including:

- a) descriptive statistics, such as means, variances, measures of inter-distribution distances;
- b) statistical clustering techniques, such as k-means or Gaussian Mixture Models (Bishop 1995);
- c) factor analysis, such as Principal Component Analysis;
- d) projection pursuit techniques such as Sammon mapping (Sammon 1969) or Generative Topographic Mapping (GTM) (Bishop, Svensén *et al.* 1996);
- e) artificial intelligence clustering techniques, such as Self-Organizing Maps (Kohonen 2001), fuzzy C-means (Bezdek, Keller *et al.* 1999), Hierarchical clustering (Everitt, Landau *et al.* 2001), or dendograms (Sokal and Sneath 1963; Vesanto and Alhoniemi 2000).

A basic statistic description of data is taught in any introductory course in statistics, and allows us to estimate if the data follow a well known distribution (such as Gaussian or Poisson), or if the classes are well separated (by comparing the class means and higher moments). A lot of work has been developed in variance analysis but, since it is not crucial to the development of this thesis, we will review it no further, and only mention a few good references, such as (Damon 1987).

Principal Component Analysis has already been mentioned in chapter 3, and so will not discuss it here.

Projection pursuit techniques, such as Sammon mapping (Sammon 1969), try to map high dimensional data onto low dimensional spaces where they can be visualized. This visualization will allow human inspection of the data, and consequently a direct perception of the separation/distribution of the data. Some data clustering techniques (such as SOM (Kohonen 2001)), will also perform a low dimensional mapping of the data, and as we shall see later, the

visualization process is also important for these techniques. Unfortunately, data that is intrinsically high-dimensional cannot be projected into a low dimensional space without heavy distortion, that may lead to unreliable results. Estimating the true dimension of a dataset has been the object of intense research for a long time (Trunk 1968; Fukunaga and Olsen 1971; Schwartzmann and Vidal 1975; Urquhart 1983).

An important part of exploratory data analysis is clustering, also known as unsupervised classification, unsupervised learning, or data-driven learning, and excellently reviewed in (Jain and Dubes 1988; Fasulo; Everitt, Landau *et al.* 2001). Contrary to supervised learning (overviewed in the next chapter), where we want to obtain a pre-defined partition of the data, in unsupervised learning we want the data to be partitioned according to their “natural” structure. In supervised learning, a label is “pre-assigned” to each pattern of a known dataset. This assignment may be due to a human classification of the pattern (such when a human operation identifies the vehicle present in a series of photographs), or may be due to the data gathering process (such as when we take photographs of a known vehicle). In unsupervised learning, no such labels are necessary, and the data will be clustered together according to its own characteristics (for example, photographs of vehicle with a common characteristic may be clustered together).

These techniques will group the data patterns in clusters that can then be analyzed by the designer. If these clusters contain a strong mixture of the desired classes, that will probably mean that the previous steps were not appropriate for the task at hand. A bad clustering will probably mean that the classifier will have a hard time performing the desired separation of classes, resulting in a complex classifier, and one that will probably overfit the training data. In this case, it is probably better to try different feature extraction techniques, so as to find truly significant features.

After a reasonable clustering is achieved, we may sometimes use clustering technique as a classifier by itself. This will require assigning a label to each of the clusters obtained, and then finding a way of assigning each new data pattern to one of those clusters. The labeling is usually done by assigning to each cluster the label that occurs most in the data patterns that belong to it. When a new pattern is presented it is assigned to one of the existing clusters (for example using a distance measure), and given the same label as that cluster.

Clustering techniques are sometimes divided into two broad categories: partitioning techniques, and hierarchical clustering. Partitioning techniques, also known as k -clustering techniques, will try to partition the data into a predefined number k of clusters. Hierarchical clustering techniques, on the other hand, assume no pre-defined number of clusters, and present ever more detailed sub-clusters of data, so that the user may select the level of granularity desired. It would be out of the scope of this thesis to review all the most relevant clustering technique, so we shall now review only two of the most common ones, namely k -means clustering, and Kohonens Self Organizing Maps.

4.2 – K-means clustering

The k -means clustering technique consists in pre-selecting a certain number k of centroids, or means, and then finding the positions in the input space of these centroids, so that some measure of dispersion is minimized. In the original and most widely used version, the measure to be minimized is the sum of square distances between the data patterns and the centroids they are assigned to.

The k -means algorithm was originally proposed by (MacQueen 1967) as a stochastic on-line process, and reformulated as a batch process by (Lloyd 1982). Computationally efficient implementations of this algorithm are available, such as (Kanungo, Mount *et al.* 2002). There has been a shift in name from the original term “ k -means”, to the term “ c -means”(Bezdek, Reichherzer *et al.* 1998; Duda, Hart *et al.* 2001). While the original name focused on the fact that one must choose “ k ” points thus forcing k clusters, some authors feel that the letter c is more appropriate since it is the first letter for centroid, cluster, and class. While acknowledging the new trend, being traditionalist we will use the old term.

The original k -means algorithm (MacQueen 1967) can be described as follows.

Algorithm 1 - Original k-means clustering

```

Let
    k be the predefined number of centroids
    n be the number of training patterns
    X be the set of training patterns  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ 
    P be the set of k initial centroids  $m_1, m_2, \dots, m_k$  taken from X
     $\eta$  be the learning rate, initialized to a value in ]0,1[

1 Repeat
2   For i=1 to n
3     Find centroid  $m_j \in P$  that is closer to  $\mathbf{x}_i$ 
4     Update  $m_j$  by adding to it  $\Delta m_j = \eta(\mathbf{x}_i - m_j)$ 
5   Decrease  $\eta$ 
6   Until  $\eta$  reaches 0

```

This original version requires that a learning parameter η be set to a certain initial value, and that a certain decreasing function be used to make it converge to 0. The number of steps used to make it converge to 0 can be critical for the convergence of the centroids to locations where they do minimize the sum of square distances. With the advent of computers with more memory, another algorithm was devised: the batch k-means clustering (Lloyd, 1982). It is now the most commonly used algorithm for k-means clustering, since it is faster and it will converge more reliably to the global minimum.

The batch k-means algorithm is a form a stochastic hill climbing and can be described as follows.

Algorithm 2 - Batch k-means clustering

```

Let
    k be the predefined number of centroids
    P be the set of k initial centroids  $m_1, m_2, \dots, m_k$  that are
      randomly generated (or may be taken from X)
    X be the set of training patterns  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ 

1 Repeat
2   For i=1 to k
3     Find the set of patterns  $\mathbf{x}_j \in X$  that have each  $m_i$  as their
      nearest neighbor in P
4     Let  $m_i$  be the average of those  $\mathbf{x}_j$  points
5   Until there are no more changes in the values of  $m_i$  .

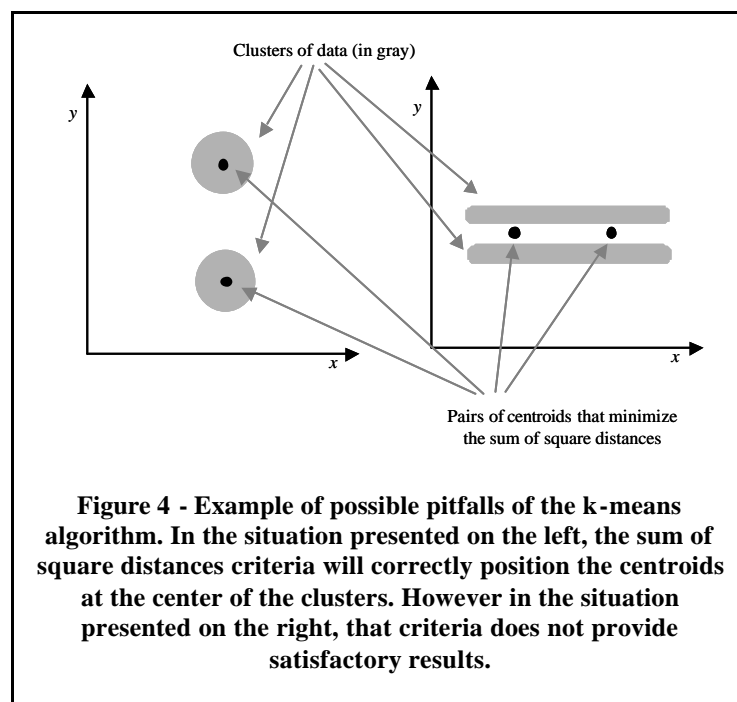
```

As noted in (Bishop 1995), the k -means algorithm can be seen as a special case of a Expectation-Maximization (EM) (Dempster, Laird *et al.* 1977) technique for a Gaussian mixture model. In a

Gaussian mixture model, the data distribution is modeled by a sum (or mixture) of Gaussian distributions, centered at different points. By using expectation-maximization (EM) learning, the optima locations for the centers of those Gaussian distributions can be determined. If we assume that the covariance within each of those distributions is 0 (i.e., they are localized “spikes”) the EM technique will lead to the well known k -means algorithm.

The k -means clustering technique has a few major drawbacks. The first is that it requires the user to pre-select the desired number of clusters. In many applications it is not obvious at the start how many clusters do exist, and so the user is forced to select an artificially big value of k to guarantee that no clusters are missed. It will thus be more appropriate to use this algorithm when we know with certainty how many clusters exist in the data. When this is not the case, we may use the final value of the sum of square distances as a measure of how well the data may be represented by a certain number of centroids. By repeating the k -means algorithm with increasing values of k we may search for a value of k that produces a sharp decrease in the sum of square distances, and use it as the best number of clusters.

The second drawback, is that since k -means minimizes square distances to the centroids, it will now cluster correctly data that have certain “long shaped” distributions, such as that presented in Figure 4. This effect can be minimized by selecting a larger k , so that each real cluster is represented by many smaller clusters, or to a certain extent by whitening the data.



Several changes and improvements have been proposed to the basic k -means clustering algorithm, but the most important is probably fuzzy c -means (Bezdek, Keller *et al.* 1999), around which many papers have been written, with many variations and improvements, e.g. (Kong, Wang *et al.* 2002). In that approach, instead of assigning each data pattern to its nearest centroid, a fuzzy membership to that centroid is assigned each pattern. That fuzzy membership will depend

on the distances between that pattern and the various centroids. A brief yet detailed explanation of fuzzy c-means can be found in (Duda, Hart *et al.* 2001).

4.3 – Self Organizing Maps (SOM)

Although the term “Self-Organizing Map” could be applied to a number of different approaches, we shall always use it as a synonym of Kohonen’s Self Organizing Map, or SOM for short. These maps are also referred to as “Kohonen Neural Networks”(Fu 1994), “Self Organizing Feature Maps-SOFM”, “Topology preserving feature maps” (Kohonen 1995), or some variant of these names.

Self Organizing Maps (SOM) were first proposed by Tuevo Kohonen in the beginning of the 1980s (Kohonen 1982), and stemmed from his work on associative memory and vector quantization. However, it was not until the publication of the second edition of his book “Self-Organization and Associative Memory” in 1988, and his paper named “The Neural Phonetic Typewriter” on IEEE Computer (Kohonen 1988) that his work became widely known. Since then there have been many excellent papers and books on SOM, but his book Self Organizing Maps (edited originally as (Kohonen 1995), and later revised in 1997 and 2001 (Kohonen 2001)) is generally regarded as the main reference on the subject. This book has had very flattering reviews, presenting a thorough covering of the mathematical background for SOM; its physiological interpretation; the basic SOM; and recent developments and applications. A thorough bibliography of SOM related issues (at <http://www.cis.hut.fi/research/som-bibl>) is maintained by the Neural Network Research Group (<http://www.cis.hut.fi/research>) that Kohonen created at Helsinki’s Technical University, and of which he, as professor emeritus, is still an active member. By July 2002, 4310 papers and books were referenced. Of these, for a comprehensive overview of SOM for clustering and visualization of data, we would recommend (Vesanto 1999) and (Vesanto and Alhoniemi 2000). A simple to follow tutorial, with illustrative examples, is available in (Lobo, Swiniarski *et al.* 1998).

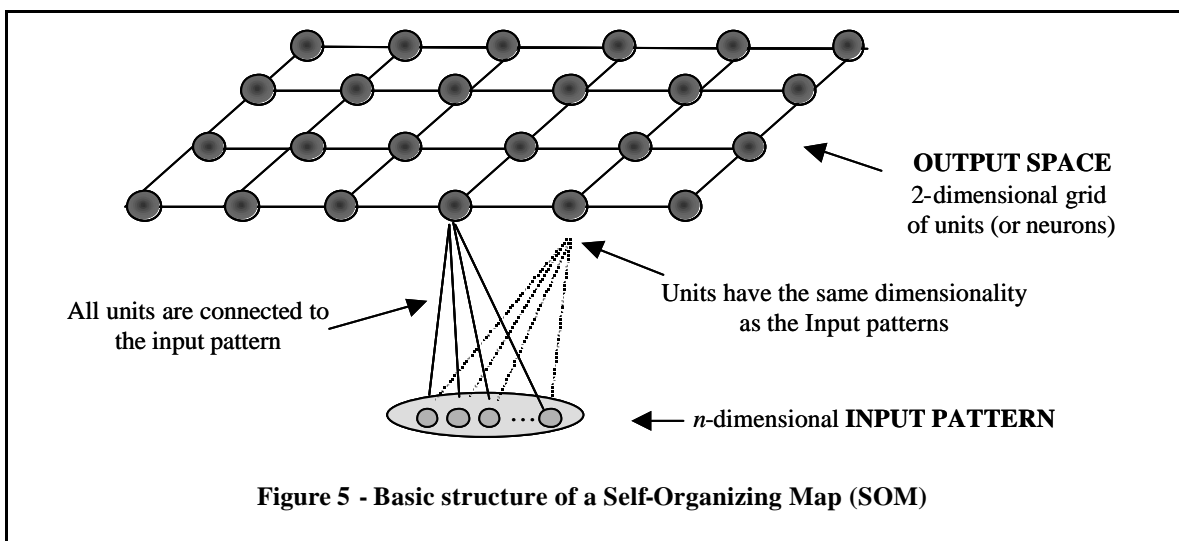
There are several public-domain implementations of SOM, of which we must mention the SOM-PAK developed by Kohonen’s group and discussed in chapter 2 of part III of this thesis, and the excellent Matlab SOM Toolbox, also developed by that group. It is currently in version 2.0 beta, and publicly available at <http://www.cis.hut.fi/projects/somtoolbox> . The SOM toolbox has, besides the Matlab routines, an excellent graphic-based user interface, that makes it very simple

to experiment with SOMs. Unfortunately, it was not available in time to be used extensively during this thesis.

Kohonen himself describes SOM as a “visualization and analysis tool for high dimensional data”, but they have used for clustering (Vesanto and Alhoniemi 2000), dimensionality reduction, classification, sampling, vector quantization, and data-mining (Kohonen 2001).

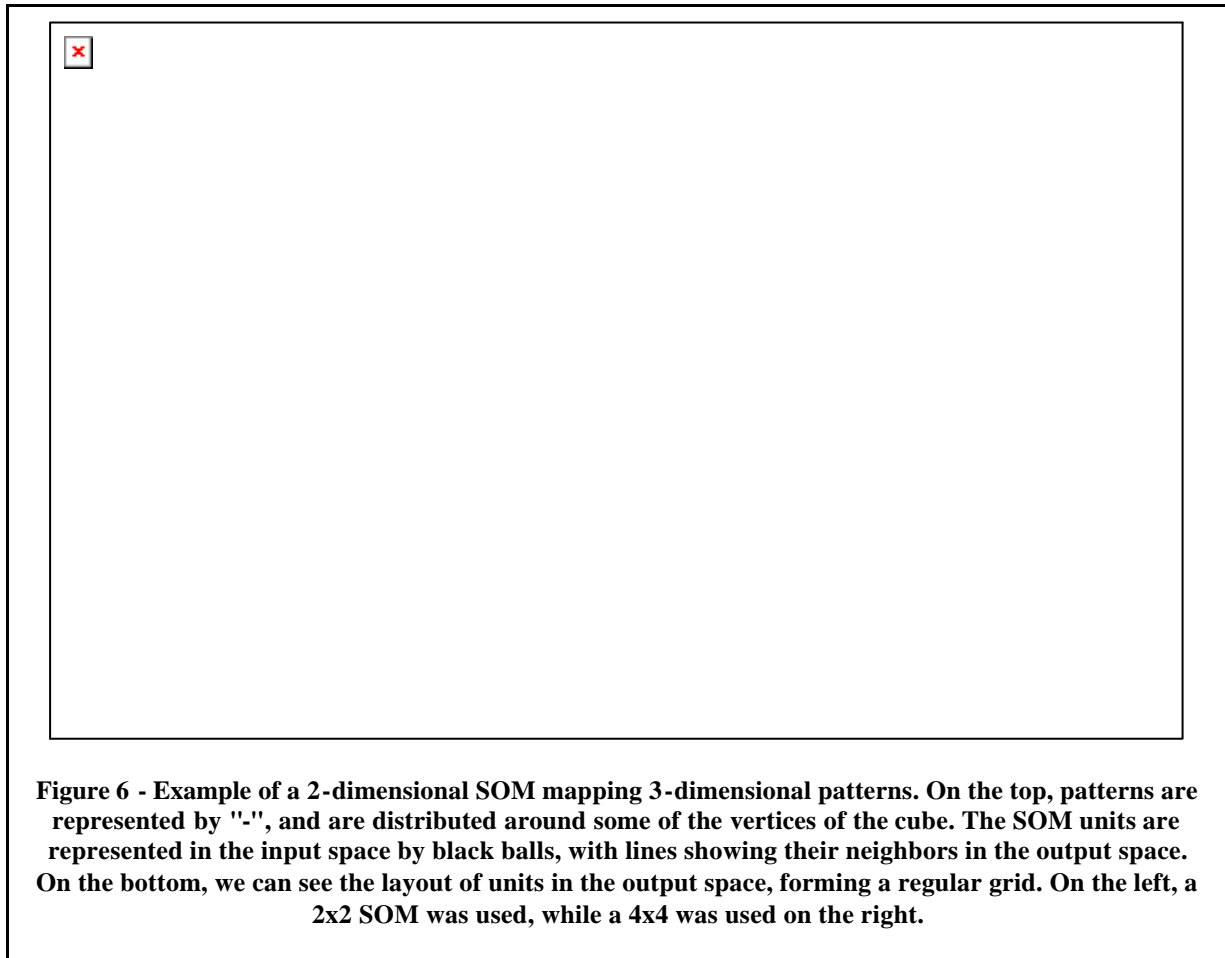
4.3.1 General and simplified overview

The basic idea of a SOM is to map the data patterns onto a n -dimensional grid of neurons or units. That grid forms what is known as the *output space*, as opposed to the *input space* that is the original space where the data patterns are, as seen in Figure 5. This mapping tries to preserve topological relations, i.e., patterns that are close in the input space will be mapped to units that are close in the output space, and vice-versa. The output space will usually be 2-dimensional, and most of the implementations of SOM use a rectangular grid of units. So as to provide even distances between the units in the output space, hexagonal grids are sometimes used (Kohonen, Hynninen *et al.* 1995). Single-dimensional SOMs are common (e.g. for solving the traveling salesman problem), and some authors have used 3-dimensional SOMs. Using higher dimensional SOMs, although posing no theoretical obstacle, is rare, since it is not possible to easily visualize the output space.



Each unit, being an input layer unit, has as many weights or coefficients as the input patterns, and can thus be regarded as **a vector in the same space as the patterns**. When we train or use a

SOM with a given input pattern, we calculate the distance between that pattern and every unit in the network. We then select the unit that is closest as the **winning unit**, and say that the pattern is **mapped onto that unit**. If the SOM has been trained successfully, then patterns that are **close in the input space** will be mapped to neurons that are **close (or the same) in the output space**, and vice-versa. Thus, SOM is “**topology preserving**” in the sense that (as far as possible) neighborhoods are preserved through the mapping process.



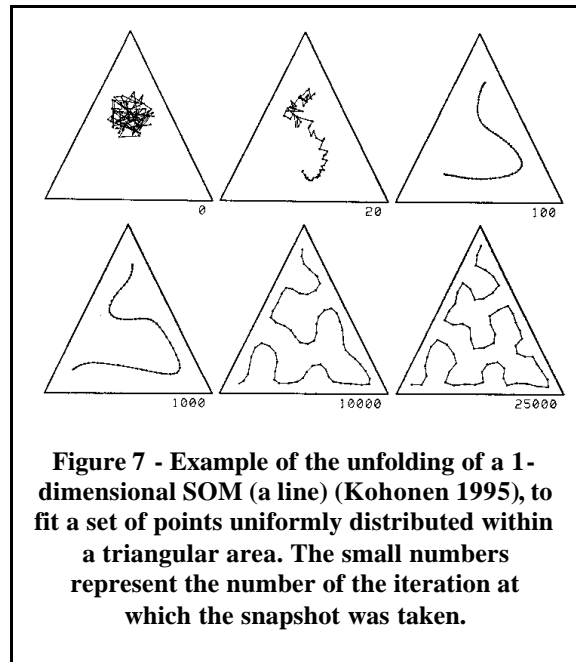
Generally, no matter how much we train the network, there will always be some difference between any given input pattern and the unit it is mapped to. This is a situation identical to vector quantization, where there is some difference between a pattern and its code-book vector representation. Thus, we refer to this difference as the **quantization error**, and use it as a measure of how well our units represent the input patterns.

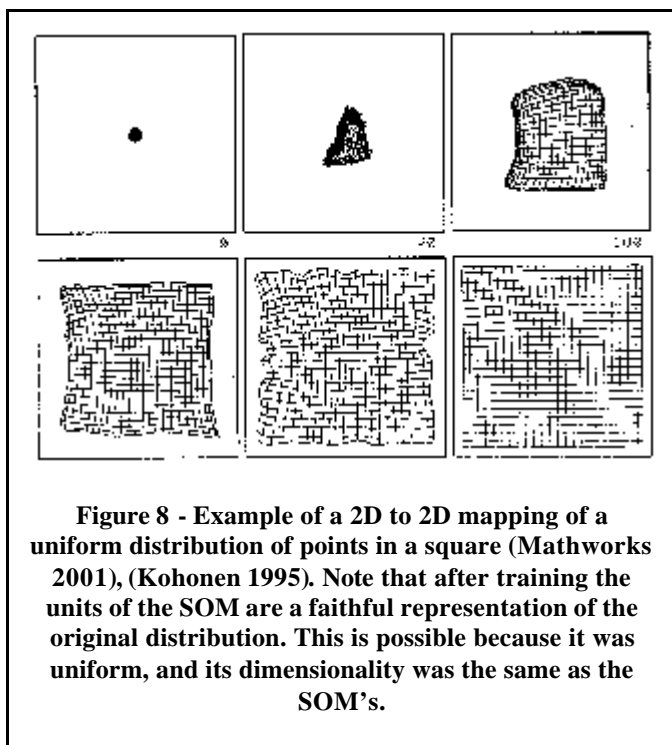
We can look at a SOM as a “rubber surface” that is stretched and bent all over the input space, so as to be close to all the training points in that space. In this sense, a SOM is similar to the input

layer of a Radial Basis Function (RBF) neural net, a neural gas model, or a K-means algorithm. The big difference is that while in these methods there is no notion of “output space” neighborhood (all units are “independent” from each other), in a SOM the units are “tied together” in the output space. It thus imposes an ordering of the neurons, that is not present in the other methods. These ties are equivalent to a strong lateral feedback, common in other competitive learning algorithms (Haykin 1999).

Let us imagine a very simple example, where we have 4 clusters of 3 dimensional training patterns, centered at four of the vertices of the unit cube: (0,0,0), (0,0,1), (1,1,0), and (1,1,1). If we trained a 2 dimensional, 4 node map, we would expect to obtain units centered at those vertices. If we use a larger map, with 16 nodes, for example, we would expect to obtain a map where the units are grouped in clusters of 4 nodes on each of the vertices (see Figure 6).

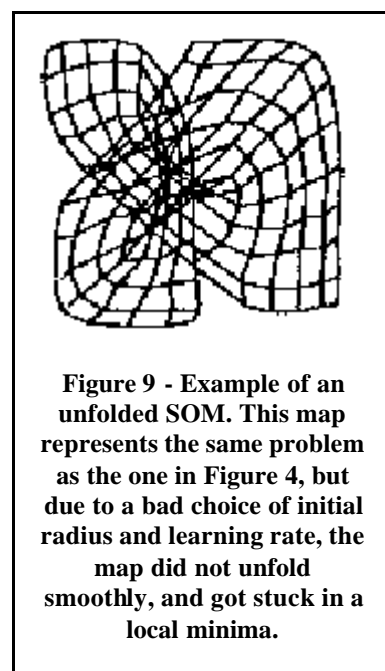
Before training, the neurons may be initialized randomly. During the first part of training, they are “spread out”, and pulled towards the general area (in the input space) where they will stay. This is usually called the **unfolding phase** of training. After this phase, the general shape of the network in the input space is defined, and we can then proceed to the **fine tuning phase**, where we will match the neurons as far as possible to the input patterns, thus decreasing the quantization error.





To visualize the training process, let us follow a 2-dimensional to 1-dimensional mapping presented in (Kohonen 1995). In this problem, 2-dimensional data points are uniformly distributed in a triangle, and a 1-dimensional SOM is trained with these patterns. Figure 7 represents the evolution of the units in the input space. As training proceeds, the line first unfolds (steps 1 to 100), and then fine-tunes itself to cover the input space.

Another very common example of a SOM mapping, that is used by the standard MATLAB demo of its neural network toolbox, is presented in Figure 8. There, a 2D map is trained on a collection of 2D points uniformly distributed in a square area. The position of the units in the input space is then tracked. This example is also useful to illustrate a rather annoying problem that may arise: local minima. In Figure 9 we can see a representation in the input space of a SOM that got stuck in local minima. In that case the map did not unfold properly, and fine adjustments to the positions of the units will not lead to a better mapping, just like when a rope gets tangled. Although it is not easy to identify unfolded maps of very high dimensional data, a good choice of learning parameters can greatly reduce the risk that they will occur.



4.3.2 - The basic learning algorithm

The basic SOM learning algorithm may be described as follows.

Algorithm 3 - SOM training algorithm (for a 2-dimensional map)

```

Let
    X be the set of  $n$  training patterns  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ 
    W be a  $p \times q$  grid of units  $\mathbf{w}_{ij}$  where  $i$  and  $j$  are their
        coordinates on that grid
     $\alpha$  be the learning rate, assuming values in  $]0,1[$ ,
        initialized
        to a given initial learning rate
     $r$  be the radius of the neighborhood function  $h(\mathbf{w}_{ij}, \mathbf{w}_{mn}, r)$ ,
        initialized to a given initial radius

1 Repeat
2     For  $k=1$  to  $n$ 
3         For all  $\mathbf{w}_{ij} \in W$ , calculate  $d_{ij} = \|\mathbf{x}_k - \mathbf{w}_{ij}\|$ 
4         Select the unit that minimizes  $d_{ij}$  as the winner  $\mathbf{w}_{winner}$ 
5         Update each unit  $\mathbf{w}_{ij} \in W$ :  $w_{ij} = w_{ij} + \alpha h(\mathbf{w}_{winner}, \mathbf{w}_{ij}, r) \|\mathbf{x}_k - \mathbf{w}_{ij}\|$ 
6         Decrease the value of  $\alpha$  and  $r$ 
7 Until  $\alpha$  reaches 0

```

This algorithm can be applied to a SOM with any dimension, making the necessary adjustments to the indexes of the units. The learning rate α , sometimes referred to as η , must converge to 0 so as to guarantee convergence and stability for the SOM. For the same reasons, the radius of the neighborhood function should also converge to 0. The decrease from the initial values of these parameters to 0 is usually done linearly, but any function may be used. The update of these two parameters may also be done after each training pattern is processed (as happens in SOM-PAK), instead of after the whole training set is processed, as described above and implemented in DSOM (see Part III).

Step 3, where the distances between a given training pattern and all units is calculated, is called the *calculation phase*. The distance measure between the vectors is usually the Euclidean distance, but many others can and are used, such as norm based Minkowski metrics, dot products, director cosines and Tanimoto measures (Garavaglia, 1996).

Step 4, where the closest unit is selected as winner is called the *voting phase*. Finally, step 5, where the units are actually changed is called the *updating phase*. The winner is sometimes also called the *best matching unit*, or BMU for short.

To stress the simplicity of the algorithm and its three important steps, the algorithm for training the network is can informally be stated as:

For each input pattern:

- a) Calculate the distance between the pattern and all units of the SOM ($d_{ij} = \|x_k - w_{ij}\|$)

This is what we call the *calculation phase*.

- b) Select the nearest unit as winner w_{winner} ($w_{ij} : d_{ij} = \min(d_{mn})$).

This is what we call the *voting phase*.

- c) Update each unit of the SOM according to the update function

$$w_{ij} = w_{ij} + \alpha h(w_{winner}, w_{ij}) \|x_k - w_{ij}\| \quad (26)$$

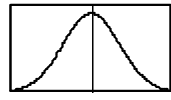
This is what we call the *updating phase*.

- d) Repeat the steps a) to c), and update the learning parameters, until a certain stopping criterion is met. Usually, the stopping criterion is a fixed number of iterations. To guarantee convergence and stability of the map, the learning rate and neighborhood radius are decreased in each iteration, thus converging to zero.

4.3.3 - Neighborhood functions

The neighborhood function, sometimes referred to as Λ or N_c , assumes values in $[0,1]$, and is a function of the position of two units (a winner unit, and another unit), and radius. It large for units that are close in the output space, and small (or 0) for units far away. Usually, it is a function that has a maximum at the center, monotonically decreases up to a radius r (sometimes called the neighborhood radius) and is zero from there onwards. For the sake of simplicity, this radius is sometimes omitted as an explicit parameter.

The two most common neighborhood functions are the bell-shaped (Gaussian-like) and the square (or bubble):

$$h_g(w_{ij}, w_{mn}) = e^{-\frac{1}{2} \left(\frac{\sqrt{(i-n)^2 + (j-m)^2}}{r} \right)^2}, \quad \text{and} \quad \text{square function} \quad (27)$$


and

$$h_s(w_{ij}, w_{mn}) = \begin{cases} 1 & \Leftarrow \sqrt{(i-n)^2 + (j-m)^2} \leq r \\ 0 & \Leftarrow \sqrt{(i-n)^2 + (j-m)^2} > r \end{cases} \quad \begin{array}{c} \square \\ \square \\ \square \end{array} \quad (28)$$

In both cases, we force $r \rightarrow 0$ during training to guarantee convergence and stability.

It must be noted that the neighborhood function depends only on the distance in the output space, i.e., the relative position of the units in the grid. This neighborhood function is responsible for the coupling between units, since when one is updated, its neighbors are updated too. This coupling in turn gives SOM its topological properties.

The algorithm is surprisingly robust to changes in the neighborhood function, and our experience is that it will usually converge to approximately the same final map, whatever our choice, providing the radius and learning rate decrease to 0. The Gaussian neighborhood tends to be more reliable (all our runs would converge to almost exactly the same map), while the bubble neighborhood leads to smaller quantization errors. A theoretic discussion of the effect of neighborhood functions (although only for the 1-dimensional case) can be found in (Erwin, Obermeyer *et al.* 1991), and a less rigorous but more general one in (Ritter, Martinetz *et al.* 1992).

4.3.4 – Theoretical aspects

A general and thorough theoretical description of the behavior of SOM has proved to be extremely difficult. There has been a lot of research in that area, excellently summarized in (Cottrell, Fort *et al.* 1998).

One of the central points of that research is to find a relationship between the underlying probability distribution of the data and the distribution of the units on a SOM. Generally, that work, together with a lot of experimental evidence, points to the fact the probability density of units on a SOM is proportional to a power of the underlying probability density of the data patterns. This power, known as a *magnification factor*, sometimes estimated at $d/(d+2)$, d being the dimensionality of the problem, will cause the SOM to under-represent areas where the probability density of the data is very high, and over represent areas where it is lower. In many applications, such as the one discussed in part III of this thesis, this scaling is a very desirable result, since areas with a very high probability density will be well represented anyway.

Another important aspect of theoretical research is to determine exactly what the learning rule is minimizing. Most neural networks have an energy function that is minimized during the training process, and it would be important to identify that function for a SOM. Assuming that Kohonen's original learning function is a gradient descent method, then by finding the primitive of that function and summing over all the network (Hertz, Krogh *et al.* 1991) we have

$$V(w) = \frac{1}{2} \sum_x \sum_i h(i, i_{winner}) |\bar{x} - \bar{w}_{winner}|^2 = \frac{1}{2} \sum_x \sum_k M_{x,k} \sum_i \sum_j h(i, k) (x_j - w_{ij})^2, \quad (29)$$

where $V(x)$ is the global energy function, and M is the *cluster membership matrix*, that encompasses the information about the neighborhood between each data pattern and the SOM units. This energy equation really does not help much, because of the difficulty in dealing with the concept of a winning unit that varies from pattern to pattern, and iteration to iteration. The cluster membership matrix can thus only be computed for a very particular instance, and will change during the training process.

If we consider the neighborhood function to be a discrete delta function, which means considering that the neighborhood radius is zero, then the energy function simplifies to

$$V(w) = \frac{1}{2} \sum_x \sum_i |\bar{x} - \bar{w}_{winner}|^2. \quad (30)$$

As noted by (Kaski 1997), if we consider the winning unit for each data pattern to be the centroid of the cluster it belongs to, this is exactly the function minimized by the k -means algorithm described earlier. Thus, a SOM with a fixed and zero radius is equivalent to k -means clustering. Such a neighborhood would also invalidate the topological ordering of the SOM for, as a k -means algorithm, there would be no relation amongst neighboring units of the SOM. However, this comparison can shed some light on the theoretical aspects of the SOM.

4.3.4 – Using SOM

The training of a SOM is more effective if it is done in two phases: the unfolding phase, and the fine-tuning phase.

For the unfolding phase, the objective is to make the SOM cover the general area where the data patterns are located, without any strong distortions or “folds”. To achieve this, the neighborhood function should have a large initial radius, so that all units are adjusted in each learning step. A

large initial learning rate should also be used, so that the map can quickly cover the input space of the patterns. Our experience points to using an initial radius just slightly less than the smallest side of the map, and an initial learning rate of 0.2.

For the fine tuning phase, the objective is to reduce the quantization error, and center the units in the areas where the density of patterns is greatest. Whereas the general mapping of the patterns does not change much during this phase, if we use it as a classifier (as we shall see later), the error rate does decrease after this phase. Our experience points to using an initial radius of 3 to 5 units for this phase, and an initial learning rate of 0.05.

After obtaining the SOM, it is useful to calculate the quantization error for the training set. This will allow us to have an idea how well the SOM represents the data. A high value for the quantization error would indicate that we either need more units, or if there are enough units, need to perform more training steps.

When the training patterns have labels (or classes) associated with them, as is the case in supervised learning problems, we may assign labels to the units of the SOM. This process is called calibration by (Kohonen 1995), but the more generic term labeling will be used in this thesis. To label a SOM, we map to it all the training set, and record for each unit the labels of patterns that were mapped to it. Each unit can then be assigned the most occurring label. A SOM thus labeled can be used as a classifier: simply map a new pattern to the map (i.e., find the unit closest to it), and use the label of that unit as the assigned class.

The number of units in the SOM can vary a lot with what is required from it, and different authors have radically different approaches. Most will use far less units than training patterns available. This will lead to SOMs where each unit maps a large number of training patterns, and thus covers a fair amount of input space. However, even in this case, if there is a clear separation in the input space between the different clusters, there will be units that because they are “pulled” both ways, will end up being positioned in the regions between the clusters, and may not map any of the training patterns.

Other authors, such as (Ultsch and Li 1993) and (Guimarães and Urfer 2000) actually use more units than training patterns. This originates SOMs where a large number of units do not map any training pattern. It can however provide a very detailed and smooth mapping of the training data,

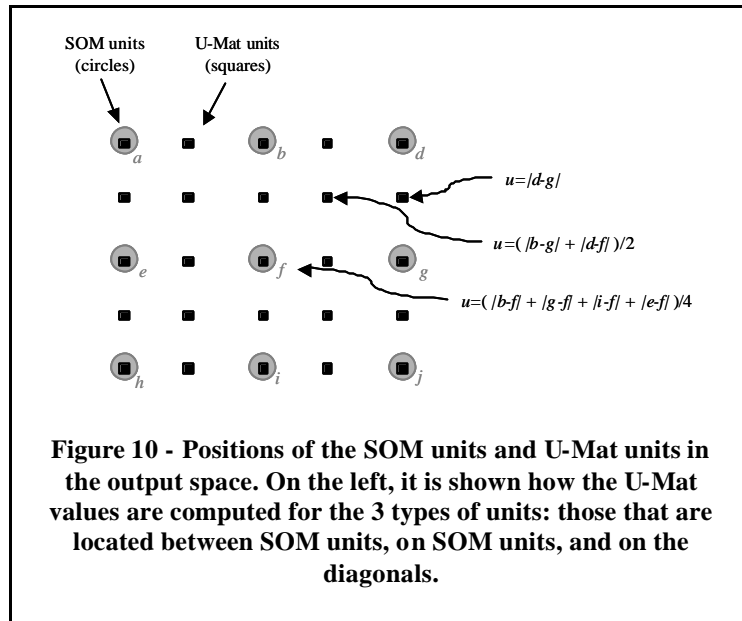
allowing the identification of small clusters, and leading to very informative U-Matrices, that shall be seen later in this chapter.

In any case, the number of units should be large enough to map each of the clusters with several units. Using too few units per cluster will make it impossible to represent clusters that do not have very regular and convex shaped distributions. One of the strong points of connectionist models is precisely the ability to distribute the information about a certain class over a number of units, and using too few of these defeats this purpose. The existence of units that do not map any training patterns can also be very desirable. On one hand, these units clearly mark the boundaries between the clusters. On the other hand, they can be useful for novelty detection. If when using a SOM, a new pattern is mapped onto these units, we will know that it is significantly different from the patterns used to train the map. However, due to the topological mapping, we will be able to have an idea how similar it is to which clusters. This is one of the characteristics that made the use of SOM particularly suited to our problem of identifying ship noise, discussed in part III of this thesis.

4.3.5 – U-Matrices

Viewing the output space of a SOM will show where the various data patterns are mapped, but will give little information about how far they are from each other. The distance in the input space between two neighboring units may vary widely amongst different areas of the SOM, so we have little information about how close different patterns really are. Also, the identification of clusters, specially when using unlabelled data patterns, can be very difficult. If there are sufficient units so that many do not map any pattern, then the areas where we have these units can be seen as borders between clusters. We will not, however, be able to say how separate those clusters are, and if there is a strong overlap of the underlying probability distributions, that separation will be impossible to see. A partial solution to this problem is to keep the count of how many patterns are mapped to each unit. This however will mask clusters with small numbers of patterns. A better solution is to use U-Matrices, or U-Mat for short.

U-Matrices were originally proposed in the end of the 80s by Ultsch (Ultsch and Simeon 1989; Ultsch and Siemon 1990), and Ultsch claims that the U stands for “Unified-distance” or “Unification”. They are computed by finding the distances, in the input space, between neighboring units in the output space.



This initial concept of U-Matrix would define values only for points between the original units of the SOM. To obtain a more usable matrix, it is usually extended to include the positions of the original SOM units, as well as points in the centers of 4 neighboring points as shown in Figure 10.

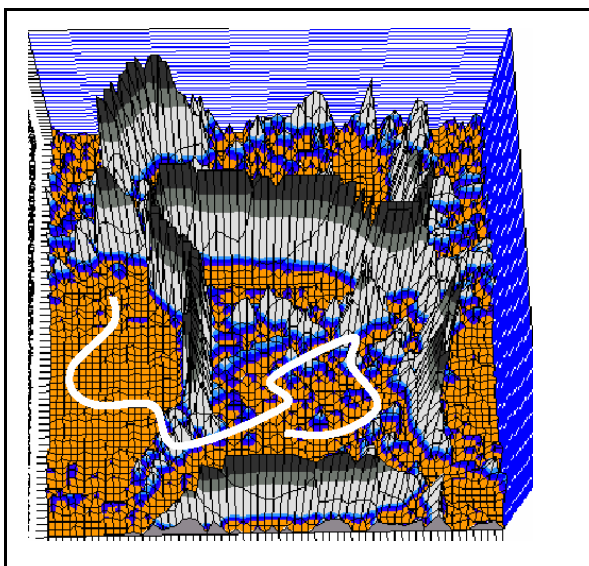


Figure 11 - Example of a 3D representation of a U-Mat, taken from (Guimarães and Urfer 2000). The central cluster is clearly separated from the rest of the map by a high ridge, and the white line represents a succession of states present in a certain patients data.

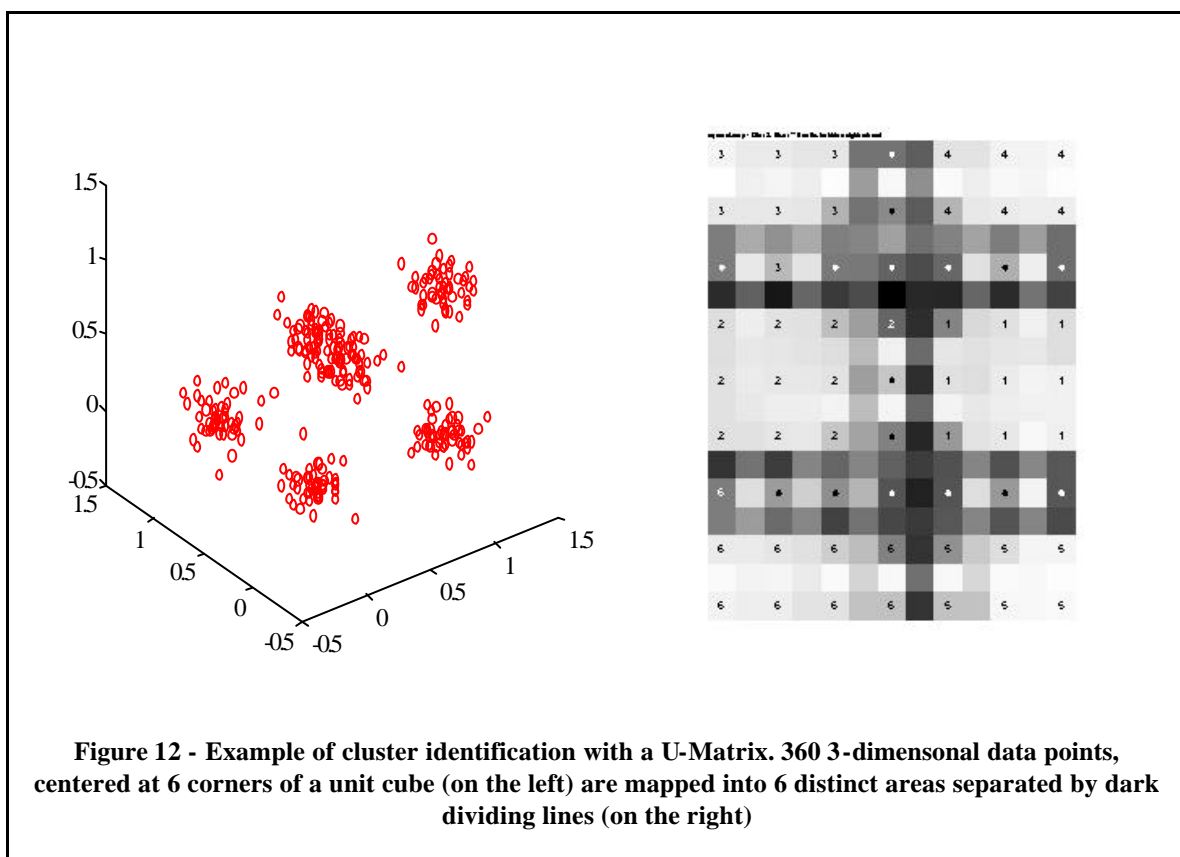
A low value for a U-Mat unit means that the SOM units are close together in the input space, and thus probably form a cluster. A high value for a U-Mat unit means that the SOM units, although neighbors in the output space, are quite distant in the input space, and thus there is a border between clusters in this area.

The two commonly used ways of visualizing U Mats. The first is to represent it in a 3D plot, where vertical dimension (the height) is given by the magnitude of the U-Mat unit at each point. The result will be a landscape where valleys represent areas where clusters of SOM units are grouped,

and ridges will represent separations between those clusters. An example of this representation is given in Figure 11.

Another way to visualize U-Mats, and probably the most common, is to color-code the values of the U-Mat. Usually a grayscale is used, with the highest value being represented by black and the lowest by white. So as make distinctions between clusters clearer, some sort of compression may be used, as is the case in the application described in Part III of this thesis.

To illustrate the power of U-Matrices and SOMs for cluster detection, we present a simple example, where 3-dimensional points are mapped with a SOM, and the clusters identified with a U-Matrix. The 360 data points have a Gaussian distribution centered at 6 of the vertices of a unit cube. We first train a 9x7 unit SOM with the data, and then compute and visualize its corresponding U-Matrix. To understand the mapping performed, we then labeled the SOM and U-Mat.



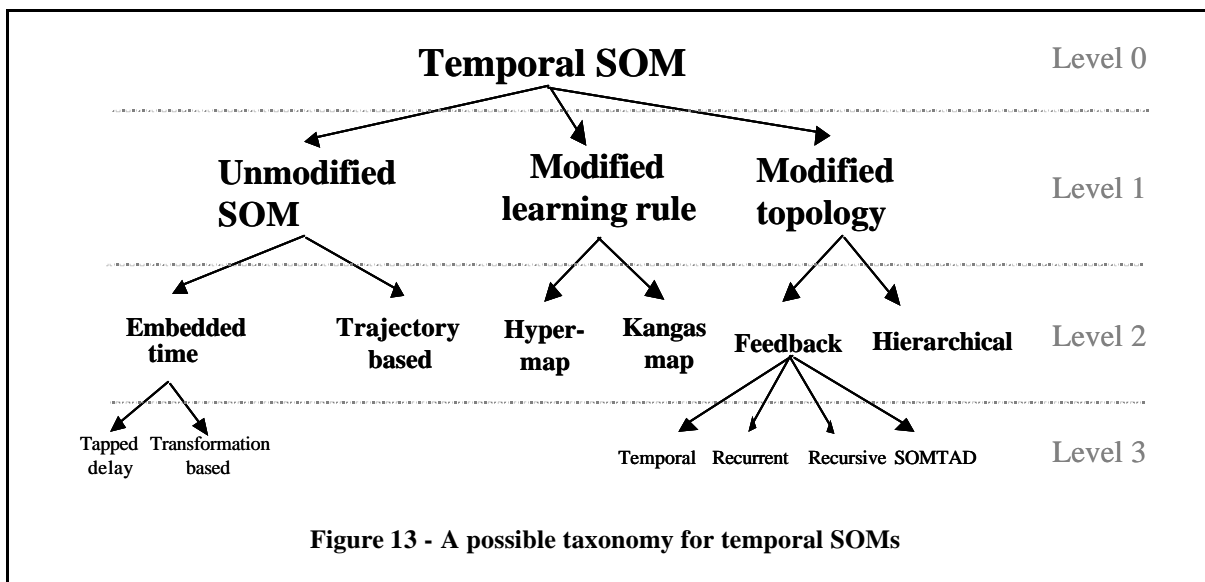
4.3.6 - Temporal SOMs

The original SOM algorithm does not take time into consideration when analyzing the patterns. However, many approaches have been used so that SOMs may process temporal data, and since ship noise signals are temporal data, we will overview some of them. We conducted a survey of temporal SOMs, and tried to define a taxonomy for them, together with Gabriela Guimarães. The results are pending publication, and will be summarized here.

Any taxonomy for temporal SOMs will only be an orienting guideline, and not a rigid classification. In fact the various ways of incorporating time are many times blended together in actual applications so as to achieve the optimum results. Nevertheless, we can identify 3 main approaches, with sub-variants:

- a) **Use a standard SOM**, and incorporate time in the pre-processing or post-processing.
- b) **Modify the learning rule** to reflect the time dependency of successive patterns.
- c) **Modify the topology** of the SOM, either by introducing feedback or by using a hierarchy of SOMs to deal with different time scales.

Our complete taxonomy is presented in Figure 13, and we shall now briefly discuss each approach.



4.3.6.1 – Unmodified SOM

In this section we discuss two distinct approaches of SOMs for handling temporal sequences that do not afford a modification of the original algorithm or network topology. One of the approaches concerns the pre-processing of a temporal sequence before presenting it to the neural network, and therefore embedding time into the pattern vector. The other approach is related to some kind of post-processing of the network outputs, resulting in a time-related visualization (or processing) of the data on the map with trajectories. In the following subsections both approaches, and several related applications will be presented.

4.3.6.1.1 - SOMs with Embedded Time

Basic Idea

The common denominator of embedded time approaches is that some sort of pre-processing is performed on the time series *before* it is presented to the SOM. Thus, the SOM receives an input pattern that is treated in the standard manner, as if time was not an issue.

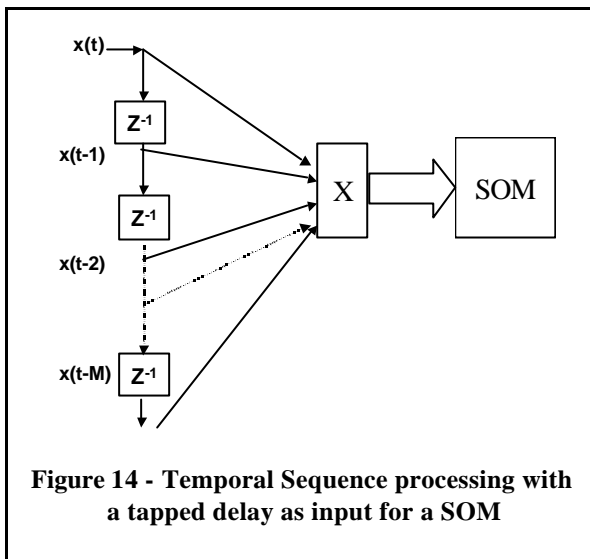
Variants

There are several ways to “hide“ time in the pattern vector, which may require more or less pre-processing and knowledge of the underlying process. A simple tapped delay will provide the easiest way of generating a pattern vector. On the other hand, a complex feature extraction algorithm may be used to generate that vector.

Variant 1 - Tapped delay SOMs

The simplest pre-processing step used when applying SOMs to temporal sequences, is to use a tapped-delay of the input as pattern vector (Chappelier and Grumbach 1995). The SOM is thus presented with a pattern that is a vector of time-shifted samples of the temporal sequence, i.e. it receives a "chunk" of the temporal sequence instead of just its last value, as is shown in Figure 14. This approach was followed by some of the early applications of SOM (Kangas, Kohonen *et al.* 1990), and is still quite popular when feature extraction techniques (e.g. Fourier transforms, envelopes, etc.) are not necessary (Príncipe and Wang 1995)). Some authors also name it Time-Delay SOM (Kankas 1994), since the approach is basically the same as the popular

backpropagation-based Time-Delay Neural Networks (TDNN) proposed by (Lang & Hinton, 1988) and (Waibel, Hanazawa *et al.* 1989).



This is a very intuitive and simple way of introducing time into the SOM, and has proved to give good results in many situations. It does however have a few known drawbacks.

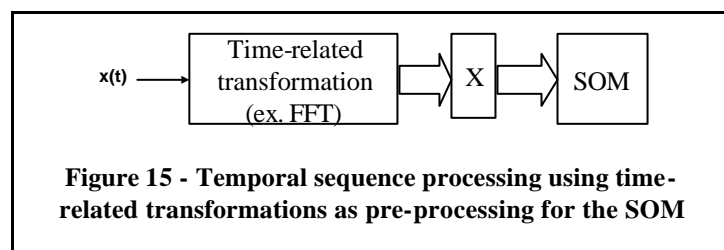
On one hand, the length of the tapped delay (the number of samples used) has to be decided in advance, and the ideal length may be quite difficult to

determine. If too few time points are used, the dynamics of the sequence will not be captured. If too many are used, apart from having an unnecessarily complex system, it may be impossible to isolate smaller length patterns. This problem also arises in other approaches, as discussed in (Davey, Hunt *et al.* 1999; Principe, Euliano *et al.* 2000).

On the other hand, since the basic SOM is not sensitive to the order in which the different dimensions of the input pattern are presented, it will not take into account the statistical dependency between successive time points. It is interesting to note that using this approach, the order of the successive time points in the final pattern vector is irrelevant (as long as that order is kept constant).

Variant 2 - Time-related transformations

In many applications, there are features of temporal sequences that are better perceived in domains other than time. The general structure of this approach can be seen in Figure



15. The most commonly used domain is frequency, and the most used technique is to perform a short-time Fourier transform on the data (Kohonen 1988). Many other transformations have been used, such as cepstral features (Kangas, Tarkkola *et al.* 1992), wavelet transforms (Pesu,

Ademovic *et al.* 1996; Moshou and Ramon 2000; Lakany 2001), and time-frequency transformations (Atlas, Owsley *et al.* 1996; Jossa, Marschner *et al.* 2001). In fact, many of the practical applications of temporal SOMs use some sort of time-related transformation as a first step in the pre-processing of the data, even if time is taken into account at a later stage. The success of these techniques is strongly dependent on the characteristics of problem at hand, and has little to do with the inherent properties of the SOM.

Discussion

These types of approaches, where only pre-processing of the data is used to deal with time, have the advantage that they preserve all the well-known characteristics of the SOM algorithm. Moreover, from a purely engineering point of view, they allow a simple integration of standard SOM software packages with the desired pre-processing software. These techniques of embedding time into the pre-processing are quite universal, and can be used to adapt almost any pattern-processing algorithm to temporal sequence processing.

Examples

One of the early papers on SOMs (Kohonen, Makisara *et al.* 1984) uses this technique. In this paper, what would later be known as the “phonetic typewriter”, was prototyped.

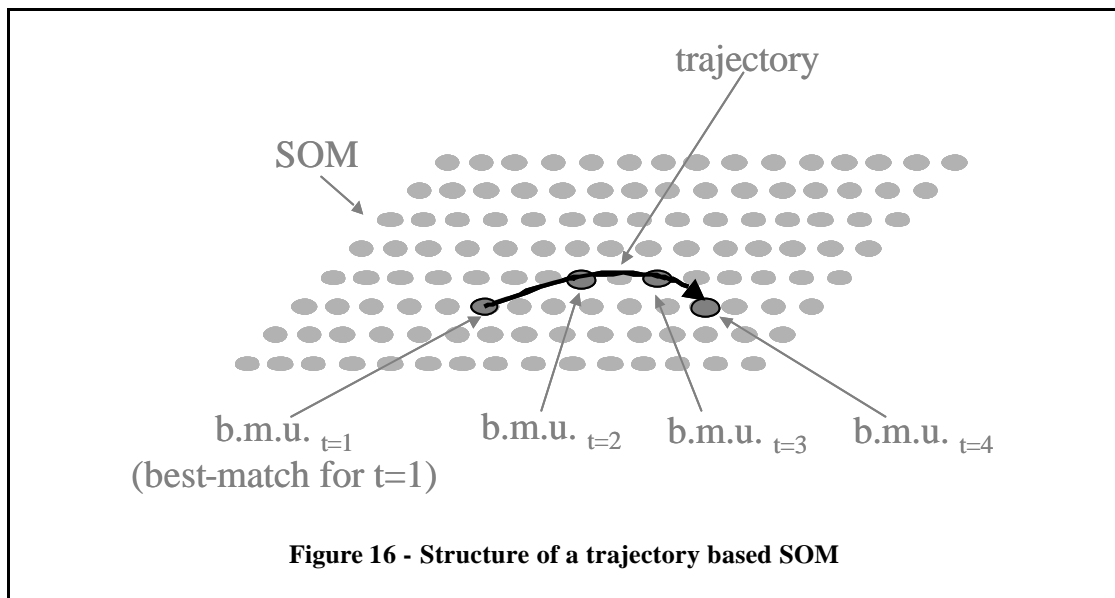
In (Leinonen, J. *et al.* 1992), for example, where the objective was to detect dysphonia, the short time power spectra of each 9.83 s chunk of signal (spoken Finnish) was calculated using 256 point FFT. The logarithms of that power spectra were then calculated, and smoothed with a low pass filter. Finally 15 of the resulting bins were selected as features, and fed into a basic SOM.

The work presented in chapters 3 and 4 of part II of this thesis also embed time in the data pattern.

4.3.6.1.2 - Trajectory-based SOMs

Basic idea

Apart from pre-processing the inputs, we can also consider using a basic SOM, without considering time during the learning process, and then post-process the results obtained during the classification phase. The most popular of these methods are what we call Trajectory-based SOMs. These consider temporal relations among succeeding best-match units. This means that at each time point $t=1, \dots, N$, the best-match $u_{i,t}$, $t \in \{1, \dots, N\}$, representing the input vector is searched and recorded on the map. Then, a representation of time-related input vectors on the map is made by joining k succeeding best-matches $u_{i_1}, \dots, u_{i_{k-1}}, i \in \{1, \dots, N-k\}$ connected forming a path, as can



be seen in. These paths are often named trajectories (hence the name of this technique), and a graphical representation is given in Figure 16.

Discussion

Trajectory-based SOMs, as opposed to the approaches presented before, constitute a genuinely new way of dealing with time. It is impossible to use “trajectories” in methods such as feed-forward neural networks or classical filters, because the topological information provided by SOMs is missing. In fact, these trajectory-based methods are successful because they explore this topological ordering, extrapolating it into the time domain.

It is interesting to see that even though training is done ignoring any sort of time dependency (and thus training data may be collected in any manner), temporal information can be recovered during the classification phase, revealing structures of the underlying process.

Another interesting feature of these methods is that information can be obtained from the direction of the path and not its exact location. Thus, for example, if we have a map trained with faulty instances of a given process, we do not need to wait until that region of operation is reached, i.e., if the winning unit moves towards that region, we can predict something is wrong before it actually occurs (Tryba and Gosser 1991; Ultsch 1993).

When processing the trajectories, it may also be important to determine the amount of successive best matches to consider, i.e., the temporal length of the trajectory. This problem is similar to the problem of determining the number of time points in a tapped delay, mentioned before.

Trajectories are often combined with other visualization techniques for the graphical representation of the weights of a learned SOM. These are, for instance, component maps where one of the components of the weights is projected onto a third dimension, as well as U-Matrices (Ultsch and Siemon 1990), where the distances between neighboring units calculated in the original space, i.e. the weights, are projected onto a third dimension. Often these additional visualization techniques lead to an enhanced interpretation of the trajectory.

Other interesting visualization techniques for SOMs have also been proposed, such as the agglomerative clustering where the SOM neighborhood relation can be used to construct a dendograms on the map (Murtagh 1995; Vesanto and Alhoniemi 2000), and a hierarchical clustering of the units on the map with a simple contraction model (Himberg 2000). Although these approaches have not yet been used in the context of temporal sequence processing, they would enable a richer perception of the significance of the trajectory, allowing varying levels of detail in the analysis of the inputs.

For most applications trajectories have been directly displayed on a map without a visualization of the network weights (Kohonen 1988; Leinonen, Hiltunen *et al.* 1993). In those cases a direct interpretation of the trajectory is possible if a prior classification of the signal exists, as for example, in different phone me types in speech recognition (Kohonen 1988) or distinct sleep stages in EEG signals (Kaski and Joutsiniemi 1993). However, if the SOM is used as a feature

extractor, the trajectory itself, regardless of any labeling, can be used as a temporal feature of the input, and fed to a higher level system (Srinivasa and Ahuja 1999). For example, if we train an unlabeled SOM with phonemes, a given word will have a distinct path that would distinguish it from other words. If we are using the SOM as a visualization tool and no prior information on the classes is known, a combination of trajectories with other visualization techniques for SOMs mentioned earlier (component maps, U-matrices, and hierarchical clustering visualizations), can be very useful.

Component maps enable to track the trajectory along a single component. This can be advantageous, if we are interested in evaluating the contribution of each of the components to the system's state changes. Notwithstanding, if a large number of variables have to be considered, this approach can originate some confusion and unclearness to the observer. In order to overcome these disadvantages, we will have to observe the development of a complex system or process on a single map using, for instance, U-matrices.

The main advantage in visualizing trajectories on U-matrices lies in the identification of state transitions. These transitions are clearly seen on a U-matrix, because when one such transition occurs, the trajectory of the best-match unit has to overcome a "wall". This means that in the original space a large distance has to be traveled, if a trajectory jumps over a wall, even if the distances on the map itself are small, i.e. they are neighboring units. This type of interpretations is not possible if the trajectories are observed only on the SOM itself.

Examples

The visualization of trajectories on the map itself was first applied to speech recognition (Kohonen 1988). Here a decomposition of a continuous speech signal is performed in order to recognize phonetic units. Before presenting the data to the network, a transformation into the frequency domain is made. A map, named here as phonotopic map, was generated with the input vectors representing short-time spectra of speech waveform computed every 9.83 milliseconds. One of the most striking results was that various units of the network became sensitized to spectra of different phonemes based only on the spectral samples of the input. However, in this approach samples only correspond to quasi-phonemes. Now, one of the problems lies in the segmentation of quasi-phonemes into phonemes. For this purpose, the degree of stability of the waveform, heuristic methods, and trajectories over a labeled map were calculated. Convergence

points of the speech waveform then may correspond to certain stationary phonemes. The main advantage of phonotopic maps is that they can be used for speech training or therapy, since people can obtain immediate feedback from their speech.

This approach was also widely applied at the early 90's to several medical applications, such as the identification of co-articulation variation and voice disorder (Utela, Kangas *et al.* 1992), the detection of fictive-vowel co-articulation (Leinonen, Hiltunen *et al.* 1993), the detection of dysphonia (Leinonen, J. *et al.* 1992), the acoustic recognition of "/s/" missarticulation enabling a distinction between normal, acceptable and unacceptable articulations (Mujunen, Leinonen *et al.* 1993), the recognition of topographic patterns in Electroencephalogram (EEG) spectra from 16 subjects having different sleep/awake stages (Joutsiniemi, Kaski *et al.* 1995), and the monitoring of EEG signals enabling the identification of six typical EEG phenomena, such as well organized alpha frequencies, eye movement artifacts and muscle activity (Kaski and Joutsiniemi 1993). All these approaches have in common that a pre-classification of the original signal was already made. In applications for speech processing such a pre-classification is always possible. Within another approach for speech recognition trajectory-based SOMs have been used at different hierarchical levels (discussed later in this paper), where each layer operates on a different time scale and deals with higher units of speech, such as phone mes, syllables, and word parts (Behme, Brandt *et al.* 1993). This means that the basic structure of all layers is similar, only the meaning of the input and the time scale are different. This approach was used for the recognition of normally spoken command sentences for robot controlling, whereat the system had to deal with extra words and other insertions not part of a robot command. So, syntax and semantic modeling also played here an important role. Trajectories have only been used at the first level. They consist of stationary parts representing vowels that remain in a close neighborhood and transitions paths with jumps to different and more distant parts of the map. In order to distinguish between stationary and transition parts, a critical jump distance separating short and long distances, as well as a minimum segment length was defined.

Trajectory-based SOMs have also been proposed to model low dimensional non-linear processes, such as non-linear time series obtained from a Markey-Glass system (Príncipe and Wang 1995). They followed three steps: the reconstruction of the state space from the input signal; the embedding of the state space in the neural field; and the estimation of locally linear predictors. Trajectories are then used to obtain a temporal representation of all 400 consecutive input samples.

Within another application, firing activities in monkey's motor cortex have been measured and presented to a SOM in order to predict the trajectory of the arm movement, especially while the monkey was tracing spirals and doing center-out movements (Lin, Si *et al.* 1998). From the map, three circle-shaped patterns representing the spiral trajectory have been identified through paths on the map. The results showed that the monkey's arm movement directions are clearly encoded in firing patterns of the motor cortex.

In (Kasslin, Kangas *et al.* 1992), for instance, components maps are used for process state monitoring where values for one parameter are visualized as gray values on a map. The lighter the unit on the map, the higher the parameter value is. Their aim was to classify the system states and detect faulty states for devices based on several device state parameters, such as temperature. Faults in the system could be detected with trajectories, if a transition to a forbidden area on the map marked with a very dark color occurred. This approach was also applied to process control in chemistry for monitoring a distillation process (Tryba and Gosser 1991).

Visualization of trajectories on U-matrices have been used for monitoring chemical processes (Ultsch 1993), and have been applied to complex processes, such as the dynamic behavior of a computer systems with regard to utilization rates and traffic volume (Simula, Alhoniemi *et al.* 1996), to industrial processes, such as a continuous pulp digester, steel production and pulp and paper mills (Alhoniemi, Hollmén *et al.* 1999), and to different subjects with distinct sleep apnea diseases (Guimarães, Peter *et al.* 2001). In order to enhance exploratory tasks with SOM-based data visualization techniques, quantization error plots can be used using bars or circles on both component maps or U-matrices (Vesanto 1999).

4.3.6.2 - Modification of the Activation/Learning Rule

Another possibility for processing temporal data with SOMs lies in the adaptation of the original Kohonen activation and/or learning rule. Here we distinguish between two distinct approaches. In the first, the input vector is decomposed into two distinct parts, a past or context vector and a future or pattern vector. Both parts are handled in different ways when choosing the best match and when applying the learning rule. This approach, named Hypermap, was first introduced by Kohonen (Kohonen 1991). The second approach, that we will call Kangas map, searches for the best match in a neighborhood of the last best match.

4.3.6.2.1 -The Hypermap Architecture

Basic ideas

In this architecture the input vector is decomposed into two distinct parts, a “past” or “context” vector and a “future” or “pattern” vector. The basic idea, now, lies treating both parts in different ways. The most common way is to use the context part to select the best match or “best-match region”, and then adapting the weights using both parts, separately or together. However many variants exist, and will be discussed later.

For time series a Hypermap means that the future (prediction) is learned in the context of its past. During the classification phase the prediction is made using only the “past” vector for the best-match search. Thus, the SOM is used as an associative memory, and the “future” part of the vector is then retrieved from the weights of the map associated with the best match.

Discussion

Originally the Kohonen algorithm is an unsupervised learning algorithm that can be used for exploratory tasks. Approaches, however, that use some kind of Hypermap architecture, perform a profound change in the interpretation of the original Kohonen algorithm towards a supervised learning algorithm, since an output vector (the future or pattern vector) is added to the input (the past or context vector). This makes sense in applications that require an extrapolation of the data into the future as, for example, in time series prediction (Ultsch, Guimarães *et al.* 1996), or in robot control (Ritter, Martinetz *et al.* 1992).

Examples

This approach was first introduced by Kohonen (Kohonen 1991) and named Hypermap architecture. It was applied to the recognition of phonemes in the context of cepstral features. Each phoneme is then formed as a concatenation of three parts of adjacent cepstral feature vectors. The idea was to recognize a pattern that occurs in the context of other patterns, where $\mathbf{x}(n) = [\mathbf{x}_{pat}(n), \mathbf{x}_{con}(n)]$. A two-phase recognition algorithm for the best-match search was proposed. In the first phase, we start by selecting a “context domain”. This is done by searching for the good-matches of the context, i.e. all units that are within a given distance of the context vector $\mathbf{x}_{con}(n)$. In the second phase, the best match is searched within the selected context domain, using only the pattern $\mathbf{x}_{pat}(n)$. During learning we also have two phases. In the first

phase, a context SOM is trained using only the context $\mathbf{x}_{cont}(n)$ and the basic SOM algorithm. After this SOM is trained, its weights are frozen (i.e. made constants). Next, we perform a learning of the pattern weights of the Hypermap. This is done using the above described algorithm for the best-match search, but performing the adaptation only on the unit weights that are related to the pattern vector $\mathbf{x}_{pat}(n)$. Furthermore, for this particular application, an extra-supervised learning step was used to associate the units with phonemes.

This architecture was generalized in (Bruckner, Franz *et al.* 1992) to perform hierarchical relationships having $n-1$ levels defining the context for the classification of EEG signals from acoustical and optically evoked potentials. This type of model was also studied for phoneme recognition using the LASSO model (Midenet and Grumbach 1994), for simulating a sensory-motor task (Ritter and Kohonen 1989), as well as for robot control (Ritter, Martinetz *et al.* 1992; Walter and Schulten 1993; Ritter 1994; Walter 1998). In this latter application, the output is the target position of the robot arm (for instance, given by the angles), while the input is given as a four-dimensional vector describing the spatial position of the robot arm obtained by the images of two cameras.

Hypermaps have also been used for prediction tasks, for instance, using SOMs for local models in the prediction of chaotic time series (Koskela, Varsta *et al.* 1998). The time series is embedded in a state space using delay coordinates $\mathbf{x}(n) = [x(n), x(n-1), \dots, x(n-(N-1))]$, where N is the order of the embedding. The embedded vector is then used to predict the next value of the series $x(n+1)$. The following vector $\mathbf{y}(n) = [x(n), x(n+1)]$ is presented to the map during the learning phase. However, when searching for the best match, the target value is left out. This means that only the first part (past) of the whole vector is used for the determination of the best match. During learning, the unit weights are adapted using the whole input vector, and the standard Kohonen algorithm. Now, during the classification phase, only the first part of the vector is used for the best-match search, and indeed it is the only part available, since we are trying to predict the future. Thus this future part is obtained through an associative mapping with the past. In (Ultsch, Guimarães *et al.* 1996) a two step implementation of this approach was used for the prediction of hailstorm. First, it was used to identify distinct types of hailstorm developments. After the classification part, prediction was made using the completed vector.

4.3.6.2.1 - Kangas Map

Basic ideas

Instead of considering explicitly the context as part of the pattern, as is done in the Hypermap,

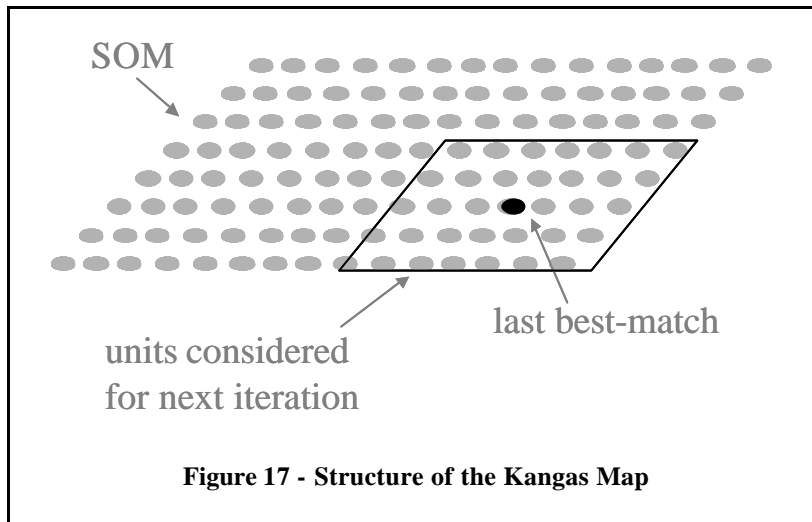


Figure 17 - Structure of the Kangas Map

we can also consider that the context is given by the previous best match, and use only the neighboring units when choosing the next one. This idea was proposed by Kangas (Kangas 1992), and so we named this approach Kangas Maps. In this approach, the learning rule is exactly the same as in the basic SOM. The

selection of the best-match is also the same, save for the fact that instead of considering all units for the next iteration step, only those in the neighborhood of the last best-match are considered, as can be seen in Figure 17.

Discussion

This type of map has several interesting features, and can in certain cases have a behavior similar to SOMs with feedback, e.g. SOMTAD (Euliano and Principe 1999), discussed later in this paper.

From a purely engineering point of view, it can be considerably faster than a basic SOM when dealing with large maps, since we only need to compute the distances to some of the units. It also requires very little change in the basic SOM algorithm, and keeps its most important properties. The area where the next best-matches are searched for acts as a “focus region”, where the changes in the input are tracked. In a Kangas map, we may have various distinct areas with almost the same information (relating to the same type of input signal), but with different neighboring areas. Thus, the activation of the units will depend on the past history, i.e. on how the signal reached that region of the map. Thus, this approach uses the core concepts of neighborhood and topological ordering of SOMs, to code temporal dependency.

In the original paper (Kangas 1992) some variants of the basic idea are proposed, though not explored in depth. One of them allows for multiple best matches, and thus multiple tracking of characteristics of the input signal.

A similar approach, that also uses feedback (discussed later) was proposed in (Chandrasekaran and Palaniswami 1995), and named Spatio-Temporal Feature Map (STFM). In a STFM, the units that are used when searching for a best-match are selected according to a rule that includes more than just the neighborhood (in the output space) to the last best-match. Two core concepts are used in this selection: a so-called spatial grating function that basically defines a spatial area of influence of each unit; and a gating function, that is a time-dependant function of past activations, and determines the output of the units. Using these two concepts, a so-called competition set of units is selected, where the winner will be searched. Finally, in the above-mentioned papers, the trajectory of the best match is also used, and named the “spatio-temporal signature” of the temporal sequence.

Many other rules may be used to select the candidate best matches.

Instead of using past activations to select candidates for best matches, we can also use those past activations to exclude certain candidates. The Sequential Activation Retention and Decay NETwork (SARDNET), proposed in (James and Miikkulainen 1995) does just this. In this approach (which also uses feedback), the best match is excluded from subsequent searches. Thus, a sequence of length l will select l different best matches, or a l -length trajectory. As discussed in (James and Miikkulainen 1995) this will force the map to be more detailed in the areas where each of the sequences occur, thus representing small variations of those sequences with greater detail. A decay factor is also introduced to make the selection of the best winner depend on past activations, but we will not discuss its influence here. This hybrid approach was used successfully to learn arbitrary sequences of binary and real number, as well as phonetic representations of English words.

Kangas Map based approaches have been used for speech recognition tasks (Kangas, Tarkkola *et al.* 1992; Kankas 1994), texture identification, and 3-D object identification (Chandrasekaran and Palaniswami 1995; Chandrasekaran and Liu 1998).

4.3.6.3 - Modification of the Network Topology

The third possibility in handling temporal data lies in modifying the network topology, introducing either feedback connections into the network or several hierarchical layers, each with one or more SOMs. Feedback SOMs are intimately related to digital filters and ARMA models, which are a more traditional way of dealing with temporal sequences. The latter approach is mainly used when a segmentation of complex and structured problems is needed in application domains, such as image recognition, speech recognition, time series analysis, process control, and protein sequence determination.

4.3.6.3.1 - SOMs with Feedback

Basic Idea

One of the classical methods to deal with temporal sequences, which have been used with great success in control theory, is to feed some sort of output back into the inputs. This is usually done with an internal memory that stores past outputs, and uses them when generating the next outputs. One of the advantages of these methods is that they do not require the user to specify the length of the time series that must be kept in memory, as happens with the tapped-delay approaches.

Variants

There are a few different values that can be used for feedback, and a few different ways to introduce these values back into the system. Thus a large number of approaches have been proposed and tested in different environments.

Variant 1 - Temporal Kohonen Maps (TKM)

Historically, the first well-documented proposal for feedback SOMs appeared in 1993 (Chappell and Taylor 1993), named as “Temporal Kohonen Map” (TKM), and is very similar to the model used in (Kangas 1992). The main idea behind this approach lies in keeping the output values of each unit and using them in the computation of the next output value of that unit. This is done introducing a leaky integrator in the output of each SOM unit. In a TKM the final output of each unit is defined as

$$V_i(n) = \alpha V_i(n-1) - \frac{1}{2} \|x(n) - w_i(n)\|^2, \quad (31)$$

where

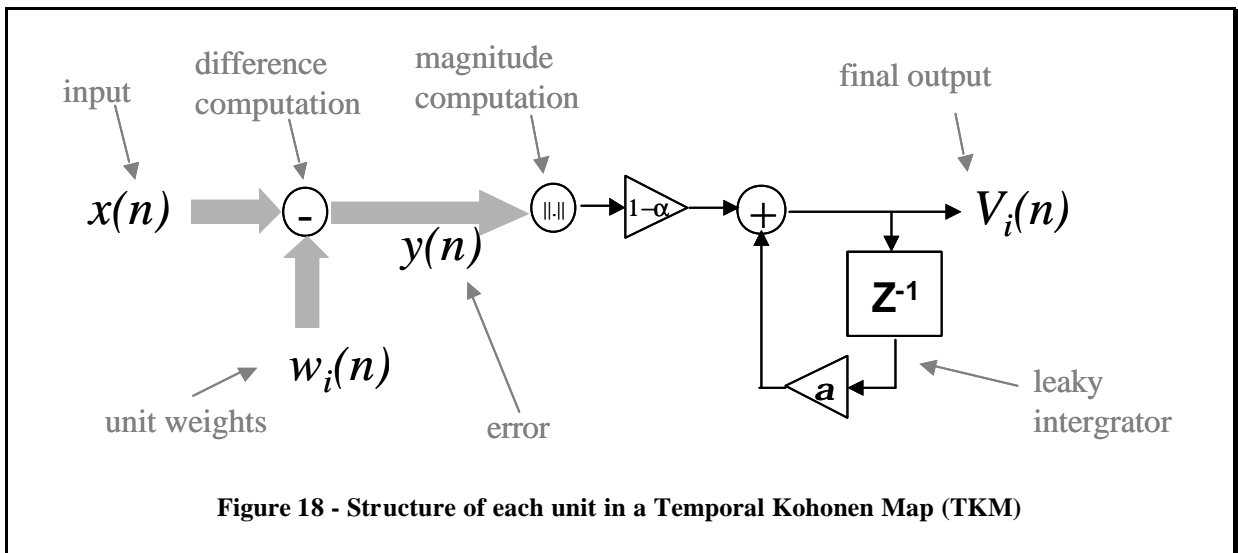
- $V_i(n)$ is the scalar output of unit i , in at time n ,
- $V_i(n-1)$ is the scalar output of unit i , in at time $n-1$,
- $x(n)$ is the input pattern presented at time n ,
- $w_i(n)$ is the SOM unit i , at time n ,
- \mathbf{a} is a time constant called decay factor, or memory factor, restricted to $0 < \mathbf{a} < 1$.

The best-matching unit is considered to that which has a higher $V_i(n)$ (which is always negative). The learning rule used is that of the basic SOM. When $\mathbf{a} = 0$, the units have no memory, and we fall into the standard SOM algorithm. It must be noted that this output transfer function resembles the behavior of biological neurons, which do have memory, and weigh new inputs with past states.

In essence, for the sake of comparing this approach with others, the activation function (to be minimized) is

$$V_i(n) = \mathbf{a} V_i(n-1) + (1-\mathbf{a}) \|x(n)-w_i(n)\| \tag{32}$$

This formulation of the TKM is shown in Figure 18.



Variant 2 - Recurrent SOM (RSOM)

The TKM keeps only the magnitude of the output of the units, and keeps no information about each of the isolated components, and thus no information about the “direction” of the error

vector. To overcome this limitation, the Recurrent SOM (RSOM) was proposed (Critchley 1994; Varsta, Heikkonen *et al.* 1997), where the leaky integrators are moved from the output to the input of the magnitude computation. As a consequence, the system memorizes not only the magnitude, but also the direction of the error.

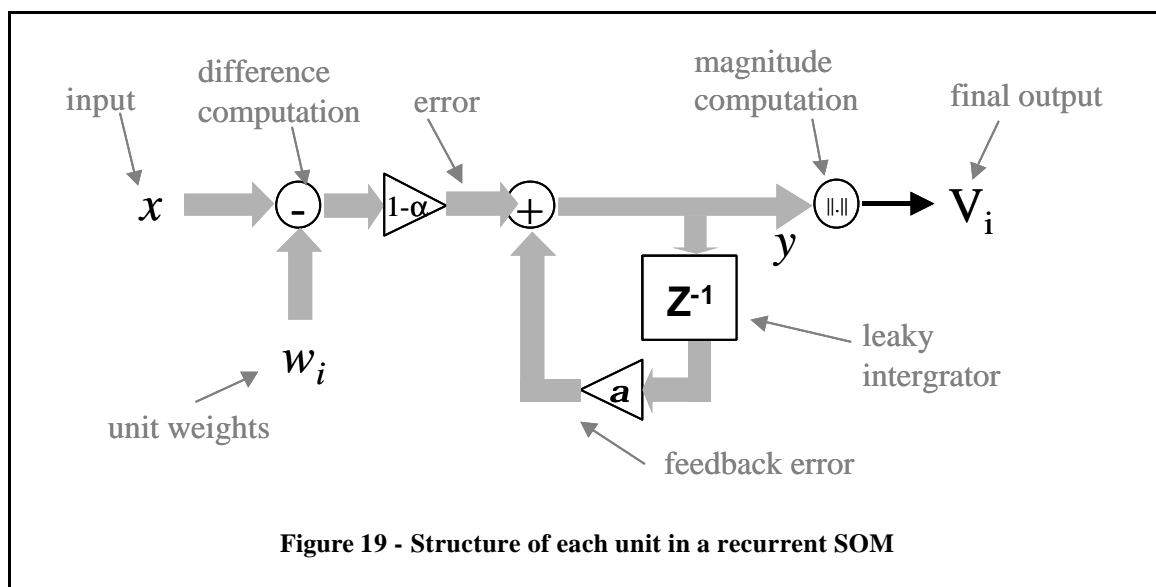
The activation function for each unit will now be

$$V_i(n) = \| y_i(n) \|, \quad (33)$$

where $y_i(n)$ is the error vector given by

$$y_i(n) = (1-\alpha) y_i(n-1) + \mathbf{a}(x(n)-w_i(n)) \quad (34)$$

This formulation of the Recurrent SOM is shown in Figure 19



Recursive SOM

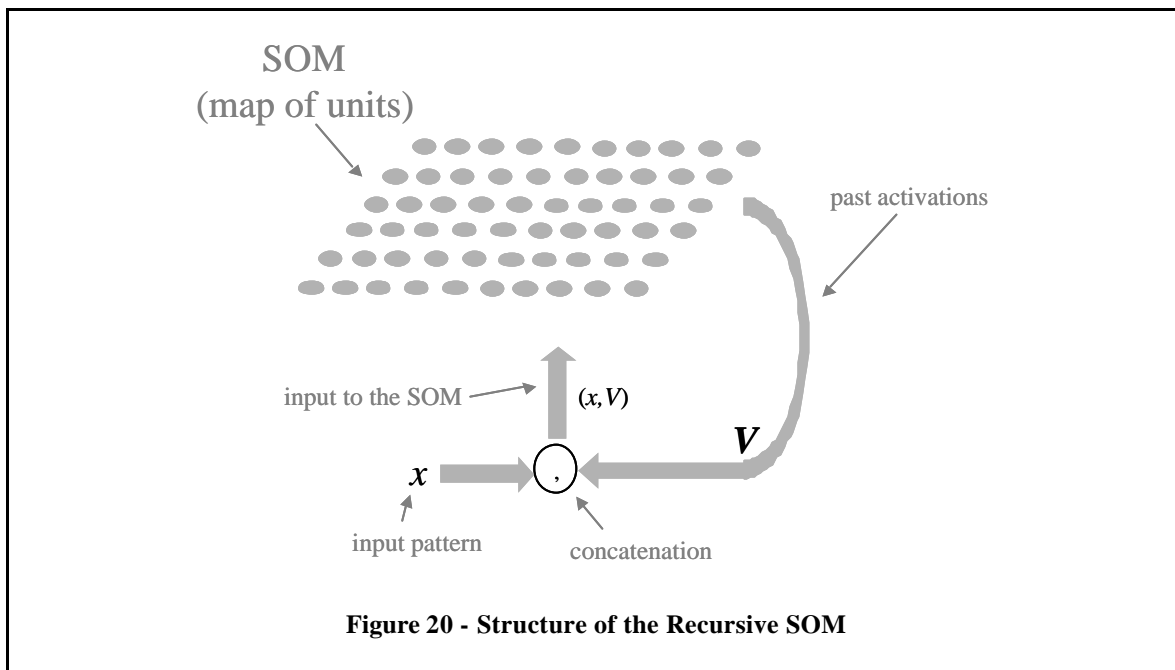
The TKM uses, for the computation of the activity of each unit, only the previous output of *that* unit. The RSOM also uses only local feedback. Another alternative is to feedback the outputs of *all* the units of the map to *each* of them. This alternative was first proposed in (Harmelen 1993), later in (Barreto and Araújo 1999), and was analyzed in detail in (Voegtlin 2000; Voegtlin and Dominey 2001), and with slight modifications in (Ruf and Schmitt 1998). The latter authors first named this approach Contextual Self-Organizing Map (CSOM), and later Recursive SOM. In this paper we use the name Recursive SOM to clearly differentiate it from the Hypermap architecture that also uses the term “context”. The dimensionality of each unit is increased significantly. Besides the standard weight vector w_i^{input} each unit i will also have a weight vector w_i^{output} , which

is to be compared with actual outputs in the previous instant. The activation function defined by (Voegtlin and Dominey 2001) is

$$V_i(n) = \exp(-\mathbf{a} \|x(n) - w_i^{input}(n)\|^2 - \mathbf{b} \|V(n-1) - w_i^{output}(n)\|^2), \quad (35)$$

where \mathbf{a} and \mathbf{b} are constant coefficients that reflect the importance of past inputs.

The formulation of the Recursive SOM is shown in Figure 20.



SOM with Temporal Activity Diffusion (SOMTAD)

Another approach, similar to recursive SOM, was proposed and analyzed in (Kopecz 1995) (Euliano and Principe 1996; Euliano, Principe *et al.* 1996; Euliano and Principe 1998; Euliano and Principe 1999) and, in the latter paper, named SOM with Temporal Activity Diffusion (SOMTAD). In a SOMTAD, instead of feeding back all past outputs (and learning their respective weights), only the activations of neighboring units are fed back. This leads to a sort of shock wave that is generated in the best match unit, and propagates throughout output space of the map. Adapting the proposed algorithm to the formalism we have been using, we will have

$$V_i(n) = (1-\mathbf{a}) V_i(n-1) + \mathbf{a} \|x(n) - w_i(n)\|, \quad (36)$$

just as in a KTM, but each unit i will also have an *enhancement* E given by

$$E_i = f(V_{neighbour}(t-1)), \quad (37)$$

where $f(\cdot)$ is some function, that couples the enhancement of one unit with the activity of its neighbor.

The best match is then

$$\text{best-match}(t) = \arg \min(\|x(n) - w_i(n)\| + \mathbf{b} E_i(t)), \quad (38)$$

where \mathbf{b} is called the spatio-temporal parameter, and controls the importance of past neighboring activations. The structure of each unit of a SOMTAD is shown in Figure 21.

It must be noted that when \mathbf{b} tends to 0, the SOMTAD becomes a standard SOM. As it increases, the behavior will be similar to a Kangas map,

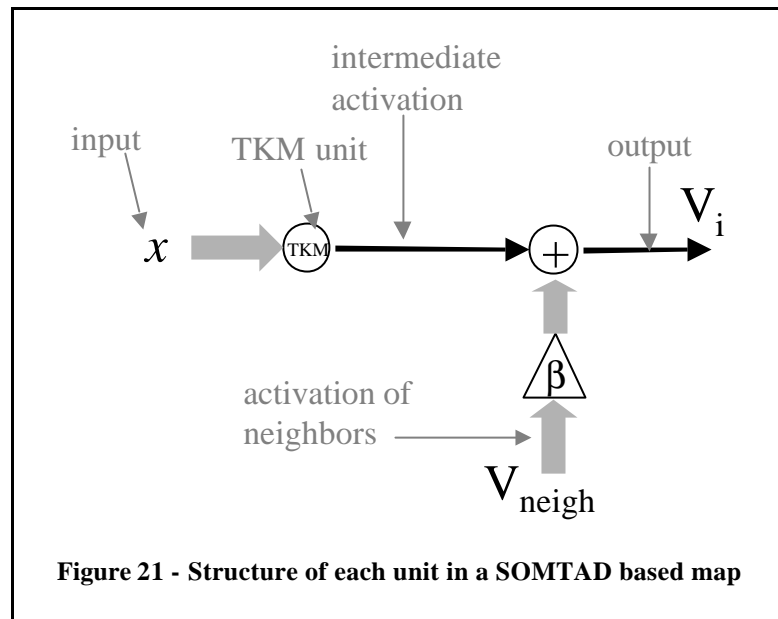


Figure 21 - Structure of each unit in a SOMTAD based map

since the best match will tend to be in the vicinity of the last best-match, and as \mathbf{b} tends to $+\infty$, the model degenerates into an avalanche network.

Discussion

None of the above proposals is universally better than any other, so one can always find an example of an application where a given approach outperforms the others. There are, however, some well-known characteristics that can help us choose a good approach for a given problem. A good theoretical comparison of the Temporal Kohonen Map (TKM) and the Recurrent SOM (RSOM), can be found in (Varsta, Heikkonen *et al.* 2000), where the authors show that TKM lacks RSOM's consistent update rule. Thus, while a RSOM will converge to optimum weights for its units, following a gradient descent algorithm, the TKM will not, hence in some way it will be unreliable. In a series of experiments, the authors show that generally a RSOM will provide a more efficient mapping of the input space signals than a TKM, which tends to concentrate its units in certain regions of that space. On the other hand in (Voegtlin and Dominey 2001) it is shown that for a classical benchmark problem, the Mackey-Glass chaotic time series, the Recursive SOM will provide a far better mapping than the Recurrent SOM. This will generally

be the case when a global perspective of the signal space is necessary, which means that in those cases, when a global perspective of the past inputs is necessary, a strictly local approach, such as TKM and RSOM, can not give good results. However, it must be noted that the complexity of the system is also considerably increased when we use a Recursive SOM.

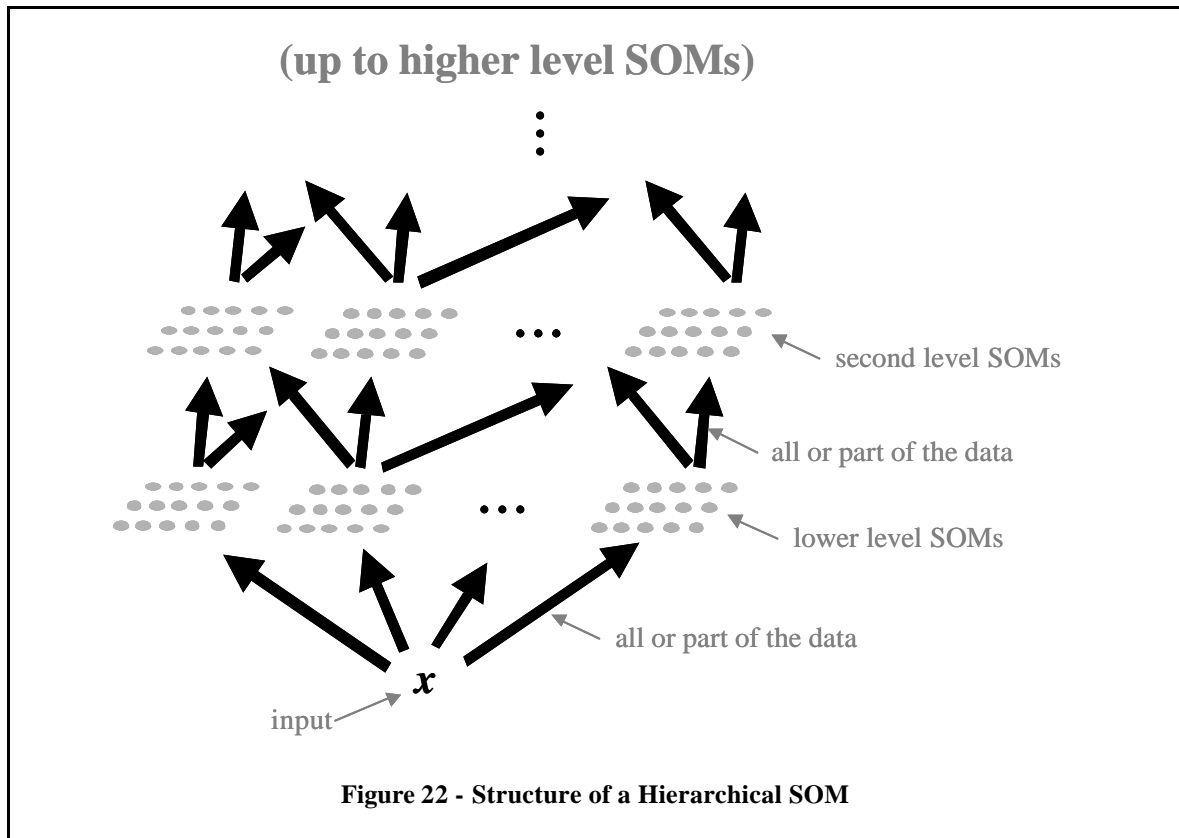
Examples

Recurrent SOMs have probably been the most used model. They have been successfully applied to the Mackey-Glass Chaotic Series, infrared laser activity, and electricity consumption (Koskela, Varsta *et al.* 1997), as well as clustering of epileptic activity based on EEG (Koskela, Varsta *et al.* 1998). As described earlier, the Recursive SOM was also used to study the Mackey-Glass series, and has outperformed the RSOM. It was also used, with slight modifications, in (Ruf and Schmitt 1998). To our knowledge, the SOMTAD model has only been applied by its authors to digit recognition, and to small illustrative problems in (Euliano and Principe 1996; Euliano, Principe *et al.* 1996; Euliano and Principe 1998; Euliano and Principe 1999).

4.3.6.3.2 - Hierarchical SOMs

Basic idea

Hierarchical SOMs are often used in application fields where a structured decomposition into



smaller and layered problems is convenient. Here, one or more than one SOMs are located at each layer, usually operating at different time scales (see Figure 22). Hierarchical SOMs in temporal sequence processing have been successfully applied to speech recognition (Kempke and Wichert 1993; Jiang, Gong *et al.* 1994), electricity consumption (Carpinteiro, Silva *et al.* 2000), vibration monitoring (Jossa, Marschner *et al.* 2001), motion planning (Barreto and Araújo 1999) and temporal data mining in medical applications (Guimarães and Urfer 2000).

Discussion

The main difference between hierarchical SOMs lies in the type of codification of the results of one level SOMs to the next upper level. They also differ in the number of levels used, which strongly depends on the type of application. Finally, they differ in the number of SOMs at each level, and the interconnections between the levels.

The rationale for using hierarchical SOMs is that of “divide and conquer”. By focusing independently on different inputs we do lose information, but we gain manageability. We can thus use a relatively low complexity model, such as a SOM to handle each of the small groups of inputs, and then fuse this partial information to extract higher-level results. Good results can thus be obtained with hierarchical SOMs in complex problems that cannot be modeled by a single SOM (Guimarães, Peter *et al.* 2001).

There are mainly three different ways to calculate the input vector for the next level SOM. First, the weights of the lower-level SOM are used as input without any further processing, either taking into account the information of the previous known classes (Kempke and Wichert 1993) or without considering any information on the classes (Walter and Ritter 1996). In this case, the first level SOM is simply being used for vector quantization. Second, a transformation of the network results is possible, for instance: 1) calculating the distances between the units (Carpinteiro 1998); 2) concatenating subsequent vectors into a single vector, thus representing the history of state transitions (Simula, Alhoniemi *et al.* 1996); or 3) taking into account the information about clusters formed at this level, and adjusting the weights towards the cluster center (Guimarães and Urfer 2000). The third possibility lies in interposing other algorithms or methods, such as segment classifiers (Behme, Brandt *et al.* 1993).

Examples

This approach was first introduced in speech recognition, where each layer deals with higher units of speech, such as phonemes, syllables, and word parts (Behme, Brandt *et al.* 1993; Kempke and Wichert 1993). For instance, in (Behme, Brandt *et al.* 1993) at each layer a SOM operating at different time scales is used, which is connected to a segmentation unit for the segmentation of the input and to segmentation classifiers. Each of the classifiers was trained to recognize a special class of segments, thus producing an output vector for each segment. These output vectors, i.e. the activities of the classifiers, form the input for the next level, which thus operates on a larger time scale. Each segment classifier m produces an output activity $a_m = \mathbf{S}_i c_i w_{im}$, c_i denoting the activity of the SOM unit i and w_{im} the synaptic strength from SOM unit i to classifier m . These connections may be inhibitory ($w_{im} < 0$), but always fulfill $\mathbf{S}_i w_{im}^2 = 1$.

In (Jiang, Gong *et al.* 1994) a speaker recognition system based on the auditory cortex model is proposed. Since an auditory cortex can be generally considered as a layered upward structure

with complex connections, three hierarchical levels of SOMs with local connections have been introduced. The output of the first map contains leaky integrators and is calculated as follows:

$$y_i(n) = \mathbf{a} \times (k / (1 + \|w_i(n) - x(n)\|)) + (1 - \mathbf{a}) \times y_i(n-1), \quad 0 < \mathbf{a} < 1. \quad (39).$$

For the sake of comparison with other approaches, this equation can be transformed into the following that should be minimized:

$$y_i(n) = (1 - \mathbf{a}) \times y_i(n-1) + \mathbf{a} \|w_i(n) - x(n)\|. \quad (40)$$

The units of the second and the third layer have input connections from the units of the immediately lower level. Additional connections exist from the first to the third level.

Kemke and Wichert (Kempke and Wichert 1993) also used hierarchical SOMs at different time scales, as mentioned before. The codification of the input at the next layer is based, however, on a pre-classification of the signal. A class is associated with each unit on the map. In order to calculate the output for a given input vector, the mean of the weights of all units belonging to the class is calculated, and used as input to the map of the next higher-level map.

Hierarchical SOMs have also been applied to monitoring and modeling the dynamic behavior of complex industrial processes, such as the dynamic behavior of a computer system (Simula, Alhoniemi *et al.* 1996). The main problem in process analysis is to find characteristic states or clusters of states that determine the general behavior of the system. In this approach, a hierarchical SOM with two levels was constructed containing a "state map" used to track the operating point of the process with trajectories, and a "dynamics map" used to predict the next state on the state map. Each unit on the dynamics map then represents a "path" leading into the corresponding state. The training set of the state map for the dynamics map is formed by concatenating subsequent vectors into a single vector, representing the history of state transitions. In prediction, the state map vector having the best matching trajectory in its dynamics map is then the predicted state.

An application of hierarchical SOMs in medicine, namely in sleep apnea research, is given in (Guimarães and Urfer 2000). Here, SOMs are used at different hierarchical levels, in order to handle the complexity given by the large number of signal channels. At the lowest level,

primitive patterns in multivariate time series are discovered for distinct time series selections, while more complex patterns are identified at the next higher-level SOM. This approach considers the patterns obtained by the low-level maps, using the information provided by the U-matrix to identify the clusters. Thus, based on this information, a cluster center is calculated, and in order to calculate the input to the next-higher level map, all weights are approximated towards its cluster center c_k according to the following adaptation rule:

$$w_{i,new} = w_i + \mathbf{a} \|w_i - c_k\|, \text{ if } c_k > w_i, \quad (41)$$

and

$$w_{i,new} = w_i - \mathbf{a} \|w_i - c_k\|, \text{ if } c_k \leq w_i. \quad (42)$$

A two-level hierarchical SOM has also been applied to short-term load forecasting (Carpinteiro, Silva *et al.* 2000), and to music data, the Bach's fugue (Carpinteiro 1998). In this approach the input to the second layer SOM is determined by the distance between the best-match $u_i(n)$ and all the other k units of the map $u_j(n), j \neq i$, leading to a k -dimensional input vector.

A hierarchical approach to parameterized SOMs (PSOMs) was proposed by Walter and Ritter (Walter and Ritter 1996), in order to cope with only a very few number of examples. This approach was applied to rapid visuo-motor coordination. One possible solution is to split the learning into two stages, both on distinct PSOMs: 1) a first level PSOM, considered as an investment stage for a pre-structuring of system, which may process a large number of examples; and 2) a second level PSOM, named as Meta-PSOM, that now is a specialized system with fast learning, and only needing a few examples as input. The weights of the first level are then used as input to the second level Meta-PSOM.

4.3.6.4 – A survey of papers

As we shall see in part II of this thesis, we opted for using embedded time in our SOMs, and allowed the user to visually do some trajectory based analysis. So as to have an idea how popular the various temporal SOMs are, we surveyed 68 papers that we considered relevant, and the results are presented in Table 3. Due to the large number of papers involving SOMs for temporal sequence processing, and due to the somewhat fuzzy borders of what are or are not temporal SOMs, many papers that could be considered relevant are not referenced.

In proposing this taxonomy, we focused on identifying the core concepts involved when introducing time into SOMs. As mentioned before, for many applications it is useful to draw on more than one of these ideas. Naturally, our taxonomy is not complete and exhaustive, in the sense that more specific and detailed approaches do exist or can be developed in the future. Also, due to the large number of papers involving SOMs for temporal sequence processing, and due to the somewhat fuzzy borders of what are or are not temporal SOMs, many papers that could be considered relevant are not referenced.

We identified three main approaches for temporal sequence processing with SOMs. These are: 1) methods requiring no modification of the basic SOM algorithm, such as embedded time and trajectory-based approaches; 2) methods that adapt the activation and/or learning algorithm, such as Hypermaps or Kangas Maps; and 3) methods that modify the network structure, introducing feedback connections, or hierarchical levels. The use of each of these approaches, which are not mutually exclusive, depends highly on the application, and none is universally better than any other. The best results are usually obtained by using a carefully tailored combination of these methods. Table 1 provides a classification of some existing and relevant approaches in this taxonomy.

References	Unmodified SOM		Modified activation /learning rule		Modified topology	
	Embedded time	Trajectory-based	Hypermap	Kangas Map	Feedback	Hierarchical
(Alhoniemi <i>et al.</i> , 1999)		X				
(Atlas <i>et al.</i> , 1995)	X					
(Barreto & Araújo, A. 1999)					X	X
(Barreto & Araújo, 2000)					X	X
(Behme <i>et al.</i> , 1993)		X				X
(Brückner <i>et al.</i> , 1992)			X			
(Carpinteiro, 1998)						X
(Carpinteiro & Silva, 2000)						X
(Chandrasekaran & Palaniswami, 1995)		X		X	X	
(Chandrasekaran & Liu, 1998)		X		X	X	
(Chappel & Taylor, 1993)					X	
(Chappelier & Grumbach, 1995)	X					
(Crichley, 1994)					X	
(Euliano & Principe 1996)					X	
(Euliano <i>et al.</i> , 1996)					X	
(Euliano & Principe 1998)					X	
(Euliano & Principe, 1999)					X	
(Guimarães, 2000)	X	X				X
(Guimarães <i>et al.</i> , 2001a)		X				
(James & Miikkulainen, 1995)		X		X	X	
(Jiang <i>et al.</i> , 1994)					X	X
(Jossa <i>et al.</i> , 2001)	X	X				X
(Joutsiniemi <i>et al.</i> , 1995)	X	X				
(Kangas <i>et al.</i> , 1990)	X					
(Kangas, 1992)				X		
(Kangas <i>et al.</i> , 1992)	X	X				
(Kaski & Joutsiniemi, 1993)	X	X				
(Kasslin <i>et al.</i> , 1992)		X				
(Kemke & Wichert, 1993)	X					X
(Kohonen <i>et al.</i> , 1984)	X	X				
(Kohonen, 1988)	X	X				
(Kohonen, 1991)			X			
(Kopecz, 1995)					X	
(Koskela <i>et al.</i> , 1997)					X	
(Koskela <i>et al.</i> , 1998a)					X	
(Koskela <i>et al.</i> , 1998b)					X	
(Lakany, 2001)	X					
(Leinonen <i>et al.</i> , 1992)	X	X				
(Leinonen <i>et al.</i> , 1993)	X	X				
(Lin & Si, 1998)		X				
(Lobo <i>et al.</i> , 1998)	X					
(Midenet & Grumbach, 1994)			X			
(Moshou & Ramon, 2000)	X					
(Mujunen <i>et al.</i> , 1993)	X	X				

	Embd	Traj.	Hyp.	Kang.	Feedb	Hierar
(Pesu et al., 1996)	X					
(Principe & Wang, 1995)					X	
(Principe et al., 2000)					X	
(Ritter et al., 1989)			X			
(Ritter, 1994)			X			
(Ruf <i>et al.</i> , 1998)					X	
(Simula et al., 1996)	X	X				X
(Speidel, 1992)	X					
(Srinivasa & Ahuja, 1999)		X				
(Tryba & Goser, 1991)		X				
(Ultsch, 1993)		X				
(Ultsch et al., 1996)			X			
(Utela et al., 1992)	X	X				
(Varsta et al., 1997)						
(Varsta et al., 2000)					X	
(Vesanto, 1997)			X			
(Vesanto, 1999)						
(Voegtlin, 2000)					X	
(Voegtlin & Dominey, 2001)					X	
(Von Harmelen, 1993)					X	
(Walter & Schulten, 1993)			X			
(Walter & Ritter, 1996)			X			X
(Walter, 1998)			X			
(Zandhuis, 1992)		X				X

Table 3- Overview of the approaches used in 68 different papers.

4.3.7 – Other variants on the basic SOM

Multiple variants of the basic SOM algorithm have been proposed some of which are reviewed in (Kangas, Kohonen *et al.* 1990) and (Kohonen 2001). Besides the Temporal SOMs mentioned in the previous section, these include: non-time related Hierarchical SOMs; Adaptive Sub-Space SOM (ASSOM), where the neurons have a lower dimension than the original pattern, thus “living” in one of its sub-spaces; Self-growing SOMs, that automatically expand when certain criteria are met; Neural Gas, where there are no pre-defined output space neighborhoods, and instead are defined and modified during training; MST-SOMs where the grid neighborhoods are replaced by neighborhoods defined on a Minimum Spanning Tree of the units (Kangas, Kohonen *et al.* 1990); and many more.

Besides the U-Matrices mentioned before, a number of other techniques have been proposed to visualize the output SOM, or to post-process it so that other representations of the data may be provided.

One visualization technique, presented in (Kaski, Venna *et al.* 1999) maps the SOM units directly into a color space, the CIELab space (CIE 1986). This method has two main advantages to the normal display of labeled SOMs. On one hand, it avoids the arbitrary assignment of colors to the labels, in such a way that similar units will be assigned similar colors. This will give a better visual insight into the relationships amongst different units and clusters. On the other hand, this color mapping does not need labeled units, and can thus be applied to pure unsupervised learning problems.

Producing rules from SOMs, as a means of knowledge discovery is becoming an important topic too. One approach followed by (Guimarães and Urfer 2000) uses hierarchical SOMs to achieve higher degrees of abstraction before attempting to generate those rules. Fuzzy rules have also been extracted from SOMs, as for example in (Drobics, Bodenhofer *et al.* 2000), where an algorithm named FF-Miner was developed.

Finally, there have been many hardware implementations of SOM. A number of special built VLSI chips have been designed specifically to implement SOM. Some, such as (Gioiello, Vassallo *et al.* 1992) were projected, simulated, and had their performance theoretically predicted, while others, such as (Rueping 1994) were actually built at foundries. While most of them can boast impressive performances, none has become mainstream. One reason for this is the simple fact that they are custom made for SOM, and thus have little flexibility in being used for other purposes. The small volume of sales makes for a high price, that turns away potential clients. The fact that none of the implementations is clearly better than others, and that all are quite different, makes the learning curve for using them quite steep, again discouraging potential users. Finally, the impressive evolution of general purpose microprocessors makes software implementation a safer bet for investors. It is our opinion that SOM hardware is still searching for a “killer application” to get into the mainstream of computing.

PART I

CHAPTER 5

Classifier design

5.1 – Introduction

The term classification can be used in a variety of contexts, with slightly different meanings. From a computer science and engineering point of view, the term generally refers to data-driven classification, i.e., the ability classify new data, based on previously classified data and, only when possible, on prior knowledge about the problem.

Classification, in the sense that we will consider in this thesis, can be defined as follows:

Given a set X of multidimensional patterns x^1, x^2, \dots, x^n , each one with an associated class q_1, q_2, \dots, q_q , decide which is the class q of a new pattern x .

The patterns may be multidimensional patterns of any type, namely their components may be real-valued, categorical, binary, or anything else. In some instances, they may even be trees or graphs. As for the class, it must be categorical. A real-valued class leads to regression, which is a closely related subject, but one which we shall not address in this thesis. The relationship between the two has been explored in many papers, such as (Torgo and Gama 1997).

As will be discussed in Chapter 6, the set of patterns used to design the classifier is called the training set, referred to as X_{train} . Other patterns, that constitute the validation set, X_{valid} , may be used to control the design process. Finally, some patterns, that are not used for designing the classifier, may be used to estimate the probability of error of the classifier and are called the test set, or X_{test} .

The first classifiers were developed by researchers in the field of statistics and engineering, and followed what we shall call a statistical approach to classifier design. With the emergence of the field of Artificial Intelligence, many new approaches were devised, which we will call AI-based approaches. In recent years there has been a general convergence of these two basically different points of view, for it has been recognized that they have many points in common (Schurmann), and sometimes the same method has been re-invented in one community years after it was developed by the other (Ripley 1996). Statistics can give a sound theoretical foundation for many AI-based methods, and can solve in an optimal and efficient way many problems, while Artificial Intelligence can provide solutions that, if many times not optimal, can solve difficult problems in reasonable time. Comparisons and taxonomies of AI-based and statistical classifiers can be found in (Holmstrom, Koistinen *et al.* 1997), or some of the pattern recognition textbooks referenced.

Following this more modern unifying approach, we will overview the purely statistical classifiers and AI based approaches together, making the necessary distinctions when necessary. I hope that pure statisticians will not be offended by the lack of rigorous mathematical proof for the AI based methods, nor AI researches will be bored with the formalism of statistics. Due to our background, it will necessarily be more of an AI based approach.

5.2 - Classifiers

There are basically two types of statistical approaches to designing classifiers: the parametric approaches, and the non-parametric approaches. Parametric approaches assume that the known patterns have a probability distribution that follows a known analytical function. From the data, the parameters of that function are estimated, and an optimal decision boundary is obtained. Excellent reviews of parametric classifiers can be found in any patterns classification book, of which we may recommend (Duda, Hart *et al.* 2001), (Fukunaga 1990), (Bishop 1995), (Ripley 1996), or (Marques 1999)(which is written in Portuguese).

Non-parametric approaches assume no pre-defined distribution, and try to obtain the decision boundary directly from the data.

This usually implies estimating some measure of the probability density of the data's distribution. To estimate the probability density at a given point, $p(\mathbf{x})$, from the data itself we may take n patterns of the desired class, and find out how many of them (say k) fall within a given area (\mathbf{DV}), and calculate the ratio (Duda, Hart *et al.* 2001):

$$p(\mathbf{x}) \approx \frac{k/n}{\Delta V} \quad (43)$$

This estimate will converge to the true probability density as n increases if three conditions are met:

$$\lim_{n \rightarrow \infty} \Delta V = 0 \quad (44)$$

$$\lim_{n \rightarrow \infty} k = \infty \quad (45)$$

$$\lim_{n \rightarrow \infty} k/n = 0 \quad (46)$$

Clearly the first and second conditions are difficult to meet, even approximately, with any given finite training set. Thus, no data driven probability density estimation will be without error.

The two most used approaches to effectively calculating $p(\mathbf{x})$ require fixing \mathbf{DV} and counting k/n , or letting \mathbf{DV} grow to achieve a desired k . The former technique leads to Parzen Windows based approaches, while the latter leads to k -Nearest Neighbor based approaches, of which the 1-

Nearest Neighbor, or simply Nearest Neighbor is the most common. In this thesis we will only be interested in studying the latter family of non-parametric classifiers. It must be noted that for classification purposes, we do not need to explicitly calculate the probability density of each class in any given point of the input space, but simply find out which class has a higher value of that probability density. Thus, the total number of patterns (n in the above equation), is irrelevant, as is the exact value of DV . If those two parameters are equal for all classes, we need only compute the number k of patterns of each class that fall into some n -dimensional volume DV . If we are only interested in crisp “yes or no” decisions, we are not even interested in knowing the exact k of each class, but only which one is greater.

5.3 - Nearest Neighbor Classifiers

One of most widely used methods for non-parametric classifiers is the nearest neighbor classifier. It is generally recognized that the first serious study of the nearest neighbor rule for classification was done by Fix and Hodges from the US Air Force School of Aviation Medicine, in a technical report dated February 1951, named “Discriminatory analysis, non-parametric discrimination: consistency properties” (Fix and Hodges 1951), available in (Dasarathy 1991). However, the first paper to be published in a major journal, with a sound theoretical justification of the method is due to Cover and Hart in (Cover and Hart 1967).

The nearest neighbor rule for classification can be stated as follows:

Algorithm 4 - Nearest Neighbor Classification Rule

```

Let
     $\mathbf{x}_{Train}$  Be the training Set composed of patterns and associated
                classes  $(\mathbf{x}^1, q^1), (\mathbf{x}^2, q^2), \dots, (\mathbf{x}^n, q^n)$ 
     $\mathbf{x}^{new}$  Be a new pattern
     $q^{new}$  The unknown class of the new pattern

Do
1  For  $i=1$  to  $|\mathbf{x}_{Train}|$  do
2      Calculate the distance  $d^i = || \mathbf{x}^i - \mathbf{x}^{new} ||$ 
3  Find  $i$  that minimizes  $d^i$  (  $i = \text{argmin}(d^i)$  )
4   $q^{new} = q^i$ 

```

Stated in plain English, the rule says: “find the class of the pattern that is nearest to the new pattern, and that will be the new class”.

We can now ask ourselves if this intuitively sound rule does in fact make sense, under which conditions will it perform better or worse than other rules, and how does it compare to Bayes rule, when such can be calculated. As pointed out by many authors (e.g. (Mitchell 1997)), the nearest neighbor classifiers are a particular case of the more general non-parametric probability density estimators, that are at the root of every classification procedure that tries to achieve optimality. As seen in the previous section, to estimate the probability density from data we must compute the values for equation (43), and to achieve the optimum Bayes error, the class we want is the one with greater probability density at that point. Whether or not the nearest neighbor rule converges to this value is known as the convergence problem, which has a number of variants.

The first convergence problems were solved in the 60's when (Cover and Hart 1967) showed that given mild assumptions on the continuity of the probability density function, asymptotically, when the size of the training goes to infinity:

- a) If the classes are separable, the nearest neighbor rule converges to the true class, with probability 1.
- b) If the classes are not separable, the nearest neighbor rule converges to an error rate that is less than twice the optimum Bayes error rate, also with a probability of 1.

Easier to follow, and very elegant proofs for the same problem are given in (Ripley 1996) and (Duda, Hart *et al.* 2001). More research into the asymptotical properties has also been presented in a number of papers, e.g. (Peterson 1970; Gyorfı 1978; Krishna, Thathachar *et al.* 2000).

Unfortunately, these good properties occur when the number of training patterns tends to infinity. Naturally, this is not the case in real applications, and so a lot of work has been done to try and find bounds for the error rate in finite cases. (Cover and Hart 1967) analyzed the 1-dimensional case, which was subsequently broadened to the n -dimensional case by (Gyorfı 1978; Rogers and Wagner 1978; Devroye and Wagner 1979; Fukunaga and Hayes; Psaltis, Snapp *et al.* 1994; Drakopoulos 1995; Bax 2000), and (Nock and Sebban 2001). We recommend (Bax 2000) for a general overview of solutions to the problem. Generally, the best results (Nock and Sebban 2001) show that under very weak assumptions:

$$E_{bayes} \leq E_{neighbour} \leq 2E_{bayes} - \frac{c}{c-1} E_{bayes}^2 + \sup_{x \in X} \delta_{mx}(x) \left(1 - \frac{cE_{bayes}}{c-1}\right), \quad (47)$$

where E_{bayes} is the Bayes error, $E_{neighbours}$ is the error of the nearest neighbor classifier, c is the number of classes, and $\delta_{mx}(x)$ is a likelihood function, originally introduced by (Drakopoulos 1995). As shown in (Nock and Sebban 2001), this likelihood function, that can be estimated for any given finite sample, is usually small.

As a conclusion, the error rates using the nearest neighbor rule with a finite number of training patterns are quite close the optimum Bayes error in theory, and very acceptable in practice, as has been verified in innumerable experimental situations.

The error rate in variants of the nearest neighbor classifiers have also been studied, amongst others, by (Wilson 1972) for edited nearest neighbors, (Kulkarni, Posner *et al.* 1998) for k -nn, and (Krishna, Thathachar *et al.* 2000) for broad family, including LVQ and Nearest Neighbor based Multilayer Perceptrons (NN-MLP) (Zhao and Higuchi 1996).

5.4 – Variations on nearest neighbor or prototype based systems

With the widespread use of powerful computers, and the enormous amount of data available in data warehousing systems, nearest neighbor systems have enjoyed a great deal of attention and found their way into various practical applications. Many improvements have been made on the original algorithm, and sometimes the same technique has been re-invented in different areas of knowledge with different names.

The first point we want to make, is that there are many common points between a vast array of classifier systems that rely on two fundamental principals, that can be used as the definition of **prototype based classifiers**:

- a) When designing the classifier, store instances of patterns x . These patterns are in the same input space as the patterns we will want to classify later. They may be the training

patterns themselves, a selection of them, or new patterns generated in some way. Depending on the approach we take, they may be called reference patterns, reference vectors, prototypes, neurons, stored patterns, examples, cases, etc. In this thesis we choose to adopt the term prototypes as we feel it captures the general idea in a better way, and has been widely adopted (Chang 1974; Bezdek, Reichherzer *et al.* 1998).

- b) When classifying a new pattern, compute the distance (or similarity) to each of the stored patterns, and decide on the new class based on the class of the nearest neighboring prototype or prototypes. Once again, there are small variations on whether we take the actual distance into consideration or not, on whether we consider only the nearest neighbor or a number of nearest neighbors, etc.

Any classifier that uses these two techniques shall be called a prototype based classifier in this thesis. It must be noted that there is no widespread consensus as for the best name for this family of classifiers, and a variety of different names have been used, such as Nearest Prototype Classifiers (NPC) (Kuncheva and Bezdek 1998), Voronoi networks (Vnets) (Krishna, Thathachar *et al.* 2000), Generalized Nearest Prototype Classifiers (Bezdek and Kuncheva 2001), memory based classifiers (Dietterich, Wettschereck *et al.* 1994), etc.

We shall now briefly overview some of the types of classifiers that we consider prototype based classifiers.

5.4.1 - k-means, and fuzzy c-means clustering

Although originally developed as clustering algorithms, the k-means technique and its derivatives, such as fuzzy c-means, have also been used as a way to obtain prototypes for nearest neighbor classifiers (Bishop 1995; Duda, Hart *et al.*). These techniques were reviewed, as clustering techniques, in Chapter 4. When used as classifiers, each centroid is assigned a label, based on the labels of the training patterns that are closest to it. Normally, it is assigned the label of the majority of the patterns, but we may use a more complex scheme, and assign a “probabilistic label”, that estimates the probability of that centroid belonging to any class. When a new pattern is presented, it is assigned the label of its nearest centroid.

5.4.1 - SOM and LVQ

One family of prototype based classifiers stems from the work done on vector quantization, that led to vector quantization-based classifiers, and later to Kohonen's Self-Organizing Maps (SOM) (reviewed in Chapter 4), and Linear Vector Quantization algorithm (LVQ). In this context, the prototypes are named neurons, and are generated by the algorithms based on the original training patterns. In the case of SOM, the prototypes are generated without any knowledge of the classes of the training data, and only after learning has stopped is an "inverse nearest neighbor" rule applied to yield the class of the prototype, which is then called a "labeled neuron". When using a SOM to perform classification, the distance from the new pattern to each neuron is computed, and the class of the nearest neuron (if it has any) is given to that prototype. As noted in various papers, SOMs can be quite effective when a mixture of classification/novelty detection is required, and as prototype generators, they have the advantage of filtering out outliers, and smoothly covering the input space. The LVQ neural networks are more apt for classification, and rely on training algorithms similar to SOMs. However, in LVQ, the classes of the training patterns are taken into account making it a supervised learning algorithm right from the start. If care is not taken, a lot of the smoothness that makes the SOM so useful may be lost when using LVQ. When a LVQ map converges to a stable position, the neurons belonging to different classes are clearly separated, and outliers may not be filtered out.

As with SOM, many LVQ based variants have been proposed, besides the original LVQ1, OLVQ1, LVQ2, and LVQ3 proposed by Kohonen. Two of them are the Generalized Linear Vector Quantization (GLVQ) and its fuzzy version GLVQ-F (Karayiannis, Bezdek *et al.* 1996), which have been used in benchmark comparisons with prototype minimization techniques that we shall overview later in this chapter. Another, that used different update rules and feature weight adaptation, is proposed by (Huang, Chiang *et al.* 2002), and called LVQ-H.

5.4.2 – Neural Gas, Growing Cells, and GTM

Amongst the SOM related variants some must be mentioned explicitly either because they part with the notion of map present in SOM, or have significantly different update rules, or simply because they have diverged quite a bit from the original techniques. One is the Neural Gas approach (Martinetz, Berkovich *et al.* 1993). In this approach, the notion of neighbor in the output space of SOM is substituted by neighborhood in the original input space. Although the update rule is quite similar, the topological mapping present in SOM is lost, and the network

produces just a number of neurons spread out in the input space, in a fashion that resembles the k-means. As a purely sampling technique, the neural gas can have a closer representation of the data than the SOM, especially when the dimensionality of the input space is greater than that of the output space, since there are no distortions imposed by a mapping.

Other are the Growing Cell networks (Fritzke 1991). These networks start with very few units, and add more units as they become necessary. The output plane will not be forced to be a rectangular or hexagonal grid of units, but unlike pure neural gas models, there will be “output space neighborhoods”. More recent developments have unified the Growing Cell and Neural Gas models in the Growing Grid (Fritzke 1995), and Growing Neural Gas (Fritzke 1995) models.

One of the most important alternatives to Kohonen’s SOM is the Generative Topographic Mapping (GTM), proposed in (Bishop, Svensén *et al.* 1996) and (Bishop, Svensen *et al.* 1998), as a statistically well founded alternative to SOM. Each unit in a GTM represents a Gaussian distribution. The parameters of that distribution are determined in a fashion similar to the Gaussian Mixture Model (GMM) (Bishop 1995), using a Expectation-Maximization (EM) algorithm. However, unlike in GMM, constraints are introduced between the units, so that they form a low-dimensional grid that keeps topological neighborhoods.

5.4.3 – RBF

Another neural model called Radial Basis Function Networks, (RBF) was proposed by (Broomhead and Lowe 1988). Once again many variants have been developed, but the main idea remains the same: “center” the neurons in positions that are learned in the input space of patterns, and then assign a certain radial function to each of these neurons. There may be one function for all neurons, or each may adjust its parameters separately. In any case, the centers of the RBF networks clearly correspond to our notion of prototypes.

5.4.4 – CBR

Case Based Reasoning (CBR) has been used in artificial intelligence for many years. The first proposal of CBR as a AI technique is due to (Schank 1982), but it has older philosophical and psychological foundations. The rationale behind CBR is to solve problems by analogy to known solutions or, in other words, to learn by example. A CBR system will store known **cases**, and

when a new problem arises, finds the most similar case. It will then try to adapt the known solution to the new problem. Many improvements have been proposed, and CBR has evolved into a quite mature area, with many good textbooks, such as (Kolodner 1993) for a good overview of early work, (Maher, Balachandran *et al.* 1995) for a industry-oriented perspective, or (Watson 1997) for a more recent and very practical and easy to follow reference. Several well kept internet sites, such as www.cbr-web.org and www.ai-cbr.org are dedicated solely do CBR issues.

Basically CBR systems are a particular case of prototype based systems, for they store the training data (cases), and use similarity between these and new data (new cases), to find solutions for these. However, while most prototype based systems deal with patterns that are simple multidimensional vectors with real-valued, integer, or categorical data, CBR systems will frequently deal with more complex patterns. As an example, (Emam, Benlarbi *et al.* 2001) presents a comparison of various CBR techniques for evaluating the risk associated with software components, represented by their source code and a number of associated indicators. After finding the best match amongst stored cases, CBR systems will sometimes go beyond what a classifier would do (simply find the class), and generate a more elaborate answer. From this point of view, a CBR would be a prototype based classifier followed by a post-processing system.

CBR, with small variants, is also known by many other names such as Exemplar-Based Reasoning (Kilber and Aha 1987), Instance-Based Reasoning (Aha 1991), Memory-Based Reasoning, and Analogy-Based Reasoning, as noted by (Aamodt and Plaza 1994).

5.4.5 – Lazy Learning

Lazy learning is the name given to a number of techniques, most of them reviewed in (Aha 1997). The common factor in these approaches is that little or no processing is done while constructing the classifier with training data. The data patterns are simply stored, and processing is postponed until a new pattern has to be classified. Lazy Learning algorithms are a type of prototype based algorithms because they store all training patterns and all use some type of similarity measure to compare the new patterns with the stored ones. In the preface to (Aha 1997), it is recognized that Lazy Learning is one more name for a broad family that includes the CBR systems mentioned in the previous section, and many other nearest neighbor based techniques. The editor of the book is personally responsible for quite a few different names, but

argues that each name focuses on a particular aspect, thus creating sub-families that put different emphasis on different characteristics of prototype based classifiers.

5.4.6 –SVM (Support Vector Machines) and other Kernel Based classifiers

In recent years there has been a lot of interest for Support Vector Machines (SVM), which are a particular case of a more general family named Kernel Based classifiers (Herbrich 2001), that include the above mentioned RBF networks. SVM stem from the theoretical work of Vapnik on learning theory and risk minimization, presented originally in (Boser, Guyon *et al.* 1992), and edited as a book in (Vapnik 2000). An enormous amount of papers and books that have been published on the subject, and quite a few software packages implement SVM, both for research and for commercial purposes. For an overview of SVM we would recommend (Cristianini and Shawe-Taylor 2001). For a complete yet easy to follow description of theoretical aspects of kernel machines and learning theory, we would recommend (Anthony and Bartlett 1999), while a more practical overview of the same subjects is presented in (Herbrich 2001). A short and easy to follow tutorial on the use of SVM is available in (Burges 1998). There are also a few very well kept internet sites on the subject, such as “kernel-machines.org” or “svm.research.bell-labs.com”.

The basic idea behind SVM is that it is always possible to transform the data into a space where the classes are linearly separable (Bishop 1995), which will have a dimensionality at least equal to the data’s Vapnik-Chervonenkis (VC) dimension (Anthony and Bartlett 1999). In that space, a SVM will find the data patterns that are closest to the border between the classes. These patterns are called the Support Vectors, since they are the support for choosing the optimal hyperplane that separates the cases. A SVM will then choose the hyperplane that is equidistant from the patterns of different classes. Clearly the support vectors chosen correspond to our notion of classifier prototypes, and can be used as such.

Other Kernel Based classifier also rely on finding some sort of function, called kernel function, that will be localized somewhere in a given feature space. There is a very wide variety of possible Kernel functions (although they must satisfy Mercer’s theorem (Herbrich 2001)), including polynomials and RBFs. If we consider appropriate similarity (or distance) functions, the centers of these kernel functions can be seen as prototypes for nearest neighbor classification.

5.5 – Other research on nearest neighbor related problems

5.5.1 - k -Nearest Neighbors, and voting schemes

Slightly better accuracies are possible using a variety of k -nearest neighbor schemes (Bishop 1995) which have been studied together with the basic nearest neighbor rule since (Wilson 1972). The Nearest Neighbor classifier can be seen as a particular case of these k -Nearest Neighbor schemes, with $k=1$. Nonetheless, we will not overview them in this thesis. One reason is that they require fine-tuning of a certain number of parameters, such as the number k of neighbors to consider, or the method to assign weight ϵ to the neighbors (Devijver and Kittler 1982). The main reason however is that they do not lend themselves easily to prototype minimization, which is our main interest.

5.5.2 - Influence of distance or similarity measures

The original papers on nearest neighbors, and indeed most of all the work done on nearest neighbors, use patterns in R^n , and use the Euclidean distance to find the nearest neighbors. Although widely used, this measure has several drawbacks, such as its inability to deal with non-numerical attributes, and its sensitivity to irrelevant attributes. This has led to a great deal of research into the use of other measures of distance or similarity, and their influence in the behavior of classifiers.

For lists and descriptions of different similarity measures that have been used in nearest neighbor classifiers, we would recommend annex A of (Webb 1999), the second chapter of (Devroye, Györfi *et al.* 1996), or the introductory chapter of (Kohonen 2001).

The reason why nearest neighbor classifiers are so sensitive to the similarity measure, is that different measures may lead to different neighbors, and indeed to a very different topological ordering. The choice of similarity measure is thus critically dependant on the specific problem at hand, and many similarity or distance functions have been used. A contribution for the choice of the optimal metric for nearest neighbor classification, under certain constraints, is presented in (Short and Fukunaga 1980) (Short and Fukunaga 1981).

The similarity measure does not have to be unique, and different (or local) measures may be used depending on the patterns being considered. This type of approach is used by (Hastie and Tibshirani 1996; Wettschereck, Mohri *et al.* 1997; Wilson and Martinez 1997; Ricci and Avesani 1999).

5.5.3 - Fast search for nearest neighbors

With the growing size of databases and available data for training prototype based classifiers, the problem of finding the nearest neighbor within these very large sets of prototypes has become a subject of great practical interest. Most techniques rely on efficient database organization, sometimes dividing the input space into sub-regions where fewer prototypes have to be searched, using hierarchical proximity graphs, or using “approximate nearest neighbor” techniques. Some of these techniques produce nearest neighbor classification systems that have a structure similar to those of the prototype minimization techniques that we shall see in the next section. One such method is the Reduced Complexity Nearest Neighbors (RCNN) (Lee and Chae 1998), that separates the prototypes into *anchors* and *non-anchors* in a fashion that resembles the search for small prototype sets described later. Of the many papers proposing efficient ways to store and look for data, we may suggest (Ramasubramanian and Paliwal 1992; Tai, Lai *et al.* 1996; Song and Ra 2002).

5.6 - Prototype minimization

Despite its simplicity, soundness, and ease of use, the nearest neighbor classifier has a few major drawbacks:

- a) Large memory requirements. All the training set must be stored in memory.
- b) Heavy processing requirements. For every new pattern that is to be classified, the distance to all stored patterns has to be calculated. Since there are many patterns, this will take a lot a time.
- c) Sensitivity to noise, outliers, and overlapping distributions.

As we saw in the previous section, the latter two drawbacks are addressed, albeit without really good and efficient solutions, with fast searching techniques and k-nearest neighbor rules. However, both drawbacks would be significantly minimized if we could reduce the number of patterns in the set used for classification, and do so in an “intelligent” manner. This new, smaller

subset of patterns will be from now on called *classification set*, or *set of prototypes*, since these patterns are representatives of the class they try to classify.

One of the co-authors of first major paper on Nearest Neighbors (Cover and Hart 1967), presented a first attempt at obtaining a smaller classification set in (Hart 1968), calling it Condensed Nearest Neighbors (CNN). Since Hart's original paper in 1968, several proposals have been made to obtain a smaller amount of prototypes than the whole training set, namely Reduced Nearest Neighbors (RNN) (Gates 1972), Edited Nearest Neighbors (Wilson 1972), Iterative Condensation Algorithm (ICA) (Swonger 1972), Multiedited Nearest Neighbors (Devijver and Kittler 1982), Spanning Tree based nearest neighbors, or Chang Algorithm (Chang 1974), Selective Nearest Neighbors (SNN) (Ritter, Woodruff *et al.* 1975), Ordered CNN (Tomek 1976), (Tomek 1976), (Gowda and Krishna 1979), Symbolic Condensed Nearest Neighbors (Gowda and Ravi 1994), Dasarathys Minimum Consistent Subset (Dasarathy 1994), Proximity Graph (PG) editing (Dasarathy and Sanchez 2000), DYNAGEN (Laha and Pal 2001), Tabu Search generated nearest Neighbors (Zhang and Sun 2002), and many others that we will mention later, such as (Ullmann 1974; Bezdek, Reichherzer *et al.* 1998; Ferri, Albert *et al.* 1999).

These different approaches can broadly be classified into editing techniques, when the main goal is to reduce errors by omitting certain patterns, and condensing techniques, when the sole objective is to reduce the number of patterns.

Over time, various reviews and comparisons have been made of these different condensing, editing, selection, or generating techniques. We must mention a few major ones, namely the book (Dasarathy 1991) that reviews all the early work, and contains copies of the original articles, and (Wilson and Martinez 1997), that contains a brief but very good review of more recent work. For a more up to date review this thesis is hopefully a good reference, and we intend to present a short paper with the key concepts shortly.

The first question that arises is whether there is an “optimal” *classification set*, in the sense that it has the minimum number of prototypes necessary for the classification of a given training or test set. This is a sensitive question, since this optimal classification set for one set of test patterns is not necessarily the optimum set for another set of test patterns. Worse still, what really matters is the optimal classification set for the unseen data patterns, which obviously cannot be computed. Another problem arises as to whether that “optimal classification set” must be selected from an

available set of prototypes, or whether new prototypes may be generated at the “optimal” locations. To clarify concepts, we shall define some concepts before overviewing prototype minimization techniques.

5.6.1 - Consistent Subset

The concept of consistent subset was introduced by (Hart 1968), and can be defined as follows:

Let X be a set of patterns. The set of patterns $C \subset X$ is said to be a consistent subset of X if and only if for every pattern $\mathbf{x} \in X$, the closest pattern to it in C has the same class.

A consistent subset is said to be minimal² if its cardinality is less or equal to any other consistent subset. It was proved (Wilfong 1991) that finding a minimal consistent subset for patterns in \mathbb{R}^2 is equivalent to the disc covering problem, and thus NP-complete. Although a general case proof has not been produced, it is reasonable to extend the concepts used in (Wilfong 1991) and believe that save for very particular cases, the search for a minimal consistent subset is always NP-complete. This probably explains why so many different techniques have been developed for finding it, and why none is truly optimal and practical at the same time.

A minimal consistent subset of prototypes is what is sought when we attempt to find an “optimal” classification set by selecting available prototypes. However, this minimal consistent subset may produce decision boundaries that are quite far from the original sets boundary, and thus another concept was developed, that is closer to these boundaries.

² Some authors, like Wilfong (1991). Nearest Neighbor Problems. 7th ACM Symposium on Computational Geometry. use the term minimum, while others such as Ritter, G. L., H. B. Woodruff, S. R. Lowry and T. L. Isenhour (1975). "An Algorithm for a Selective Nearest Neighbor Decision Rule." IEEE Transactions on Information Theory: 665-669., Dasarathy, B. V. (1994). "Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design." IEEE Transactions on Systems, Man, and Cybernetics **24**(3): 511-517. use minimal. We choose to use the latter.

5.6.2 - Selective Subset

The concept of selective subset was introduced by (Ritter, Woodruff *et al.* 1975), and with a few modifications can be defined as follows:

Let X be a set of patterns. The set of patterns $S \subset X$ is said to be a consistent subset of X if and only if for every pattern $x \in X$, the closest pattern to it in S has the same class, and is closer than any pattern $x \in X$ that has a different class.

A selective subset is said to be minimal, if its cardinality is less or equal to any other selective subset. The difference from the definition of consistent subset may seem subtle, but is crucial. In a selective subset, we require that the prototypes of each class not only classify correctly all patterns when we use the all prototypes for classification, but that they still classify correctly all patterns, when all patterns of the other classes are used as prototypes. Thus, each class has to choose its prototypes assuming that the other classes will retain all their patterns as prototypes. It is a “keep every inch” approach, that leads to final interclass boundaries very close to the original nearest neighbor boundaries. Given the well-known properties of these boundaries, it can be argued that the minimal selective subset, though having more prototypes than the minimal consistent subset, will yield a better classifier. Once again, the question of the applicability of Occam’s razor to classification can be raised (Natarajan), but we leave that discussion for (Domingos 1999).

5.6.3 – A taxonomy of prototype minimization techniques

Since there are so many prototype minimization techniques, and they vary so much amongst themselves, we shall attempt to classify them according to their characteristics, before going into the details. We choose to do that classification according to the following parameters.

5.6.3.1 - Consistency (**Consistent, Selective, none**)

Some approaches generate only consistent subsets, others selective subsets, and others attempt neither. When using this categorization, we relax the requirement that the classifier prototypes be a proper subset of the original set of prototypes, and consider that it may be a subset of a

hypothetical superset of prototypes. We do this to allow algorithms that generate their own prototypes to be considered consistent, if they have the consistency properties. Techniques that are neither consistent nor selective, generally admit errors in the training set. Selective subsets are always a particular case of Consistent subsets, so if a method produces Selective subsets, we will not include it the consistent methods.

5.6.3.2 - *Selection/Generation* (**Select, Generate**)

Some approaches only select prototypes from the initial available ones, while others will generate new prototypes at more convenient locations. In some applications, it may not make sense to do so, since these would lead to prototypes at locations that, for some reason, do not make sense (for example, families with 2.3 children or cars with 4.8 wheels). It may also be difficult to generate new prototypes when the patterns are not real valued, such as when they are trees, complex data structures, or probability distributions.

5.6.3.3 - *Determinism* (**Deterministic, Order dependent, Parameter dependant, Random dependent**)

Some approaches will, given the same prototypes and patterns, generate always the same classifier, and are thus dubbed deterministic. Those that are not deterministic, may have a number of different factor that affect the outcome, and these factors may occur simultaneously. Some approaches will depend on the order by which the patterns or prototypes are presented to it. Others rely on one or more parameters that are user-definable, such as maximum allowed error, or the choice of a kernel function. Others still rely on random variables to search for the best solutions, as is the case with genetic algorithms or simulated annealing.

We shall now review the main methods, and in the process present the theory developed by each.

5.6.4 – **Prototype Minimization techniques**

Over the years many different techniques have been developed to minimize the number of prototypes necessary for classification. We shall now attempt to review them.

5.6.4.1 – CNN - Condensed Nearest Neighbors

The first attempt to minimize the number of prototypes, named Condensed Nearest Neighbors (CNN) was proposed by Hart in (Hart 1968). CNN has become a benchmark against which most other algorithms are compared. Formally the algorithm can be described as follows:

Algorithm 5 - Building Condensed Nearest Neighbors set (CNN)

```

Given

   $\mathbf{x}_{Train}$       Training set, with patterns  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ 
   $|\mathbf{x}_{Train}|$       Number of patterns in the training set
  CNN              Condensed Nearest Neighbor set
  Additions      A Boolean flag

Do

1      CNN =  $\{\mathbf{x}_1\}$ 
2      Repeat
3          Additions=FALSE
4          For i =2 to  $|\mathbf{x}_{Train}|$ 
5              Classify  $\mathbf{x}_i$  with CNN
6              If  $\mathbf{x}_i$  is incorrectly classified
7                  CNN = CNN  $\cup$   $\{\mathbf{x}_i\}$ 
8                  Additions=TRUE
9      Until Additions = FALSE

```

This algorithm guarantees that all patterns in the training set will have the same classification with CNN and with the original classification set, and that the new set will not be larger than the original one. In practice, the classification set thus obtained, which we shall call *CNN*, is much smaller than the original set.

While simple and reasonably efficient, this algorithm is far from optimal, and has a number of shortcomings.

The first concerns minimality. The final classification set, *CNN*, depends on the order by which the patterns are presented. Besides being annoying for many applications, this fact by itself shows that CNN will not find an absolute minimal classification set. In many cases the first prototypes to be added to *CNN* will later be made redundant, since prototypes closer to the border between the classes will inevitably be selected. The sensitivity to the order by which the patterns are presented may be partially overcome by re-initializing CNN with different permutations of the training set, as done in (Cerverón and Ferri 2001).

The second concerns robustness to noise. Since all patterns in the training set will be classified exactly as they were in the original classification set, any outliers will be retained. Since the *CNN* will have fewer prototypes, those that remain will have greater importance, because we can no longer use the *knn* algorithm to smooth out the outliers. Thus, the *CNN* method works best when the classes are separable. The issue of separability is discussed in (Cover 1965) and (Haykin 1999). If the classes are not separable, it is advisable to use some editing algorithm (see 5.6.4.3 - *ENN* - Edited Nearest Neighbors) before *CNN*, since editing will “clean up” the overlap area. This general principle is applicable to many of the prototype minimization techniques that we will overview.

To show the effectiveness of the *CNN* algorithm, a toy problem is proposed in (Hart 1968), that, with only minor modifications, has been used as benchmark and visualization example for many other authors (such as (Ritter, Woodruff *et al.* 1975), (Gowda and Krishna 1979), (Tomek 1976)). In this problem, which we shall call Hart’s problem, we have 2 classes with uniform distribution in the areas shown in Figure 23. The two classes form “F” shapes (one of them inverted), one with boundaries defined by the line that joins (0,0), (7.5,0), (7.5,5), (15,5), (15,10), (7.5,10), (7.5,15), (15,15), (15,20),(0,20), and the other by its complement in the rectangle limited by (0,0),(22.5,20). Each class has 200 patterns used for training, and another 200 used for validation.

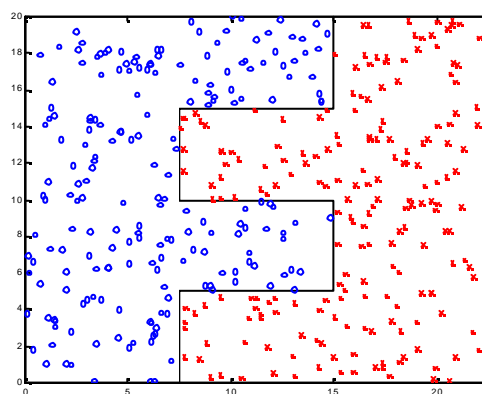


Figure 23 – Hart’s problem: two classes, each with 200 patterns, with uniform distribution in the “F” shapes given

In the next section, a comparative study is done on the performance of the different prototype minimization techniques, but just as an example, in a we show the results of applying *CNN* to

Harts problem. In this run, only 47 of the original 400 patterns were selected as classifiers (21 patterns for class 1, 26 for class 2).

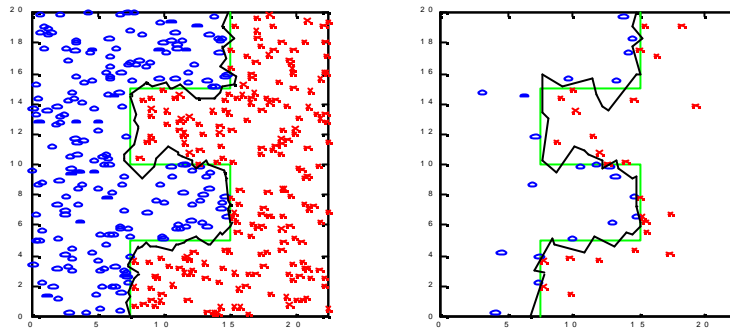


Figure 24 - Comparison of NN and CNN for Hart's problem. In the right figure, only 47 of the original 400 patterns were selected as classifiers.

More recent studies on the properties and complexity of the original CNN procedure have been made by (Baram 2000), and shown that in the general case, the complexity is $O(n^3)$, where n is the size of the original set, and the expected

5.6.4.2 – RNN - Reduced Nearest Neighbors

One of the main reasons why the CNN will not yield a (at least local) minimal number of prototypes is that, in the first steps of the algorithm, prototypes are included that will later be made redundant by new additions. An obvious solution was proposed by (Gates 1972), named the Reduced Nearest Neighbors. This method of obtaining a classification set uses as a starting point the *CNN*, and then prunes it. It can be stated as follows:

Algorithm 6 - Building the Reduced Nearest Neighbor set (RNN)

Given

\mathbf{X}_{Train} Training set, with patterns $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$
 CNN Condensed Nearest Neighbor set
 $|CNN|$ Number of patterns in CNN
 RNN Reduced Nearest Neighbor Set, with prototypes $rnn_1, rnn_2, \dots, rnn_n$
 $Candidate_RNN$ A set of prototypes

Do

```

1    $RNN = CNN$ 
2   For  $i = 1$  to  $|CNN|$ 
3     Let  $Candidate\_RNN = RNN - \{rnn_i\}$ 
4     Classify all  $\mathbf{X}_{Train}$  with  $Candidate\_RNN$ 
5     If all patterns in  $\mathbf{X}_{Train}$  are correctly classified
6        $RNN = Candidate\_RNN$ 
  
```

As we did for CNN, in Figure 25 we present an example of the use of RNN on Hart's problem (exactly on the same data used for CNN). In this case, the number of patterns used for classification dropped to only 29 (14 for class 1, 15 for class 2).

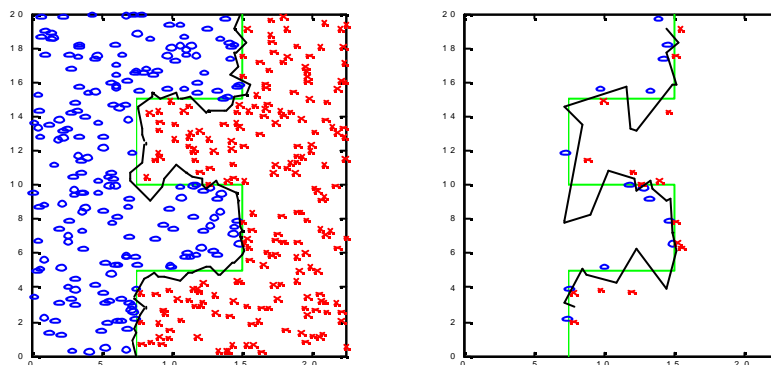


Figure 25 - Comparison of NN and RNN for Hart's problem. In the right figure, only 29 of the original 400 pattern were selected as classifiers.

The RNN will select the Minimum consistent subset (MCS) of the available prototypes if and only if the MCS is contained in the CNN, as proved in (Gates 1972). This last condition is generally not met, and thus the set of prototypes selected, although small, might not be the MCS.

5.6.4.3 - ENN - Edited Nearest Neighbors

As mentioned earlier, a k -nearest neighbor rule, with $k > 1$, can give slightly lower errors than the simple nearest neighbor rule. One of the reasons, is that the k -nearest neighbor rule will filter out

the outliers. With this in mind, (Wilson 1972) proposed the use of what he called an editing technique, that used the 3-nearest neighbor rule to classify each of the prototypes. If the classification thus provided was incorrect, that prototype was dropped. The final set of prototypes will therefore be smaller, so as a byproduct, this technique is also a prototype minimization technique. The original paper has a very thorough theoretical study of the asymptotical properties of the obtained classifier, showing that it is very close to Bayes optimal classifier.

A number of improvements were rapidly developed for the basic Edited Nearest Neighbor technique (Tomek 1976), and these are rather well summarized in (Devijver and Kittler 1982). The first logical step is to apply condensing techniques to this technique so as to further decrease the number of prototypes. The obtained classifier does in fact have smoother borders than those produced by the classical condensing techniques. Another improvement that was attempted was to use different values of k for the editing phase, using weighted voting schemes, (k, ϵ) -nearest neighbors. Finally, the editing procedures can be iterated, giving rise to the so-called multi-edit techniques (Devijver and Kittler 1982).

The same basic editing ideas have also been used with different neighborhood measures by (Dasarathy and Sanchez 2000), and were called Proximity Graph (PG) editing. In this approach, Euclidean distances were substituted by neighborhoods in Gabriel Graphs (GG) and Relative Neighborhood Graphs (RNG).

Another approach, that while not citing edited nearest neighbors explicitly uses the same principle, is (De and Pal 2001), where fuzzy neighborhoods are used to select the prototypes.

5.6.4.4 - ICA - Iterative Condensation Algorithm

This approach, proposed in (Swonger 1972), tries to improve the original CNN by allowing some of the prototypes to be discarded. Unlike RNN, it will discard not only those that are not necessary for a correct classification, but also those that are responsible for more misclassifications than correct classifications. By allowing the error to be greater than 0 (thus not producing a consistent subset), the ICA is more tolerant to outliers, and achieves better results when the classes are not separable.

5.6.4.5 - Chang – Chang's Algorithm

The Chang algorithm was originally proposed by (Chang 1974), and later adapted to batch processing (Yen and Chang 1994), and a modified version was used by (Bezdek, Reichherzer *et al.* 1998). Chang's algorithm main idea is to generate new prototypes, by merging two existing prototypes into only one, located at their weighted mid-point. The merging process stops when the classification error starts to rise. This algorithm can be described by:

Algorithm 7 – Chang's Algorithm

```

Given

XTrain Training set, with patterns x1, x2, ..., xn
|XTrain| Number of patterns in the training set
Chang Chang's set of prototypes chang1, chang2, ...
Changwi Weight associated with prototype changi in Chang
Error Real valued variable, to store the error rate

Do

1   Chang = XTrain
2   changw = 1 for all prototypes
3   Repeat
4     Find the two nearest neighbors changi, changj in Chang
5     Remove changi, changj from Chang
6     Create changk at the weighed mid point between changi,
changj, so that changk=
(changwi*changi+changwj*changj) / (changwi+changwj)
7     changwk=changwi+changwj
8     Classify XTrain with Chang, and calculate the error rate
Error
9     If Error is greater then the desired error, remove changk
from Chang, re-insert Changi and Changj and terminate the
merging
10    Until Error exceeds the desired error rate

```

This algorithm was inspired by the Minimum Spanning Tree algorithm (MST) (Baase and Gelder 2000), and thus techniques developed for MST can easily be applied to Chang's algorithm.

The modifications proposed by (Bezdek, Reichherzer *et al.* 1998) do not use Chang's weights, apply the merging processes locally (by dividing the input space into regions), allow merging of more than two prototypes simultaneously, use the distance between prototypes as weights for merging, and when the two nearest neighbors cannot be merged, the modified version will attempt to merge the next best matches.

While good classification results have been obtained with Chang algorithm classifiers, it must be pointed out that, as they generate new prototypes, the distances from prototypes to training patterns has to be computed many times, thus imposing a heavy computational burden.

5.6.4.6 - SNN - Selective Nearest Neighbors

The Selective Nearest Neighbors (SNN) was originally proposed by (Ritter, Woodruff *et al.* 1975). It was one of the first papers that aims at optimality, and its approach is quite similar to that of (Dasarathy 1994) and to the new approach presented in part II of this thesis. The main concern of (Ritter, Woodruff *et al.* 1975) is that a consistent subset, such as those produced by the CNN rule, may lead to interclass borders that are quite far from the original nearest neighbor borders, that we know are quite close to the optimum Bayes decision boundaries. This happens because prototypes close to these borders can easily be deleted by the condensing techniques, and almost certainly the minimum consistent subset will not contain most of these prototypes. The paper then introduces the concept of selective subset discussed earlier, arguing that it will be closer to the original borders. The problem is then to find the minimal selective subset.

A key concept of this paper is that of *selective neighbor* of a pattern i , denoted Y_i . A prototype is a selective neighbor of a pattern if it has the same class as the pattern, and is closest to it than any pattern of a different class³. A selective subset of the original prototypes must include at least one selective neighbor of each pattern. A binary matrix A is then constructed, where each column i corresponds to a pattern, and each row j to a prototype, such that

$$A_{ij} = \begin{cases} 1 & \text{if } p_j \in Y_i \\ 0 & \text{if } p_j \notin Y_i \end{cases} \quad (48)$$

It then starts the SNN algorithm, that is described as follows:

³ The selective neighbors of a pattern, in Ritters sense, are equivalent to the Q-set of a pattern in the positive only approach described in part II of this thesis, or to the $Q_0(x)$, since in Ritters framework $R_0(x)$ will always be empty.

Algorithm 8 – Selective Nearest Neighbors

```

Given

A           Ritter's binary matrix
P           Set of all prototypes (  $p_1, p_2, \dots, p_n$  )
SS          Selective subset of prototypes, initialized to  $\emptyset$ 

 $|X_{Train}|$       Number of patterns in the training set
CNN         Condensed Nearest Neighbor set
Additions     A Boolean flag

Do

1           For all  $i$  corresponding to columns remaining in A
2             If column  $i$  of A has only a single 1, then:
3               Store the index of the row  $j$  where that 1 occurs
4               SS = SS +  $\{p_j\}$ 
5               Delete all columns of A where row  $j$  has the value 1
6           For all  $j$  corresponding to rows remaining in A
7             For all  $k \neq j$  corresponding to rows remaining in A
8               If for all  $i$   $A_{ji} \leq A_{ki}$  , delete row  $j$ 
9           For all  $i$  corresponding to columns remaining in A
10            For  $k \neq i$  corresponding to columns remaining in A
11              If for all  $j$   $A_{ji} \leq A_{jk}$  , delete column  $i$ 
12           If A is empty, terminate the algorithm
13           If any deletions were made in steps 1 to 8 go back to step 1
14           Use a branch and bound algorithm to select the minimum number
              of prototypes, corresponding to the rows, that will guarantee
              that each column has at least one 1.

```

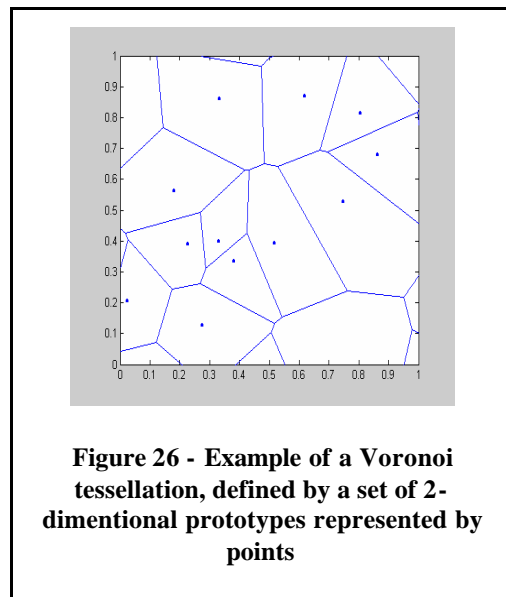
Steps 1 to 11 of Algorithm 8 will select the obvious choices of selective nearest neighbors, and will prune Ritter's matrix, while the computationally hard choices are left to step 14. Step 1 to 5 will choose the prototypes that are the only selective neighbor of any given pattern. These steps are exactly the same as the first steps of the positive-only Q -set heuristic presented in Chapter 1 of Part II. Steps 6 to 8 will prune Ritter's matrix by eliminating prototypes that are selective neighbors of only a subset of the patterns "covered" by another prototype. These prototypes should not be chosen as selective nearest neighbors, since there is another prototype that classifies correctly all the patterns that they do, and still some more. Steps 9 to 11 will prune Ritter's matrix by eliminating patterns that will be correctly classified if another pattern is correctly classified, i.e., that are particular cases of a more difficult classification. These pruning steps will greatly reduce the computational effort of search performed in step 14.

The search procedure of step 14, for which a specific algorithm is proposed in (Ritter, Woodruff *et al.* 1975), is computationally very hard, effectively limiting the use of SNN to simple problems.

It must be noted however, that the results produced by the SNN procedure are identical to those produced by the *positive-only Q-sets with optimal selection*, presented later in part II of this thesis.

5.6.4.7 – Voronoi - Voronoi boundary nearest neighbors

Most of the data condensing techniques change the actual boundary between the classes, even if they do keep exactly the same error rate. When considering 2-dimensional patterns, (Toussaint and Poulsen 1979) developed a technique based on Voronoi tessellation⁴, based on the earlier work by (Dasarathy and White 1978). The Voronoi tessellation is a partition of space into disjoint regions around a number of reference points, in such a way that any point in the region around a reference point is closer to it than to any other reference point, as seen in Figure 23. In simpler words, a Voronoi tessellation determines the “area of



influence” of a reference point. All prototype based classifiers are implicitly defining a Voronoi tessellation, and the borders between classes are the edges of the tessellation that lie between two reference points (in this case prototypes) that belong to different classes. If we keep only the prototypes whose edges are edges of prototypes with a different class, we will have exactly the same borders, and hopefully less prototypes. One of the fastest ways of computing the Voronoi

⁴ As pointed out by Halls, P. J., M. Bulling, P. C. L. White, L. Garland and S. Harris (2001). "Dirichlet neighbours: revisiting Dirichlet tessellation for neighbourhood analysis." Computers, Environment and Urban Systems **25**(1): 105-117., the Voronoi tessellation is really due to Dirichlet who developed it for the 2-dimensional case. Voronoi later extended the concept to the n-dimensional case, and Thiessen further improved it for practical applications. It is thus alternatively known as Dirichlet tessellation, or Thiessen tessellation

edges, is to compute first the Delaunay triangulations (Mathworks 2001). The Delaunay triangulation algorithms produce a list of sets of 3 points (patterns) that define the triangles, and a simple inspection of the classes of these sets will determine which points (patterns) to include: if a triangle has more than 1 class in its 3 vertices, all 3 vertices need to be included. Although we have not seen this explicitly mentioned in any paper, we use this principle in one of our Matlab routines presented in part III of the thesis.

For high dimensional dataset, a Voronoi will be extremely difficult to compute, and we do not know of any procedure that is easily extendable to a n -dimensional case. Algorithms for computing it in R^2 are $O(n \log n)$, where n is the number of points (Baram 2000), and for R^3 they are $O(n^2 \log n)$. For higher dimensions, we do not know of any results.

Although not keeping the exact Voronoi boundary, recent papers have used Voronoi boundaries do extract small sets of prototypes, such as (Baram 2000), and same principle is used in the neural network community, leading to the Voronoi-diagram based Neural Networks (Gentile and Sznaiier 2001).

5.6.4.8 - MNV – Mutual Neighborhood Value

The Mutual Neighborhood Value (MNV) algorithm was originally proposed by Gowda (Gowda and Krishna 1979) using work done for his PhD thesis in 1978 (we were not able to find a copy of this thesis).

A modification of the original MNV is proposed by (Gowda and Ravi 1994), in which the actual values of each pattern are substituted by *symbolic* values, implicitly performing a varying quantization of those values, and a new *symbolic similarity measure* is used. This new approach is sometimes called Symbolic Nearest Neighbors.

5.6.4.9 – RPC - Reduced Parzen Classifier

The Reduced Parzen Classifier was initially proposed by (Fukunaga and Hayes 1989). The basic idea is somewhat similar to the Reduced Nearest Neighbor (RNN), in that it tries to improve a classifier by tentatively eliminating each of its prototypes sequentially. Unlike RNN, the criteria used for keeping the prototypes is not the classification error, but the change in density estimate

using Parzen windows. The RPC also allows the introduction of new prototypes from the training set, provided there is a significant improvement in the density estimation, as compared to that that is possible using the full training set. Unfortunately, this technique is computationally very demanding.

5.6.4.10 – IBL, IB2, IB3, TIBL, BIBL – Instance Based Learning

In (Aha, Kibler *et al.* 1991), a number of optimizations for Instance Based Learning (see 5.4.4) are proposed. Of these, the IB2 and IB3 became the most popular, and have been the subject of further improvements. The original IB2 is just a re-invention of CNN (see 5.6.4.1), but IB3 introduces additional heuristics that make it more effective in certain circumstances. The IB algorithms became important in the Instance Based learning community, and are frequently used as benchmark references, e.g. (Brighton and Mellish 2002), (Zarndt).

Typical Instance Based Learning (TIBL), proposed in (Zhang 2002) tries to use the centermost prototypes first, considering them more typical than the border prototypes. The measure of “typicalness” takes into account the similarity with other prototypes of the same class and the distance from prototypes from different classes. The same author also tries to use the exact opposite of TIBL, that tries to select the less typical, or boundary instances first. This latter approach is called Boundary Instance Based Learning (BIBL). In the experiments proposed by (Zhang 2002), TIBL is compared with BIBL, pure nearest neighbors (there called Instance Based Learning – IBL), and a variation of IB2 (there called Storage Reduction Based Learning – SRBL). As expected, the results varied widely from dataset to dataset. TIBL would sometimes be outperformed in accuracy, although never by much, and it would always yield far fewer prototypes than the other methods.

5.6.4.11 - NGE – Nearest Generalized Exemplars

Nearest Generalized Exemplars (NGE) were introduced by (Salzberg 1991). The basic idea is similar to Parzen windows and RBF, since it tries to construct ever larger hyper-rectangles around the selected prototypes. This approach has proved to be very popular. An implementation of NGE is supplied in (Aha 1995), and comparisons with other methods are available in (Wettschereck and Dietterich 1995), that also proposes changes to the basic NGE.

5.6.4.12 – DMCS - Dasarathys Minimum Consistent Subset

Dasarathys Minimum Consistent Subset (DMCS) procedure was proposed in (Dasarathy 1994), originally claiming to find the minimum consistent dataset. It bears some resemblance to the SNN algorithm (Ritter, Woodruff *et al.* 1975), but it does not attempt to build a selective subset, and reconstructs the neighborhood of each pattern each time a prototype is excluded. On the other hand, it also resembles ICA (Swonger 1972) in that it iterates successive consistent subsets, re-evaluating them against the original patterns, and trying to improve them. Dasarathy introduces the concept of nearest unlike neighbor (NUN) of a pattern, that is the closest prototype to it that has a different class. It is argued that this NUN is critical for defining how much simplification can occur for the class of the pattern in question. The prototypes of the same class that are nearer than the NUN form Ritters selective neighbor set. The patterns then cast a vote for all the prototypes in their selective neighbor set, and the most voted prototype is selected for the next generation consistent subset. The patterns that voted for this prototype are then removed, and a new vote is performed. This process is repeated until no more patterns remain. Thus far, the MCS is a heuristic approach to the SNN optimal method, more or less equivalent to the positive-only heuristic presented in part II of this thesis, and it produced a selective subset of the prototypes. However, the process is now iterated, using only the prototypes obtained in the last iteration, and inserting any of the original prototypes that do not increase the number of errors. This new iterative process will hopefully calculate in each iteration NUNs that are further away than the previous, since the prototypes close the boarder tend not to be chosen.

Unfortunately, the claim that the iterative process will converge to a minimal consistent subset is not well founded and a counter-example to this claim has been found, for example, in (Kuncheva and Bezdek 1998)⁵, (Cerverón and Fuertes 1998), and (Zhang and Sun 2002). This last paper also provides an explanation why the process is not optimal.

⁵ Interestingly, the Iris datasets used by Dasarathy, B. V. (1994). "Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design." IEEE Transactions on Systems, Man, and Cybernetics **24**(3): 511-517. and Kuncheva, L. I. and J. C. Bezdek (1998). "Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search ?" IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **28**(1): 160 - 164. are not exactly the same, as is discussed in Bezdek, J. C., J. M. Keller, R. Krishnapuram, L. I. Kuncheva and N. R. Pal (1999). "Will the real iris data please stand up?" IEEE Transactions on

5.6.4.13 – GA - Genetic Algorithm Selection

Genetic algorithms (Fogel 1999) have been used successfully in pattern selection (Chang and Lippmann 1991), (Kuncheva 1995), (Kangas 1999), (Ho, Liu *et al.* 2002). The basic idea is to consider sets of prototypes as chromosomes, and apply the genetic operators of replication, crossover, mutation, and natural selection to find the best set.

As has happened in many fields, the use of the evolutionary inspired algorithm yields quite good results, using moderate computing resources. No claim can be made on the optimality of the results, while most algorithmic approaches can guarantee that they are at least locally optimal. Thus, genetic algorithms should, if possible be followed by a local gradient descent based method, to ensure that at least locally, they are in fact minimal.

Interestingly, it was through the use of genetic algorithms that a counter-example to some optimality claims was found (Kuncheva and Bezdek 1998). In that paper, a 12 element set of prototypes was found, using genetic algorithms, that classifies without error the Iris dataset (see section 1.6.1 of Part II). On exactly the same data set, the other algorithm could find only a 14 element set⁶.

5.6.4.14 - RISE – Rule Induction from a Set of Exemplars

Rule Induction from a Set of Exemplars (RISE) is the name given by (Domingos 1995) to an algorithm that unifies rule induction with instance based learning. There are several versions of RISE, described in detail in (Domingos 1996) and (Domingos 1997). Version 3.1, described in (Domingos 1997) is the most recent one, and the best performer in benchmarks. It is arguable whether RISE should be considered a prototype based system. In our definition of prototype based systems, it is implicit that the stored entities have the same dimension as the data patterns. RISE is a rule inducing system, and as such it does not store the data patterns themselves, but the

Fuzzy Systems 7(3): 368 - 369. However, later mails exchanged between the authors, concluded that when the MCS algorithm is applied to the true Iris dataset, the results are even worse, since a 14 set consistent subset is found, when a 12 set consistent subset has been found.

⁶ Due to an error, a 13 element set was originally claimed.

rules extracted from them. However, we can see these rules as entities in a subspace of the original data pattern space, and thus as stored patterns. This view is implicit in RISE's algorithm for classifying new patterns, as it computes a "distance" between the new pattern and the stored rules. In accordance with this line of thought, rule generation is a form of feature selection, and this is explicitly mentioned in (Domingos 1997). Therefore, we consider RISE to be in the broad family of prototype based classifiers. Since RISE will try to form ever more general rules (and thus fewer rules), it can also be seen as a prototype minimization technique.

5.6.4.15 - RT – Reduction Technique

In (Wilson and Martinez 1997), three slightly different prototype minimization techniques are proposed, named Reduction Technique 1 (RT1), RT2, and RT3. The basic idea of RT1 is to build, for each prototype, a list of its nearest neighbors (belonging to same class), and a list of its so called associates. A prototype is said to be an associate of a given pattern if it is the closest prototype to that pattern that has a different class. In Q-set formalism (see Chapter II-1), it would be the first prototype in either R_0 or R_1 . The algorithm will attempt to remove each prototype in turn, checking to see if the number of associates that will become correctly classified outweighs the number of nearest neighbors that cease to be correctly classified.

The author points out that RT1 is very sensitive to the order by which prototypes are removed, and may sometimes enhance outliers instead of filtering them out. To reduce sensitivity to order, a more complex procedure, named RT2, is proposed. RT2 sorts the prototypes before attempting their removal, in such a way that "border" prototypes are removed last. Finally, since RT2 is still sensitive to outliers, RT3 is proposed. RT3 basically applied a data editing technique (similar to (Wilson 1972)) before applying RT2. This editing step, that as mentioned before improves most of the prototype minimization techniques, makes RT3 the best performer of the three techniques.

5.6.4.16 – RS - Random Selection

Although it might seem almost childish at first sight, random selection of prototypes has shown to produce reasonable sets of classification prototypes (Kuncheva and Bezdek 1998). The use of random selection, and random walk methods in general became possible with the advent of widely available powerful computers, and relies on pure luck to find the best solution. Naturally that luck is enhanced by attempting many different selections. Unlike genetic algorithm approaches where the randomness is guided with the genetic selection rules, in pure random selection no attempt is made to guide the process.

In this approach, a given number of prototypes are randomly selected from amongst the available ones, and the set with fewer classification errors is chosen. The number of prototypes selected in each run may be itself a random variable, or may be modified to increase/decrease the error rate until a satisfactory value is reached.

5.6.4.17 – SHC - Stochastic Hill Climbing Selection

Stochastic hill climbing is a simple optimization technique that randomly chooses a candidate direction, and moves along it if there is any decrease in the cost function. It has been used by (Kuncheva 2001) to search for minimal classifier sets. A prototype is randomly selected or excluded, and a certain fitness function is computed, that weighs classification accuracy and the size of the training set. If the value of the fitness function increases, the change is made permanent, and if not, it is reversed. In the paper where it is proposed, it performs worst than the combination of Hart's CNN and Wilson's ENN.

5.6.4.18 – SVM - Support Vector based methods

Support Vector Machines (SVM) are a particular case of kernel based methods presented earlier. A SVM will find the most representative patterns for the interclass boundary definition (called support vectors), and thus can also be used as a condensation algorithm. They do however have the inconvenient that they require a fair amount of training, and are thus computationally very hard to find. (Mitra, Murthy *et al.* 2000) proposes a mixed SVM/CNN algorithm, that basically uses the CNN procedures, but when deciding to add new prototypes, uses a SVM classifier instead of the nearest neighbor rule. Their experimental results show that this mixed method achieves considerably better condensation than the original CNN on very large datasets, but requires far less time than the pure SVM approach.

5.6.4.19 - DYNAGEN

(Laha and Pal 2001) proposes a method, called DYNAGEN that relies on a very small SOM to find initial candidate prototypes, and then follows a certain number of steps, somewhat similar to Chang's spanning tree procedures, to create new prototypes and adjusting them to better fit the data. It relies on 4 modification procedures: merging two prototypes; modifying a labeled prototype; splitting a prototype into two; and deleting a prototype. It then starts iterating applications of these 4 procedures, subject to data-dependant conditions, until the desired

accuracy is reached. Although very heuristic in its approach it was tested on standard datasets, and achieved a reasonable error rate with a very small number of prototypes.

5.6.4.20 – TS - Tabu search

Tabu search (Glover and Laguna 1997), is a optimization technique that tries to avoid local minima by “outlawing” search strategies that have to them. Since finding a minimal consistent subset is a optimization problem, it is only natural that tabu search has been used on this problem (Cerverón and Ferri 2001), (Zhang and Sun 2002). There are several ways in which tabu search can be applied, and both referenced papers use slightly different techniques, but a direct experimental comparison is not possible since they use different datasets. On the Iris dataset, (Zhang and Sun 2002) obtained a number of prototypes that varied between 11 and 15, having an average of 14 ± 0.8 .

The tabu search for consistent sets is not deterministic, depending both on the order by which the patterns are presented, and on random internal variables, and thus there is always a certain variance associated with the number of prototypes selected for any given problem.

5.6.4.21 – Simulated Annealing

Simulated Annealing is an optimization technique inspired in physics (Kirkpatrick, Gelatt Jr. *et al.* 1983). It tries to simulate what happens in certain materials when a decrease in temperature leads to rearrangements of particles. When used for optimization, this technique consists of allowing random searches around known solutions, providing the cost function does not increase more than a certain amount, that corresponds to the simulated “temperature”. As the optimization progresses the “temperature” decreases, (i.e. the allowed increase in the cost function decreases,) until only solutions that have a lower cost are accepted (i.e. the temperature is zero), and thus the optimization algorithm becomes greedy.

Simulated Annealing has been used for prototype minimization by (Huang, Liu *et al.* 1996; Decaestecker 1997; Liu and Nakagawa 2001; Devi and Murty 2002), with promising results.

5.6.4.22 – DMCNN - Devi Modified CNN

A recent paper (Devi and Murty 2002) proposes an incremental CNN based method, called Modified Condensed Nearest Neighbor, to which we add the author's name (DMCNN). The method is really a mix between the simple means and CNN. The process starts by using only the simple means of each class as prototype for that class. It then finds which patterns are misclassified by this first set of prototypes, and calculates means of these patterns, adding them to the prototype set. This process is iterated until no more patterns are misclassified. A further improvement of the procedure deletes prototypes that are no longer essential to keep the error rate from growing. The authors present encouraging experimental comparisons with other methods, on standard and private datasets. Although we have not experimented with their method, it seems intuitive that this procedure is not applicable to datasets where each class may have disjoint areas of significant probability density, or is "highly non-convex".

5.6.4.23 - ICF - Iterative Case Filtering

In a recent paper, (Brighton and Mellish 2002) proposes a method named Iterative Case Filtering (ICF). It is based on the ENN (see 5.6.4.3) with the iterations proposed by (Tomek 1976), but uses the concepts of neighbors and associates of RT (see 5.6.4.14), albeit with different names, to decide on whether remove prototypes or not. Although its name is very similar to ICA (see 5.6.4.4), the principles used in the iterations are quite different. In some of the benchmarks, ICF outperformed the other methods with which it was compared, namely RT3 and ENN.

5.6.5 – Benchmark comparisons between methods

As stated before, none of the discussed algorithms is universally better than the others. The performance of each is extremely problem dependant, and different parameters can be used when judging them. Nevertheless, it is useful to compare the different methods in various toy and practical problems. Many authors have produced such comparisons. Since there are so many variants, there is no global comparison, but for reference, we present a list of some papers and the methods compared in Table 4.

Paper	Algorithms compared
(Bezdek, Reichherzer <i>et al.</i> 1998)	Chang, Modified Chang, LVQ, GLVQ-F, DR, Dasarathy
(Kuncheva and Bezdek 1998)	Chang, Modified Chang, GA, RS, Dasarathy, M-FCM, simple means, LVQ, GLVQ-F
(Kuncheva 2001)	CNN, ENN, GA, TS
(Liu and Nakagawa 2001)	LVQ, MLVQ3, GLVQ, DSM, MCE, SA, MSE, MAXP,DA, k-means, PMSE, CMSE, MAXP MAXP1, k-nn
(Kangas 1999)	GA, selection mean,
(Devi and Murty 2002)	GA, SA, TS , CNN, DMCNN.
(Huang, Chiang <i>et al.</i> 2002)	LVQ, SA, LVQ-H
(Cerverón and Ferri 2001)	TS, MCS, CNN, MNV
(Zhang and Sun 2002)	TS,CNN,MCS
(Bezdek and Kuncheva 2001)	CNN, ENN, RS, GA, TS, LVQ, DSM, GLVQ-F, FCM, Bootstrap
(Wilson and Martinez 1997)	k-nn, RT1, RT2, RT3, H-IB3
(Brighton and Mellish 2002)	ICF, RT3, ENN

Table 4- List of some papers that compare prototype minimization techniques

Just for the sake of curiosity, in Table 5 we present the best attempts to produce a consistent subset for the Iris dataset, discussed in Chapter II-1.

Algorithm	N° of Prototypes	Paper
Mod Chang	11	(Kuncheva and Bezdek 1998)
TS	11	(Zhang and Sun 2002)
GA	12	(Kuncheva and Bezdek 1998)
Dasarathy	14	(Dasarathy 1994)
Chang	15	(Kuncheva and Bezdek 1998)
QSET	17	(Lobo 2002)
CNN	18	(Zhang and Sun 2002)

Table 5 - Smallest size of consistent subsets obtained for the Iris data

PART I

CHAPTER 6

Validation

6.1 – Introduction

After obtaining a classifier by any of the discussed methods, it is important to have an idea how reliable that classifier is, i.e., when a new pattern is given to the classifier, what is the probability that it will be correctly classified? The answer to this question is the true error rate.

The true error rate, also known as actual error rate, conditional error rate, or e_T , is defined as the expected probability of misclassifying a randomly selected pattern (Webb 1999). In some very

particular cases, when the exact probability distribution of the data is known, this error may be calculated exactly. Such is the case for the simple problems presented in Appendix A and B. Generally, the true probability distribution of the data is not known, and thus the true error rate must be estimated from the data itself.

To estimate error rates from data, we must present patterns for which the true class is known to the classifier, and see whether it assigns them their true class. If it does not, we consider that the classifier committed an error. The error rate will be the total number of errors divided by the total number of patterns tested, and is usually expressed as a percentage:

$$\text{Error rate} = e = \frac{\text{number of missclassified patterns in the given set}}{\text{total number of patterns in the given set}} \times 100 \quad (48)$$

Depending on the dataset on which the error rate is computed, we will obtain different estimates of errors.

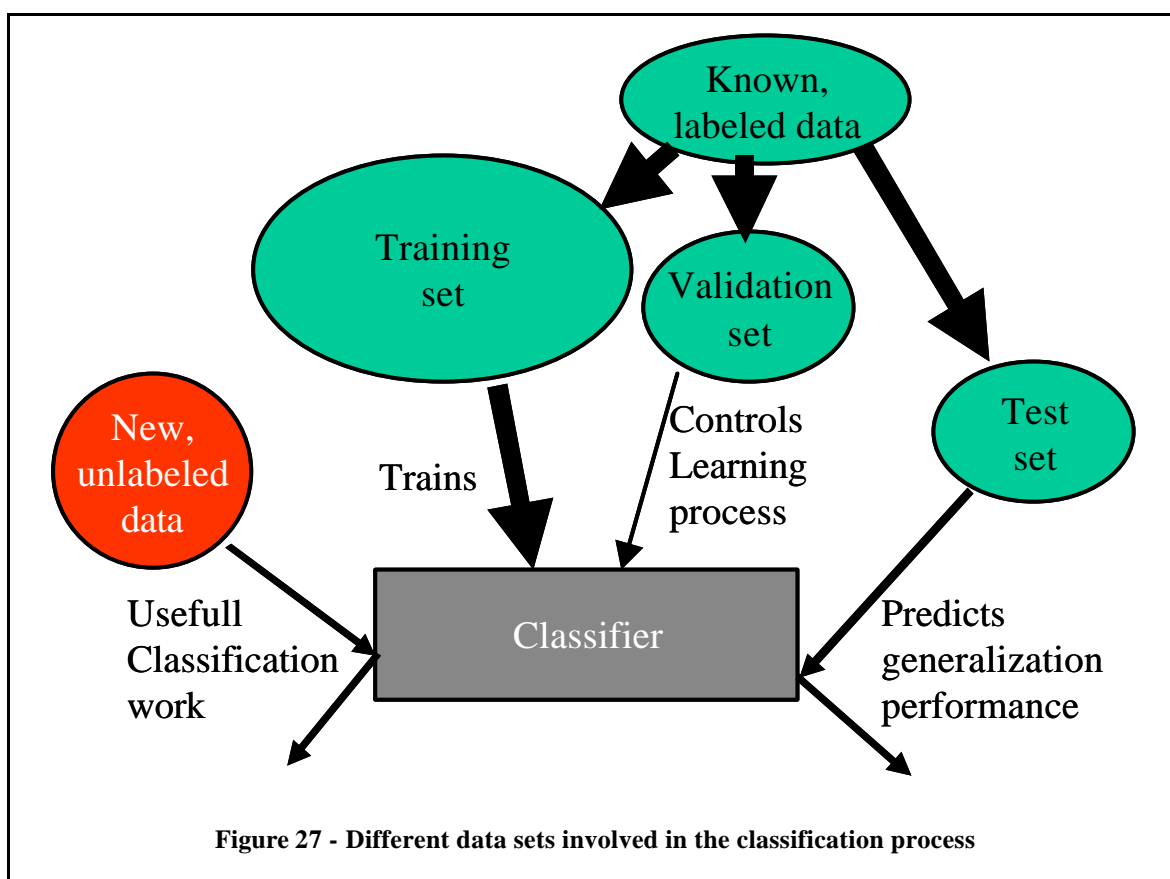
If it were possible to use an infinitely large dataset, the error rate on that dataset would be an unbiased (and zero variance) estimate of the true error. Naturally, it is impossible to obtain and use such a set. Therefore, we must use finite size sets to estimate the true error rate. Let us then clarify what sets may be used, and by which names they are used.

6.2 – Known sets, training sets, validation sets, and test sets

The set of patterns for which the associated class is known is sometimes called the *training set*, since it can be used to train the classifier. However when we are designing classifiers, it is convenient to use only a subset of these patterns for actual training, and keep the rest of them to check the performance of the classifier. Thus, the name training set should be used in a more precise manner, and a precise definition should be given for these different sets. Since different authors use slightly different names for these sets, we shall define them as follows:

- known set - Set of all patterns for which the class is known (X_{known}).
- training set - Set of patterns used to train (or design) the classifier (X_{train}).

- validation set - Set of patterns available during the training process, but used only to assess that process, or select one of the available models (X_{valid}).
- test set - Set of patterns not used at all during the training process. These patterns may be used only for the final assessment of the classifier (X_{test}).



While the definition of known set and training set are quite clear, the difference between test and validation set can be confusing, and their use (or even existence) depends a lot on the method used to obtain the classifier. The validation set, although not actively used for training, in the sense that the design parameters are not derived from it, is available during training as a means of checking the performance of the system. In a neural network, for example, it may be used to check that the network is not overfitting, and thus be used to generate the stopping criteria of the training process. On the other hand, the test set must never be used at all during the design process, not even for iterating it, for that would “shape” the classifier to its particular characteristics. Another difference between these two sets is that in many design algorithms (for example, when leave-n-out multiple classification trees (Breiman, Friedman *et al.* 1984) are used), validation and training sets are interchanged, while test sets are always left out of the

process. Finally, it should be noted that many authors use the term test set for what we here call validation set⁷.

It is arguable whether the use of test sets is important or not. In practice, when the available known set is small, only a training set and, if necessary, a small validation set are used. As the results obtained with the test set do not influence the design of the classifier, only our confidence in it, we are usually quite happy to trade it for a better training and validation set. In any case, when a classifier is necessary for a real world problem, the classifiers obtained with training and test sets should only be used as a means of estimating the true error. The final classifier, that will perform the real classification task, should use all available known data for training.

Even when the known set is large, some authors do not use an independent test set, because in that case, the training and validation sets will be good unbiased estimators of the distribution of the data. As long as some other technique can be used to guarantee that there is no overfitting, the error rates obtained with the training and validation sets will not differ considerably from those obtained with the test set.

6.3 – Error rate estimates

As mentioned above, the true error rate cannot generally be computed, and will usually be estimated from available data.

The most optimistic estimate is the apparent error rate, also known as resubstitution error rate. This error rate is calculated by computing the proportion of training patterns that are misclassified. Since those patterns were used for training, the classifier will have been fine tuned to try and classify them, and thus the apparent error rate will be lower than that obtained

⁷ When I started to write this thesis, I thought it would be clearer to call *test set* to the data used to “test the progress of the training process”, and use the term *validation set* to identify the data used for “final validation of the classifier”. This convention would take into account that many authors use what they call “test set” to control the overfitting of the training process. However, Prof. Joseph Kittler, although recognizing that different communities call different and sometimes opposing names to the same things, convinced me that it is better to stick to the convention used in the pattern recognition community, which is explained in the text.

with an independent test set. The bias of the resubstitution error rate can be reduced using jackknife techniques (Miller 1974) (Webb 1999) but will always be optimistic.

A more reliable estimate for the true error can be obtained using an independent test set. This error rate estimate is known as holdout estimate (Devijver and Kittler 1982), and will depend the size of the test set (assuming that this test follows the probability distribution of the problem at hand). The following deduction of the confidence of the holdout estimate is based on (Mitchell 1997).

Let us assume that the true error rate for a given classifier is p . Given any pattern in the test set, it will be incorrectly classified with probability p , and correctly classified with probability $1-p$. Given n such test patterns, the total number of errors r will follow a binomial distribution $P(r)$ given by:

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \quad (49)$$

The expected value for the number of errors, and its standard deviation will be

$$E[P(r)] = np \quad (50)$$

$$\mathbf{s}_p = \sqrt{np(1-p)} \quad (51)$$

We may thus obtain an unbiased estimate for p using

$$p = \frac{E[P(r)]}{n} \quad (52)$$

and substituting $E[P(r)]$ by the r obtained in a given experiment leading to the estimator

$$\hat{p} = \frac{r}{n} . \quad (53)$$

The standard deviation of this estimator will be

$$\mathbf{s}_{\hat{p}} \approx \mathbf{s}_p = \frac{\mathbf{s}_{P(r)}}{n} = \frac{\sqrt{np(1-p)}}{n} = \sqrt{\frac{p(1-p)}{n}} \approx \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} = \sqrt{\frac{r(n-r)}{n^3}} \quad (54)$$

If we wish to calculate confidence intervals for p , we may, providing $n > 30$ and $np(1-p) > 5$, approximate the binomial distribution of p to a Gaussian distribution (Mitchell 1997), and in that case, we will have:

$$p \in \left[\hat{p} - Z_N \mathbf{s}_{\hat{p}}, \hat{p} + Z_N \mathbf{s}_{\hat{p}} \right], \quad (55)$$

where Z_N is a constant, function of the desired confidence.

Thus, the larger the test set, the better our estimate will be. If data can be generated at low cost, a large test set will yield a very accurate estimate of the true error. Even when the exact true error can be calculated analytically, as is the case in Appendixes A and B, using a test set may be simpler and more cost efficient.

However, if the total number of known patterns is limited, increasing the test set will decrease the number of patterns available for training, and thus produce a less reliable classifier. The holdout estimate will thus be a pessimistic estimate of the true error.

Having both a optimistic apparent error rate and a pessimistic holdout error rate, bootstrapping techniques (Efron 1979; Efron; Jain, Dubes *et al.* 1987; Efron 1990) may be used to obtain an estimate of the true error that will have less bias than any of the above.

The less biased holdout estimate would be obtained using almost all known patterns for training, and very few (in the limit only one) for testing. The variance of this estimate would be very large, but can be reduced using a technique called cross-validation.

The cross validation error rate is also known as U-method, leave-one-out (or leave- n -out), rotation estimation (Kohavi 1995), or deleted estimate error rate (Webb 1999). It is obtained by partitioning the known data into m sets on n patterns. We then select one set as test set, and use the remaining $m-1$ as training set, and repeat the process m times selecting a different test set each time. The average error rate obtained, although slightly pessimistic, will be closer to the true error than a standard holdout estimate. The minimum bias will be obtained when all but one patterns are used for training, leading to the pure leave-one-out technique. Other schemes for partitioning the available data into different training and test sets are possible, and discussed in (Mullin and Sukthankar 1999).

Many good and comprehensive reviews of how to measure error rates and compare classifiers have been published, such as (Fukunaga and Hayes 1989; Michie, Spiegelhalter *et al.* 1994; Dietterich 1998; Lim, Loh *et al.* 2000).

6.4 – Confusion Matrices

The error rate estimates presented in the previous section give equal importance to all types of errors, i.e., as long as the class given by the classifier is not the same as the true class of the patterns we increase the number of errors by 1.

There are a few reasons for wanting to know more about what errors are being committed. The main one is that there may be a cost associated with each type of error. If, for example, we want to detect intruders with an alarm system, the cost of not detecting an intrusion when it occurs (a false-negative classification), is greater than the cost of thinking there is an intrusion when none has occurred (a false-positive classification). This problem has been the subject of a lot of attention in classical detection theory, and has been thoroughly addressed in many text books, such as (Kay 1988; Kay 1998), the latter including examples of MATLAB code for implementing most of the techniques.

Another reason for wanting to know more about what types of errors are occurring has to do with the exploratory data analysis issues discussed in chapter 4. By observing which classes tend to be misclassified, and what those misclassification are, we may be able to add some pre-processing or extra classifier to distinguish amongst those cases.

The most common way of presenting a detailed description of the errors is the confusion matrix (Fukunaga 1990). The rows of a confusion matrix represent the actual class of the patterns, while the columns represent the class assigned by the classifier, as seen in Table 6.

Assigned class \ True class	A	B	C
A	80	20	0
B	13	87	0
C	1	0	99

Table 6- Example of a confusion matrix. Numbers on the diagonal correspond to correctly classified patterns. In this case, it clear that class C is correctly classified, but there are errors in distinguishing class A from class B.

PART II

Original contributions

PART II

CHAPTER 1

Q-Sets: A Boolean formalization for minimizing prototype-based classifiers

1.1 - Introduction

As overviewed in Chapter 5 of Part I, prototype-based classifiers constitute a broad and very important family of non-parametric classifiers. As stated, this family of classifiers has 2 common characteristics:

- a) The classifier stores labeled patterns, which we have called prototypes, which are of the same nature as the patterns that are to be classified.
- b) When a new pattern is presented for classification, a similarity measure is used to find the prototype or prototypes that are nearest to it, and the class is decided based on the classes of these classes

As was seen, a lot of effort has been put into finding a small set of prototypes that will perform the classification efficiently and with as low an error rate as possible. The holy grail of this quest is to find the minimal set that will perform such a task. Under certain assumptions, (Wilfong 1991) proved that finding this set is equivalent to a certain special case of the Disk Cover problem, that is known to be NP-Complete.

In this chapter, we will present a new formalization for the problem of finding the minimum set of prototypes to classify a given set of patterns, that was first proposed in (Lobo, Swiniarski *et al.* 1998), and that transforms this problem into Boolean function manipulation problems. This link between the areas of classification and Boolean algebra, brings not only insight into the core of our problem, but also a vast array of techniques that can be used to efficiently compute small sets of classifier prototypes.

1.2 - Informal presentation of the theory

Before going into the formal and complete description of the Q-set formalization, let us first overview the method informally. While not complete, this overview is nonetheless accurate and gives an insight that makes the necessarily detailed formalization more palatable.

Our aim is to select a few prototypes from a large set of candidates that will correctly classify a given set of training patterns.

The basic idea behind what we have come to call Q -set formalization is quite basic, and requires us just to invert the nearest neighbor rule when training the classifier: when using the nearest neighbor rule, each pattern's nearest neighbor must have the same class as that pattern. If a given pattern does not meet this requirement, i.e. if its nearest neighbor has a different class, then it will be misclassified if we use all candidate prototypes, and we will ignore it when selecting the final prototypes. As for the rest of the patterns, they will have at least one, and generally a set of prototypes with the same class that are closest to it than any prototype of another class. This is, at first

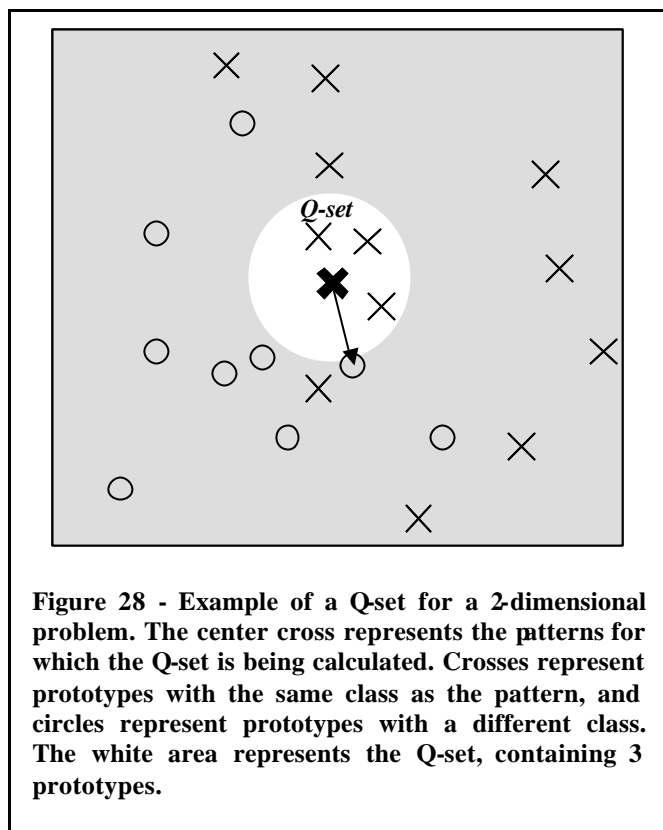


Figure 28 - Example of a Q -set for a 2-dimensional problem. The center cross represents the patterns for which the Q -set is being calculated. Crosses represent prototypes with the same class as the pattern, and circles represent prototypes with a different class. The white area represents the Q -set, containing 3 prototypes.

glance, the Q -set of that pattern: the set of prototypes that will correctly classify it using the nearest neighbor rule. In Figure 28 we can see a graphical example of a Q -set for a 2-dimensional problem. The existence of any of the prototypes of the Q -set in the final classifier is sufficient to guarantee that that pattern will be correctly classified. If we consider a Boolean function that indicates whether that pattern is correctly classified (having logical value of 1) or not (a logical value of 0), as a function of the presence in the final classifier of the candidate patterns, it will be a logical-OR of the presence of the prototypes in the Q -set. Without trying to be rigorous, we can say that

$$\text{Pattern } \mathbf{x} \text{ is correctly classified} = Q(\mathbf{x}) = \mathbf{p}_a \cup \mathbf{p}_b \cup \dots \cup \mathbf{p}_k = \sum_{p \in Q_{set}(\mathbf{x})} \mathbf{p} \quad (56)$$

where \mathbf{p}_a is a Boolean variable with the value 1 if and only if prototype \mathbf{p}_a is in the final classifier. From the point of view of the training set, any single one of the prototypes of the Q -set is equally good for the classification task, but we should choose one that apart from classifying this patterns also classifies others. The basic idea, is that we must classifier all the patterns: pattern \mathbf{x}_1 , $\mathbf{x}_2, \dots, \mathbf{x}_n$ (i.e. all the patterns). Once again, without being too formal we may say that

$$\text{All patterns are correctly classified} = Q(\mathbf{x}_1) \cap Q(\mathbf{x}_2) \cap \dots \cap Q(\mathbf{x}_n) = \prod_{i=1}^n Q(\mathbf{x}_i) = \prod_{i=1}^n \sum_{p \in Q_{set}(\mathbf{x}_i)} \mathbf{p} \quad (57)$$

Since the conjunction and disjunction operators of Boolean algebra are both distributive in relation to each other, we can rearrange the above equation into

$$\text{All patterns are correctly classified} = \prod \sum \mathbf{p}_k = \sum \prod \mathbf{p}_k' \quad (58)$$

This last form, a sum of products, is ideal for finding the minimum number of prototypes. If we simply select the prototypes that appear in the shortest term of the sum, then the global function will be one, and we will correctly classify all patterns in the training set that were classified by the ensemble of all candidates. The shortest term of a Boolean function written as a sum-of-products is its minimum prime implicant, and the problem has been studied extensively. Unfortunately it is a hard problem to solve since factoring out the original function can be computationally very expensive, so a number of heuristic methods have been devised, and later in this thesis we propose one that is particularly suited to the classification problem.

What we have just presented is what we will later call the *positive-only Q-set* approach, which is a particular case of the broader formalization. In this positive-only approach, we assumed that we had to have at least one of the prototypes of a pattern's Q-set in the final classifier. We assumed that because if that were not the case, there would be another prototype, of a different class, that would be nearer to the pattern than any other prototype, thus yielding a classification error. But is that really true? What if that prototype with the wrong class was not selected for the final prototype? There may be another prototype of the same class as pattern x , that is further away, but that will perform a correct nearest neighbor classification if the "bad" prototype is removed. We must then revise our concept of Q -set and associated q -function to include the possibility that if we *exclude* a certain prototype, then other prototypes, further away, may be acceptable choices. Thus, the Boolean function that determines whether the classifier performs correctly as a function of the prototypes it includes will now have negations, and will cease to be in the convenient conjunctive normal form (CNF). We will later see that it takes the form

$$\text{All patterns are correctly classified} = \prod \sum ((\prod \bar{\mathbf{p}}_r)(\sum \mathbf{p}_q)) \quad (59)$$

This more general approach, while far more complex may allow us to force the correct classification of all the training set patterns, even those that were originally incorrectly classified. We shall see later that is equivalent to the rather well known Satisfiability Problem, that was the first problem proved to be NP-Complete (Cook; Cook; Stoffel, Kunz *et al.* 1997). Since no one has, up to date, proved that NP=P, no simple solution to this classification problem is yet available. However, since no one has been able to prove otherwise, there is still hope, and in the

mean time, a vast array of methods have been developed to find acceptable solutions. The main point is that we can use well known techniques from other areas to simplify our classifier design problem.

Now that a rather intuitive introduction has been made, we may proceed with a formal description of the Q-sets.

1.3 - Theoretical framework

Let X be a space of n -dimensional patterns x , for which we want to design a classifier. Let $X_{train} \subset X$ be the set of patterns $x \in X$ that we have available for designing that classifier.

Let s be a measure of similarity defined in $s : X \times X \rightarrow [0,1]$. It is not important for this measure to give values in the range $[0,1]$, but it can be done without loss of generality. As a similarity measure, it is necessary that $s(x,y)=1 \iff x=y$. We choose to use a similarity measure as it is more general than a true distance measure: any distance measure may be mapped to a similarity measure with the required constraints, but the inverse is not true.

Let P be the set of classifier prototypes p available ($p \in X$). These prototypes are simply the patterns, for which the true class is known, that will later be used to perform the classification. Let $H = \mathcal{P}(P)$ the power set of P , i.e., the set of all the subsets of P . Let $h \in H$ be a classifier, constructed with prototypes available in P (sometimes also called the model for X (Mitchell 1997)). Let us now define an indicator function $h(p) : X \rightarrow \{0,1\}$, associated with each set h (Schneeweiss 1989), as

$$h(p) = \begin{cases} 1 & \text{if } p \in h \\ 0 & \text{otherwise} \end{cases} \quad (60)$$

Later, for the sake of clarity and when there is no possible confusion about whether we are referring to the prototype p or the indicator function $h(p)$, we shall use p to mean that $h(p)=1$ and \bar{p} to mean that $h(p)=0$.

Let C be the set of all possible classes defined as $C = \{c_1, \dots, c_{|C|}\}$. Let $c(x, h)$ be a function defined in $c : X \times H \rightarrow C$, that assigns the pattern x to a class c , according to the model h . That

function $c(\mathbf{x}, \mathbf{h})$ computes the similarity between pattern \mathbf{x} and each of the prototypes $\mathbf{p} \hat{I} \mathbf{h}$, and assigns to \mathbf{x} the same class as the prototype \mathbf{p} that is most similar to \mathbf{x} . Let us denote the true class of pattern \mathbf{x} as $c_{true}(\mathbf{x})$.

The classifier with fewer prototypes is the one that minimizes the integer-valued sum (that we shall name classifier cost) given in the following equation

$$\text{Classifier cost} = \sum_{p \in P} h(p) \quad (61)$$

It must be noted, however, that if we impose no restrictions, the trivial solution of considering no prototypes (i.e. $\forall p, h(p)=0$) would be the minimum. It would also not classify anything, so when minimizing equation (61), we must impose the restriction that the classifier should classify correctly (or with a certain given error rate) the set of patterns \mathbf{X}_{train} .

To do this, we introduce the concept of Q and R sets, and the associated “good” and “bad” neighborhoods.

1.3.1 - Definition of Q and R sets

Let us introduce the concept of “good neighborhood” order i , of the pattern \mathbf{x} , denoted by $\mathbf{Q}_i(\mathbf{x})$, or \mathbf{Q} -set order i of \mathbf{x} , and “bad neighborhood” order i , of the pattern \mathbf{x} , denoted by $\mathbf{R}_i(\mathbf{x})$, or \mathbf{R} -set order i of \mathbf{x} .

The *good neighborhoods* of \mathbf{x} are sets of prototypes $\mathbf{p} \in \mathbf{P}$ that have the same class as \mathbf{x} , while *bad neighborhoods* are sets of prototypes that have a different class. The order of the neighborhood is determined by the similarity between the given pattern \mathbf{x} and the prototypes of the neighborhood. The prototypes of the *bad neighborhood* order 0 are closer to the pattern \mathbf{x} than any prototype of the same class as \mathbf{x} . If the bad neighborhood order 0 of all patterns is empty ($\mathbf{R}_0(\mathbf{x}) = \{\}$), then the prototypes can certainly classify the whole set without errors, if they are all included in the final classifier. The good neighborhood order 0 of \mathbf{x} , $\mathbf{Q}_0(\mathbf{x})$, is the set of prototypes \mathbf{p} that have the same class as \mathbf{x} , are not as similar to \mathbf{x} as any of the prototypes of the bad neighborhood order 0 , but are closer than any other prototypes of a different class (namely they will be closer than those of the bad neighborhood order 1). For the sake of simplicity we shall use “+” to represent the set

reunion ($A+B \equiv A \cup B$) and “-“ to represent the set exclusion ($A-B \equiv A \setminus B$). We can now define recursively all the good and bad neighborhoods formally as:

$$\mathbf{R}_0(\mathbf{x}) = \{ \mathbf{r} \in \mathbf{P} \mid c(\mathbf{x}) \neq c(\mathbf{r}), \forall \mathbf{p} \in \mathbf{P}, c(\mathbf{x}) = c(\mathbf{p}), s(\mathbf{x}, \mathbf{r}) > s(\mathbf{x}, \mathbf{p}) \} \quad (62)$$

$$\mathbf{Q}_0(\mathbf{x}) = \{ \mathbf{p} \in \mathbf{P} \mid c(\mathbf{x}) = c(\mathbf{p}), \forall \mathbf{r} \in \mathbf{P} - \mathbf{R}_0(\mathbf{x}), c(\mathbf{x}) \neq c(\mathbf{r}), s(\mathbf{x}, \mathbf{p}) > s(\mathbf{x}, \mathbf{r}) \} \quad (63)$$

$$\mathbf{R}_1(\mathbf{x}) = \{ \mathbf{r} \in \mathbf{P} - \mathbf{R}_0(\mathbf{x}) \mid c(\mathbf{x}) \neq c(\mathbf{r}), \forall \mathbf{p} \in \mathbf{P} - \mathbf{Q}_0(\mathbf{x}), c(\mathbf{x}) = c(\mathbf{p}), s(\mathbf{x}, \mathbf{r}) > s(\mathbf{x}, \mathbf{p}) \} \quad (64)$$

$$\mathbf{Q}_1(\mathbf{x}) = \{ \mathbf{p} \in \mathbf{P} - \mathbf{Q}_0(\mathbf{x}) \mid c(\mathbf{x}) = c(\mathbf{p}), \forall \mathbf{r} \in \mathbf{P} - (\mathbf{R}_0(\mathbf{x}) + \mathbf{R}_1(\mathbf{x})), c(\mathbf{x}) \neq c(\mathbf{r}), s(\mathbf{x}, \mathbf{p}) > s(\mathbf{x}, \mathbf{r}) \} \quad (65)$$

or in general

$$\mathbf{R}_n(\mathbf{x}) = \left\{ \mathbf{r} \in \mathbf{P} - \sum_{j=0}^{n-1} \mathbf{R}_j(\mathbf{x}) \mid c(\mathbf{x}) \neq c(\mathbf{r}), \forall \mathbf{p} \in \mathbf{P} - \sum_{i=0}^{n-1} \mathbf{Q}_i(\mathbf{x}), c(\mathbf{x}) = c(\mathbf{p}), s(\mathbf{x}, \mathbf{r}) > s(\mathbf{x}, \mathbf{p}) \right\} \quad (66)$$

$$\mathbf{Q}_n(\mathbf{x}) = \left\{ \mathbf{p} \in \mathbf{P} - \sum_{j=0}^{n-1} \mathbf{Q}_j(\mathbf{x}) \mid c(\mathbf{x}) = c(\mathbf{p}), \forall \mathbf{r} \in \mathbf{P} - \sum_{i=0}^n \mathbf{R}_i(\mathbf{x}), c(\mathbf{x}) \neq c(\mathbf{r}), s(\mathbf{x}, \mathbf{p}) > s(\mathbf{x}, \mathbf{r}) \right\} \quad (67)$$

The construction ceases when the first \mathbf{Q} -set or \mathbf{R} -set is empty. The \mathbf{R} -set of the same order contains patterns that have a different class and are further away from the pattern \mathbf{x} than any prototype with the same class. For practical purposes this last \mathbf{R} set may be ignored, but its simple definition permits also a recursive but symmetrical definition of the \mathbf{R} and \mathbf{Q} sets. The total number of \mathbf{Q} sets for a pattern \mathbf{x} will be the order of the last plus one.

In Figure 29 we may see a graphical example of these sets for a 2-dimensional problem. Let us now reflect for a moment on the meaning and properties of these Q and R sets.

Each Q and R set of a given pattern x encompasses patterns that are in a hyperspherical crown around that pattern. The space around this pattern is thus divided in multiple layers, like skins in an onion. The width and number of these crowns will vary considerably from case to case. There will be at least two non-empty sets for any given pattern (and thus two crowns), and there may be as many sets as patterns in P . The simplest case, from a classification point of view,

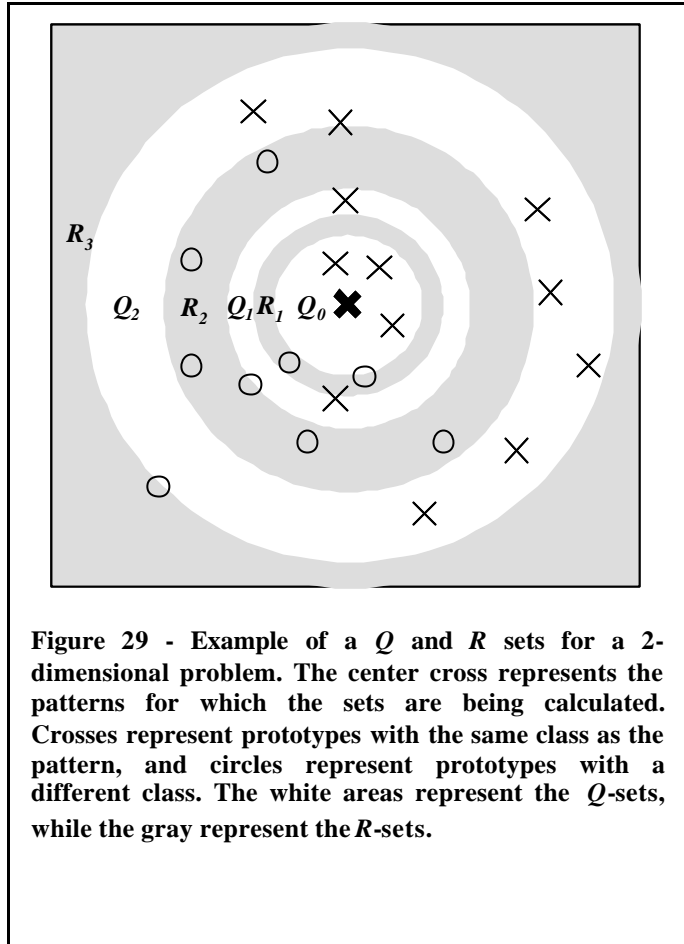


Figure 29 - Example of a Q and R sets for a 2-dimensional problem. The center cross represents the patterns for which the sets are being calculated. Crosses represent prototypes with the same class as the pattern, and circles represent prototypes with a different class. The white areas represent the Q -sets, while the gray represent the R -sets.

is when there are only 2 non-empty layers, with $R_0 = \{ \}$, $Q_0 = \{ p \in P : c(x) = c(p) \}$, $R_1 = \{ p \in P : c(x) \neq c(p) \}$. In such a case, any single one of the prototypes of the same class as x can be used to classify it correctly.

In most cases, $R_0(x)$ will be an empty set. When this is not the case, then using the original nearest neighbor rule with all available prototypes in P will result in a classification error for pattern x . Whenever there is a pattern for which $R_0(x) \neq \{ \}$, there may be no subset of P that classifies the given set X without errors, as we shall see later.

1.3.2 - Partial and generalized q -functions

Having defined the Q and R sets of patterns, we may now define the associated Boolean indicator functions. For reasons that will become clear later, let us call these *Partial Qfunctions*. Let us then define the functions $q_{partial}, r_{partial} : X \times H \times n \rightarrow \{0,1\}$ as

$$q_{\text{partial}}(h, \mathbf{x}, n) = \begin{cases} 1 & \text{iif} & \exists \mathbf{p} : h(\mathbf{p}) = 1, \mathbf{p} \in Q_n(\mathbf{x}) \\ 0 & \text{otherwise} & (\forall \mathbf{p}, \mathbf{p} \in Q_n(\mathbf{x}) \Rightarrow h(\mathbf{p}) = 0) \end{cases} \quad (68)$$

and likewise

$$r_{\text{partial}}(h, \mathbf{x}, n) = \begin{cases} 1 & \text{iif} & \exists \mathbf{p} : h(\mathbf{p}) = 1, \mathbf{p} \in R_n(\mathbf{x}) \\ 0 & \text{otherwise} & (\forall \mathbf{p}, \mathbf{p} \in R_n(\mathbf{x}) \Rightarrow h(\mathbf{p}) = 0) \end{cases} \quad (69)$$

Since the model membership functions $h(\mathbf{p})$ are Boolean, the definitions given for the partial q-functions are equivalent to the following:

$$q_{\text{partial}}(h, \mathbf{x}, n) = \bigcup_{\mathbf{p} \in Q_n(\mathbf{x})} h(\mathbf{p}) \quad (70)$$

We may also use the more practical notation common when dealing with Boolean functions, and represent the disjunction as the Boolean sum, or

$$q_{\text{partial}}(h, \mathbf{x}, n) = \sum_{\mathbf{p} \in Q_n(\mathbf{x})} h(\mathbf{p}) \quad (71)$$

Naturally, the partial R-function will be

$$r_{\text{partial}}(h, \mathbf{x}, n) = \sum_{\mathbf{p} \in R_n(\mathbf{x})} h(\mathbf{p}) \quad (72)$$

Let us now introduce the concept of *generalized q-function*, defined in $q : \mathbf{X} \times \mathbf{H} \rightarrow [0,1]$:

$$q(h, \mathbf{x}) = \begin{cases} 1 & \text{iif} & c(\mathbf{x}, h) = c(\mathbf{x}) \\ 0 & \text{otherwise} & \end{cases} \quad (73)$$

Simply stated, the generalized q-function is 1 when the pattern is correctly classified by model h .

THEOREM

Given a model h and a pattern \mathbf{x} ,

$$q(h, \mathbf{x}) = \sum_{n=0}^{|\mathcal{Q}(\mathbf{x})|} \left(\prod_{i=0}^n \bar{r}_{\text{partial}}(h, \mathbf{x}, i) \right) \cdot q_{\text{partial}}(h, \mathbf{x}, n) \quad (74)$$

PROOF

For $c(\mathbf{x}, h) = c_{true}(\mathbf{x})$, it is necessary that the closest prototype, in model h , of pattern \mathbf{x} , have the same class as \mathbf{x} . Therefore, it is necessary that at least one prototype of the correct class (i.e. of a *good neighborhood*) exists, i.e., $\exists n: q_{partial}(h, \mathbf{x}, n) = 1$, and that no prototype of a different class (i.e., of a *bad neighborhood*), of lower order exist, i.e., $\forall m \leq n, r_{partial}(h, \mathbf{x}, m) = 0$, or alternatively $\bar{r}_{partial}(h, \mathbf{x}, m) = 1$. Thus, if any of the products $q_{partial}(h, \mathbf{x}, n) \cdot \prod_{i=0}^n \bar{r}_{partial}(h, \mathbf{x}, i) = 1$, then $q(\mathbf{x}) = 1$.

Since this may occur for any of the orders of the neighborhood, we have

$$q(h, \mathbf{x}) = \sum_{n=0}^{|Q(\mathbf{x})|} \prod_{i=0}^n \bar{r}_{partial}(h, \mathbf{x}, i) \cdot q_{partial}(h, \mathbf{x}, n) \quad (75)$$

Q.E.D.

It must be noted that, for any given pattern, its q -function is a Boolean function of the indicator functions $h(\mathbf{p})$. Unfortunately, the form given is not canonical. Its logical value may be changed by changing the values of these functions, i.e., by including or excluding prototypes \mathbf{p} from the model h . We must therefore choose an assignment of $h(\mathbf{p}_j)$ for $j=1$ to $|\mathbf{h}|$ that forces $q(h, \mathbf{x})$ to be true. For a single pattern this is always possible, provided that at least one of the Q -sets is non-empty.

1.3.3 - Correctness function

When designing the classifier, we usually want it to classify all patterns of the given training set X_{train} without errors. If we are willing to accept errors, we may consider only a subset of X_{train} and call that subset X_{train} . Let us define our final function, which we shall call correctness function, as $Correct : X \times H \rightarrow \{0,1\}$:

$$Correct(X_{train}, h) = \begin{cases} 1 & \text{iif } \forall \mathbf{x} \in X_{train}, c(\mathbf{x}, h) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (76)$$

From the above definition it is clear that we can define $Correct(X_{train}, h)$ as a function of the q -functions of the patterns in X_{train} :

$$Correct(X_{train}, h) = \prod_{j=1}^{|X_{train}|} q(\mathbf{x}_j, h) = \prod_{j=1}^{|X_{train}|} \sum_{n=0}^{|Q(\mathbf{x}_j)|} \left(\prod_{i=0}^n \bar{r}_{partial}(h, \mathbf{x}_j, i) \right) \cdot q_{partial}(h, \mathbf{x}_j, n) \quad (77)$$

Going back to the original problem of minimizing the classifier cost $\sum_{p \in P} h(p)$, it is now clear that this minimization must be done with the constraint that $Correct(X_{train}, h) = 1$.

We shall now analyze a particular case where this minimization can be done efficiently, before going on to the more general case.

1.4 - Positive-only q-functions

The problem of finding an assignment for $h(p)$, $p \in P$ that minimizes the cost while maintaining $Correct(X_{train}, h) = 1$ is greatly simplified if we consider *positive-only q-functions*. By *positive-only q-functions*, we mean that we may only force positive assignments of $h(p)$ (i.e. $h(p) = 1$) when attempting to force a q -function to be 1. In other words, we never assume that a particular $h(p)$ will be 0: we either assume it is 1 or it is a “don’t care”. This is equivalent to admitting that all the *bad neighborhood* patterns may be in the final classifier, and thus each class must minimize its own prototypes without assuming the other classes will do any minimization, just like when two warring factions rearrange their defenses without trusting their opponent will withdraw anything. When X_{train} and P are one and the same, as is many times the case, this is equivalent to finding the minimum *selective subset of P* (Ritter, Woodruff *et al.* 1975).

It is clear that the set of prototypes found using this approach may be larger than the absolute minimum set necessary to classify a given training set of patterns. We must however bare in mind that the goal of classification is to perform well in the unseen data, not the available training data. By using the positive-only approach, we are keeping the original class borders as untouched as possible, and thus we may be safeguarding that the error in the unseen does not increase. On the other hand, we are not allowing much smoothing of those borders, which in some circumstances may also have adverse effects.

When considering positive-only q-functions, $Correct(X_{train}, h)$ can only be 1 if $\forall x \ r_{\text{partial}}(h, x, 0) = 0$. If for any pattern x this is not the case, then that pattern cannot be correctly classified unless we guarantee that the patterns of its 0 order R-set are removed (which violates our positive-only assumption). Thus, we must ignore these patterns as errors. As we shall see later, some of these patterns may actually end up being correctly classified, for the prototypes in their 0 order R-set may not be chosen for the final classifier. The number of patterns for which the 0 order R-set is 0

will contribute to what we called the *a priori error rate*, which is the maximum error rate that the procedure might yield on the training set.

For the remaining patterns, the q -functions are greatly simplified. All \mathbf{R} -sets of order greater than 1 are necessarily non-empty, so if we cannot assume that we will remove any of their prototypes, the partial r -functions will have to be assumed 1, and thus all terms but the first of eq.77 must be assumed 0. The resulting q -function is thus

$$q(h, x) = q_{\text{partial}}(\mathbf{h}, \mathbf{x}, 0) = \sum_{\mathbf{p} \in Q_0(\mathbf{x})} h(\mathbf{p}) \quad (78)$$

Consequently the resulting correctness function $Correct(X_{\text{train}}, h)$ will be

$$Correct(X_{\text{train}}, h) = \prod_{i=1}^{|X_{\text{train}}|} q(x_i, h) = \prod_{i=1}^{|X_{\text{train}}|} \sum_{\mathbf{p} \in Q_0(x_i)} h(\mathbf{p}) \quad (79)$$

Finally this function is almost in a canonical form. In fact it is in the Conjunctive Normal Form (DNF) (Wegener 1987), which has been studied extensively. It also has the peculiarity that it contains only positive (or affirmative) literals, since there are no negations involved in this equation. To find the minimum cost classifier we must simply find the minimum number of assignments that will make the function 1, which is an equivalent definition of a minimum size prime implicant of the Boolean function (Wegener 1987). This exact problem is of great importance in a number of different fields, namely in Computer Aided Design (CAD) and fault diagnostic, so we shall briefly review the techniques developed in those fields. Unfortunately, it has been proved that in the most general case, finding a minimum size prime implicant is NP-complete (Eiter and Gottlob 1995).

1.4.1 - Known methods for finding prime size implicants

The simplest way to find the minimum size prime implicant is simply to factor out the terms of equation (79). Since the conjunction and disjunction operators of Boolean algebra are both distributive in relation to the other, a product of sums can be factored out to a sum of products:

$$Correct(X_{\text{train}}, h) = \prod_{i=1}^{|X_{\text{train}}|} \sum_{\mathbf{p} \in Q_0(x_i)} h(\mathbf{p}) = \sum \prod h(\mathbf{p}_k) \quad (80)$$

Any of the terms of the summation (a product of $h(\mathbf{p}_k)$) can make the function 1, so we need only assign the values 1 to the $h(\mathbf{p}_k)$ of the shortest of those terms. Unfortunately, that is easier said

than done, and the indexes of the summation and products in equation (80) were deliberately left unspecified, because they can only be found by actually factoring out the previously equation. Factoring out that equation has a complexity that is exponential on the number of patterns used. We wrote MATLAB code to perform this function, and when we later compare numerical results, we shall see that factoring out is only possible for rather small toy problems.

As with many NP-Complete problems, the branch-and-bound technique (Fukunaga 1990) can be used to find the exact solution in what may be reasonable time. We also wrote MATLAB code to implement this approach, but although it does improve dramatically the processing time, we will see that it is still only viable for small size problems.

The complexity of this problem is very well explored in (Wegener 1987), but significant improvements have been made since then in finding the most efficient ways possible of solving it. One of the most promising solutions is to use Integer Linear Programming (ILP) techniques. This approach was proposed in (Pizzuti 1996), and later improved by (Silva 1997). The proposed method is designed to solve general propositions in CNF, so one of the steps involves transforming it into a affirmative only propositions, substituting negated literals by a associated positive literal, and adding the restriction that only one may be true. This obviously is not necessary in our case, and so we are left with a relatively simple unbounded minimization problem to be solved by ILP.

1.4.2 - Algorithm for building positive-only Q-sets

It is possible to construct positive-only Q-sets for a training set, with a time complexity $O(n \times m)$ where n is the number of patterns in the training set, and m the number of prototypes. As for the memory requirements, they are between $O(n)$ and $O(n \times m)$, depending on the particular problem.

The algorithm is as follows:

Algorithm 9- Computing Positive -only Q-sets

```

Let
   $\mathbf{X}_{train}$  be the set of training patterns  $\mathbf{x}$ 
   $\mathbf{P}$  be the set of candidate prototypes  $\mathbf{p}$ 
   $\mathbf{Q}$  be a vector with the  $\mathbf{Q}$ -sets of each pattern, initialized to  $\emptyset$ 

1 Do
2 For  $i=1$  to  $|\mathbf{X}_{train}|$ 
3   For  $j=1$  to  $|\mathbf{P}|$ 
4     Calculate the similarity  $s(\mathbf{x}_i, \mathbf{p}_j)$ 
5     Find the largest value  $v$  of  $s(\mathbf{x}_i, \mathbf{p}_j)$ , for which the class  $\mathbf{p}_j$ 
      is different from that of  $\mathbf{x}_i$ .
6     For each  $s(\mathbf{x}_i, \mathbf{p}_j)$ , add index  $j$  to  $\mathbf{Q}(\mathbf{x}_i)$  if  $s(\mathbf{x}_i, \mathbf{p}_j) > v$ 

```

A MATLAB implementation of this algorithm is given in appendix, and was used extensively throughout the experimental part of this thesis.

The memory requirements for this algorithm are relatively modest and are mainly reduced to those necessary for the input and output data. The only internal variable that requires any considerable memory is a single vector of length $|\mathbf{P}|$, that has to store an index and a similarity value in each element. This requirement, linear in $|\mathbf{P}|$, is absorbed by requirements needed for the input and output. The input data requires $O(|\mathbf{X}_{train}|, |\mathbf{P}|)$. In the worst case, the output requirements are $O(|\mathbf{X}_{train}| \times |\mathbf{P}|)$ since each Q-set may have as many elements as there are prototypes. In practice, the Q-sets will be much smaller rendering the requirements closer to $O(|\mathbf{X}_{train}|)$. However, in some implementations sets are rather cumbersome to work with, and it is simpler to substitute them by Boolean vectors, where membership is represented by a logical 1 in the corresponding component of the vector. In those implementations the memory requirements are $O(|\mathbf{X}_{train}| \times |\mathbf{P}|)$, but since each value is Boolean, that cost is really quite small.

It is also important to stress that the similarity values (or distances), need not be kept, and they usually require far more space than a simple index or Boolean value.

The time requirements for the algorithm are $O(|\mathbf{X}_{train}| \times |\mathbf{P}|)$. In step 2, we repeat the procedure $|\mathbf{X}_{train}|$ times, and in step 3 $|\mathbf{P}|$ times, making it at least $O(|\mathbf{X}_{train}| \times |\mathbf{P}|)$. Steps 5 and 6 are both linear in $|\mathbf{P}|$, and thus asymptotically absorbed by the previous step.

1.4.3 - Heuristic Q-set algorithm for selecting prototypes

Since an optimum selection of the prototypes is equivalent to finding the minimum size prime implicant of a Boolean function, and that has been shown to be NP-Complete, we must resort to some sort of heuristic algorithm to obtain acceptable solutions in acceptable time. Most heuristic algorithms explore certain characteristics, so we developed our own heuristic, that is very similar to the one proposed, using a completely different formalization, by (Ritter, Woodruff *et al.* 1975), and very similar to the Davis-Putnam algorithm for refutation (Davis and Putnam 1960) that has been used previously for minimizing Boolean functions (Barth 1995). The main idea is that we must select the prototypes that are the only element of any Q-set, and after that, we should be greedy and choose the prototype that classifies correctly more patterns. The algorithm can be presented as follows:

Algorithm 10 - Qset Heuristic for selecting prototypes

```

Let
  Q      be a vector length i with the Q-sets of each pattern  $x_i$ 
  P      be the set of indexes of candidate prototypes  $p$ , appearing in
         Q
  Psel  be the set of indexes of the selected prototypes

1 Do
2 Let Psel =  $\emptyset$ 
3 Find all Q(i) that have a single element, and add that element to
   Psel, remove it from P, and remove those  $i^{th}$  components from
   vector Q
4 For all remaining components of Q, remove them if they have any
   element that is also in Psel
5 While there are any components remaining in Q do
6   For all elements of P, calculate how many times they appear
   in Q
7   Find the most occurring element, add it to Psel, and remove it
   from P
8   For all remaining components of Q, remove them if they have
   any element that is also in Psel

```

The algorithms memory requirements are basically that of its inputs, and thus are not a problem. As for the time complexity it is $O(|P| \times |Q|^2)$. In step 3, all components of $Q(i)$ have to be searched, making it $O(|Q|)$. In step 4, for each component of Q all P may have to be searched, making it $O(|P| \times |Q|)$. Step 5 will iterate at most $\#Q$ times the next steps, of which steps 6 and 8 must search at most $|P| \times |Q|$ possibilities, rendering the algorithm $O(|P| \times |Q|^2)$.

The first part of this algorithm (selecting the prototypes that are the single element of a Q-set) is inevitable, and thus, at least in some sense optimal. The second part, where the most occurring prototype is selected is clearly non-optimal, and many different search strategies can be used.

It must also be noted that since this procedure generates selective subsets, there is no interaction between the choice of prototypes for each class. Therefore, this last step of selecting prototypes can be done independently for each class, thus reducing considerably the complexity of the task.

1.4.4 - Other selection techniques

One of most interesting approaches for selection of prototypes is to consider prototypes as features of the Q-set, and use feature selection techniques to choose the best prototypes. In this type of approach, each Q-set is seen as an *object*, which has a certain number of features: the prototypes.

As was reviewed in part I of this thesis, Rough Set Theory (Pawlak and Slowinski 1994) provides us with tools to make an optimal choice of features. Moreover, for this application, it has the advantage over other feature selection techniques that it was developed for categorical data, as is the case with prototypes that are either are part of a given Q-set or not.

Rough Sets have also another advantage in this case, since they provide not only reducts with minimum sets of classifier prototypes, but also cores, which will contain the prototypes that appear in all minimum sets of classifiers.

1.5 – General case

When considering the general case, i.e., the case in which may exclude certain patterns from the final classifier, we will be able to construct a classifier with less prototypes than the one obtained with positive-only Q-sets. However the computational cost of constructing it will be considerably greater.

Let us look again at the correctness function in this case:

$$Correct(\mathbf{X}_{train}, h) = \prod_{j=1}^{|\mathbf{X}_{train}|} q(\mathbf{x}_j, h) = \prod_{j=1}^{|\mathbf{X}_{train}|} \sum_{n=0}^{|\mathcal{Q}(\mathbf{x})|} \left(\prod_{i=0}^n \bar{r}_{partial}(h, \mathbf{x}_j, i) \right) \cdot q_{partial}(h, \mathbf{x}_j, n) \quad (81)$$

Both r and q -functions are disjunctions (or Boolean sums) of literals, but as the r -functions are negated, due to DeMorgans laws the result will be a conjunction (or Boolean product) of negated literals. Thus the product of r -functions will be a single product of negated literals. Due to the presence of the final q -functions, the result will be a 4 level Boolean formula (Wegener 1987), with products of sums of products of sums.

The first remark that must be made is that there may be no assignment of logical values to the literals $h(\mathbf{p})$ that make the function 1. In fact this is the classical Satisfiability (SAT) problem (Cook 1971; Cook 1983): Given a generic Boolean formula, find an assignment for its variables that makes its logical value 1. This problem was proved to be NP-complete, and a great deal of effort has been put to solving it with a P-complexity algorithm (Baase and Gelder 2000), for that would revolutionize many fields of science.

The fact that there may be no solution means, from the classification point of view, that it is impossible to classify correctly the given set of patterns with any combination of the available prototypes. To obtain a classifier we must therefore relax our constraints and assume that there may be errors. It may be quite difficult to pinpoint the exact patterns that are causing problems, but it certainly is one that has a non-empty 0 order \mathbf{R} -set ($\mathbf{R}_0(\mathbf{x}) \neq \emptyset$). As we saw when discussing the positive-only \mathcal{Q} -sets, if the first \mathbf{R} -set is empty, then any prototype of the 0 order \mathcal{Q} -set will correctly classify the given pattern. Thus, removing all patterns for which $\mathbf{R}_0(\mathbf{x}) \neq \emptyset$ guarantees that an assignment is possible that makes $Correct(\mathbf{X}_{train}, h) = 1$.

Unlike the positive-only case, when dealing with the general case we need to find the minimum number of positive assignments that make the function 1. This is no longer equivalent to finding the minimum size prime implicant, since there may be another prime implicant that, although having more literals, has fewer of them in the affirmative form. Nonetheless, it is still useful to find the prime implicants of the function, since then we can search only these to find the one with less affirmative literals.

A number of different techniques have been developed to work with binary functions with a large number of literals. One of the most successful, stems from graph theory and is known as BDD –

Binary Decision Diagrams (Bryant 1986). Most of the techniques involving BDD are concerned with solving the SAT problem, but many of them have the specific aim of finding prime implicants. One such technique for computing prime implicants of multi-level Boolean functions is presented in (Stoffel, Kunz *et al.* 1997).

1.5.1 – A Heuristic for the general case – G2P

One of the main reasons why a heuristic for the general case is not trivial is there is a lot of interaction between the different classes, i.e., a choice of one prototype for one class will have a big impact on the available choices for other classes. Such an interaction, as has been pointed out does not exist in positive-only approach. One possible solution is to try and de-couple the classes once again, through a certain pre-processing, which we have called G2P – General to Positive-only.

The G2P algorithm relies predetermining an *acceptable maximum error rate* (AMER) on the training set, and then assigning a *cost/benefit* (CB) ratio to each of the prototypes that we are considering for exclusion. As we exclude prototypes, it is possible that the 0 order \mathbf{R} -set of some patterns will cease to be empty, and thus that pattern would be (a priori) incorrectly classified by the positive-only approach. When this happens, we will assume that the number of errors has increased by 1, and will check to see if the AMER (*acceptable maximum error rate*) has been reached, and thus we should stop attempting to exclude prototypes. The CB (*cost/benefit*) ratio will decide which prototype we will exclude next. We consider that there is a cost of 1 if the removal of a prototype makes pattern loose all its 0 order Q-sets, thus turning the 1st order R-Set into the 0 order R-set, and producing a *a priori* error for the positive-only approach. We consider that there is a benefit of n , when the exclusion of a prototype will increase by n the size of the 0 order Q-set of a patterns (thus giving more possibilities when choosing a prototype for that pattern). We will remove the prototype with smallest CB, calculate the a priori error rate, and proceed to the next removal until the AMER is reached.

To apply the G2P algorithm we must now compute the complete Q-sets⁸ of the patterns, apply the algorithm, and then feed the resulting q-functions to the prime implicant finding procedure (possibly the Q-set Heuristic). Let us then see how to perform the first 2 steps.

1.5.1.1 - Computing the complete q-functions

The algorithm for computing the complete q-functions may be described as follows:

Algorithm 11 - Computing the complete Q-sets

```

Let
   $\mathbf{X}_{train}$  be the set of training patterns  $\mathbf{x}$ 
   $\mathbf{P}$  be the set of candidate prototypes  $\mathbf{p}$ 
   $\mathbf{Q}$  be a vector with the complete Q-sets of each pattern
   $\mathbf{Q}_{ival}$  be a companion vector to  $\mathbf{Q}$  with the logical values

1 Do
2 For  $i=1$  to  $|\mathbf{X}_{train}|$ 
3   For  $j=1$  to  $|\mathbf{P}|$ 
4     Calculate the similarity  $s(\mathbf{x}_i, \mathbf{p}_j)$ 
5     Let  $\mathbf{Q}(i)$  be the list of prototypes ordered by increasing
      values of  $s(\mathbf{x}_i, \mathbf{p}_j)$ .

```

The algorithm is quite similar to that of computing the positive only Q-sets, but for each pattern, we must now order prototypes by decreasing similarity. This increases the time complexity to $O(n \times m \log m)$ where n is the number of patterns in the training set, and m the number of prototypes. The complete Q-sets will also have a fixed length m , thus fixing the memory requirements for the output at $O(n \times m)$.

The associated general case q-function can easily be obtained from the given complete Q-sets by observing the following rules:

- a) Insert a OR (+) operator after any prototype that has the same class as the pattern.
- b) Insert a AND (\bullet) operator after any prototype that has a different class to that of the pattern.

⁸ The term *complete Q-sets* refers to the ensemble of all Q-sets and R-sets of a given patterns, and is a easier way to call them.

- c) Open a parenthesis whenever a prototype of a different class is followed by one that has the same class of the pattern.
- d) Close all parenthesis at the end of the expression.

1.5.1.1 – Applying G2P

We must now apply the described procedure G2P, which can be summarized in the following algorithm:

Algorithm 12 - G2P - General to Positive

```

Let
  Q      be a vector with the Q-sets of each pattern  $x_i$ 
  P      be the set of indexes of candidate prototypes  $p$ , appearing in
         Q
  Pcand be the set of indexes of the candidate prototypes for
         exclusion
  Pexcl be the set of indexes of the excluded prototypes
  AMER   be the acceptable maximum error rate

1  Do
2  Let Pexcl =  $\emptyset$ 
3  Repeat
4      Save the Q to Qold so as to be able to backtrack later
5      Let Pcand =  $\emptyset$ 
6      For i=1 to #Q
7          Find the first  $p$  of Q(i) that has a different class to  $x_i$ 
          and add it to Pcand if the next  $p$  on the list has the same
          class as  $x_i$ 

8          For all  $p$  in Pcand
9              Let cost( $p$ )=1 and benefit( $p$ )=1
10             For i=1 to |Q|
11                 If  $p$  has the same class as  $x_i$  then
12                     If  $p$  is the first prototype and the next one
                       is of the wrong class add 1 to cost( $p$ )
13                 else
14                     If  $p$  is the first prototype of the wrong
                       class, count the number of prototypes of
                       the right class that immediately follow
                       it, and add them to benefit( $p$ )
15             Select the  $p$  with smallest ratio cost( $p$ )/benefit( $p$ ), add it
               to Psel, remove it from P and from all Q(i)
16             Calculate the present a priori error rate apriori_error
17 Until apriori_error > AMER
18 Restore Qold to Q

```

It may seem strange that steps 4 and 18 are introduced to actually go one iteration further than what is apparently necessary. However, it must be noted that it is possible to remove prototypes

without any impact on the error rate in the training set, and thus, even after we have reached the maximum error allowed, it is still worth trying to remove more prototypes.

Steps 6-7 also provide considerable speedups, since the search for a prototype to remove will occur only within those that will certainly give benefits.

Although this procedure does require considerable computing power, its complexity is still quite acceptable. Each iteration is less than $O(|Q|, |P|)$.

1.6 - Comparison with other methods

Despite its elegance, the practical value of the Q-set theory can only be appreciated when compared with other prototype minimization techniques, in practical applications. The vast amount of minimization techniques, some with many parameters that have to be fine tuned for any given application, makes an extensive comparison out of the scope of this thesis. We will therefore limit ourselves to a comparison with the most standard and reliable methods, using widely known benchmark data sets. From amongst these, we chose the Hart's double F problem (already discussed in part I), and the classical Iris Dataset. Both these problems are difficult to solve using exact minimization techniques, so we added a very simple "straight line" problem (explained later), to compare Q-sets with exact minimization and the other techniques.

As stated in Part I, the CNN-Condensed Nearest Neighbor (Hart 1968), together with the RNN-Reduced Nearest Neighbor, are the two fastest and simplest prototype minimization techniques, yet they have proved to compare very favorably with most other techniques. Like Q-sets, they only select prototypes amongst a given set, as opposed to other methods that generate new ones. We shall therefore use CNN and RNN as benchmarks for comparisons with Q-sets. It must be noted that, unlike these two methods, the Q-set method does not rely on order, and thus we can expect lesser variance in the results.

1.6.1 – The double F problem

This problem, initially proposed by (Hart 1968) and already presented in Part I, consists of two classes of bi-dimensional patterns with a uniform distribution in two interlocked F shapes, as seen in Figure 30. The two classes lie in the 22.5 x 20 rectangle with the bottom left corner at the origin (0,0), and have boundaries defined by the line that joins (7.5,0), (7.5,5), (15,5), (15,10), (7.5,10), (7.5,15), (15,15), (15,20). In most papers, 200 patterns of each class are used. We generated a “reference set” with 200 patterns of each class, that

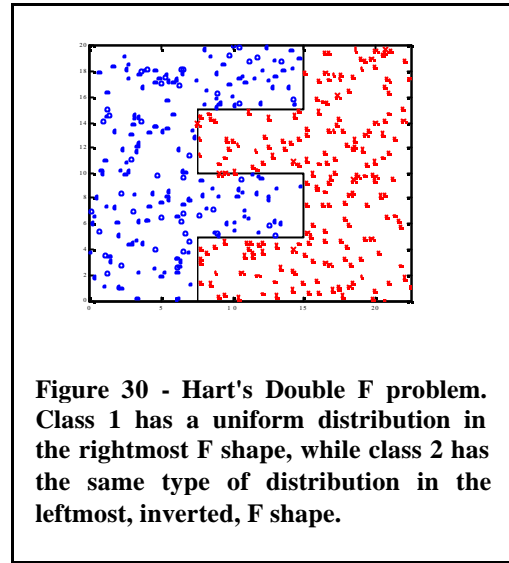


Figure 30 - Hart's Double F problem. Class 1 has a uniform distribution in the rightmost F shape, while class 2 has the same type of distribution in the leftmost, inverted, F shape.

was used in part I, but for comparisons, we generate multiple random datasets with the double F distribution, using different numbers of prototypes. Since the true borders between the classes are known, the generalization error of a given classifier set of prototypes may be calculated exactly. The generalization error is simply the area between the true boundaries and the Voronoi boundaries defined by the selected prototypes (divided by the total area if an error rate is desired). However, the direct computation of this error is very time consuming. Therefore, we use a Monte Carlo method that consists of using a test set of 100.000 patterns of both classes, and classifying them with the selected prototypes. Besides obtaining an estimate of the generalization error, we also obtain an estimate of the time each classifier takes to classify a large dataset. Given the number of test patterns available, and the estimated error probabilities, the variance of the estimate for each of the classifiers is always less than 0.1%.

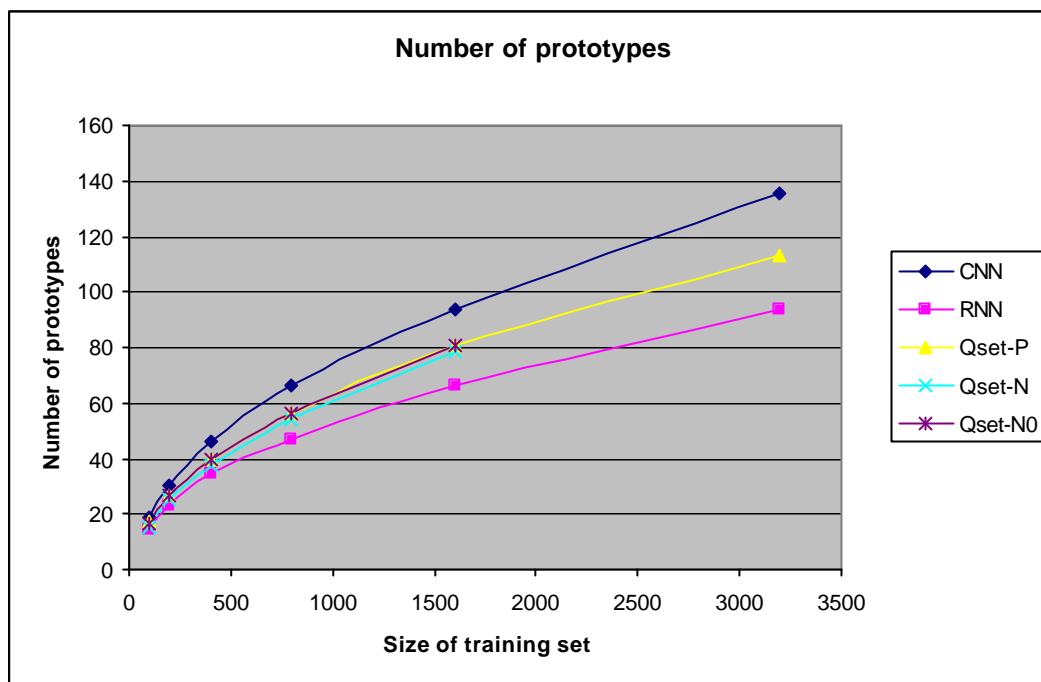


Figure 31 - Number of prototypes used for Hart's double F problem.

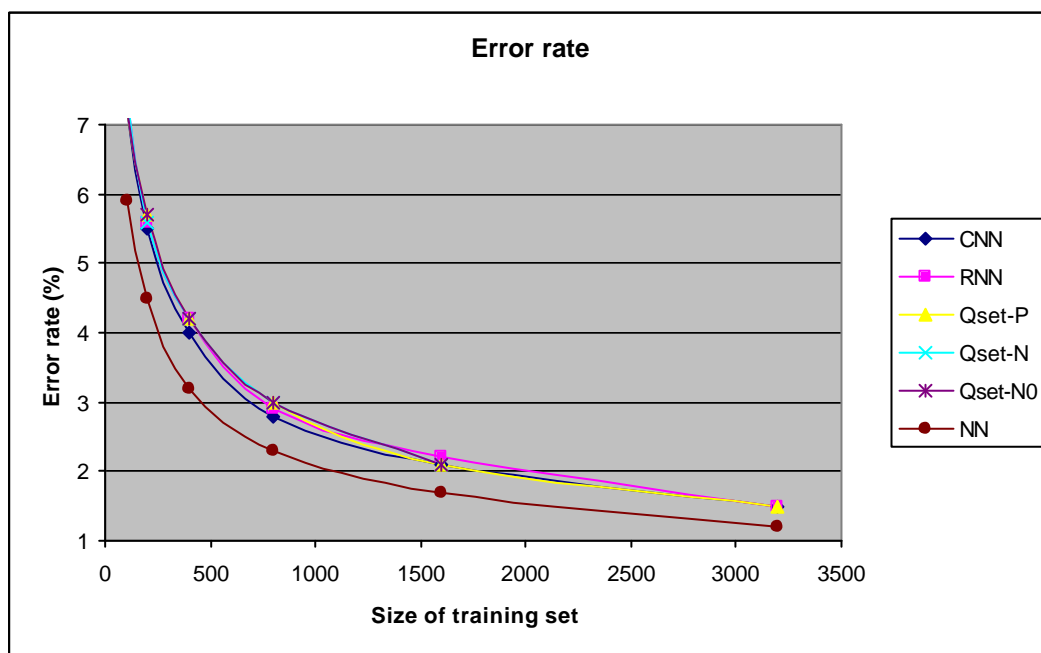


Figure 32 - Error rate for Hart's double F problem.

We compared the basic nearest neighbor (NN) rule with CNN, RNN, Q-sets with positive-only heuristic (Qset-P), Q-sets with negations (general case) using the G2P heuristic with one admissible error (Qset-N), and the same algorithm using zero admissible errors (Qset-N₀). We started by using a total of 100 training patterns (50 of each class), and steadily increased that number to 3200 training patterns. In each case, we repeated the experiments 30 times, using

different randomly generated datasets. The complete results are presented in Appendix A, and summarized in Figure 31, Figure 32, and Figure 33.

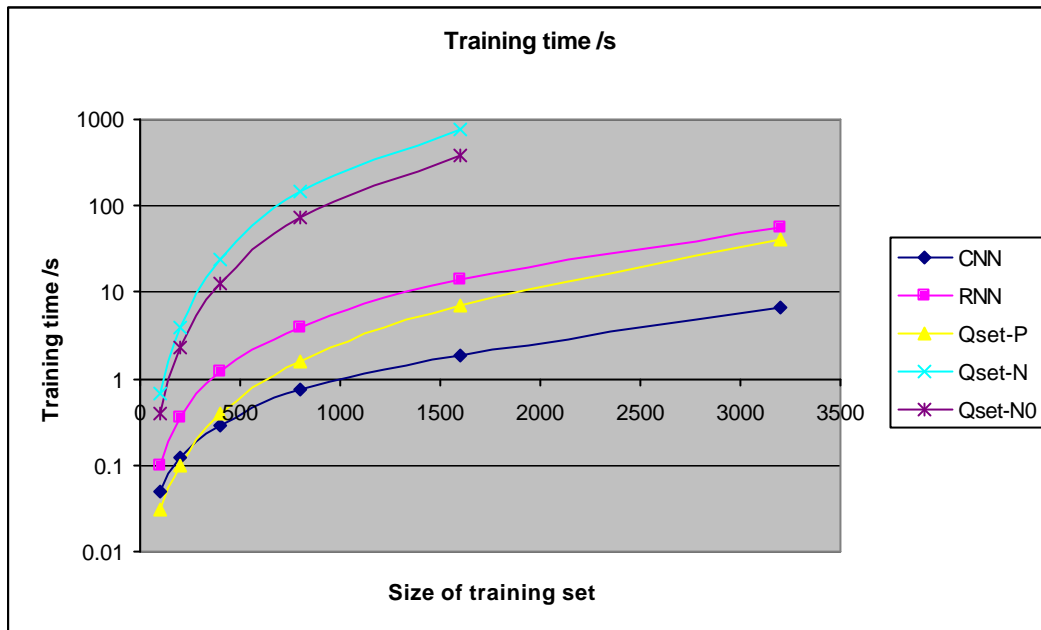


Figure 33 - Training time required for Hart's double F problem.

It was observed that, for this problem, the Q-Set positive-only heuristic's performance was consistently between that of CNN and RNN. The number of prototypes was always less than that obtained with CNN, but larger than that obtained by RNN. It is only natural that the number of prototypes be larger than RNN, for while RNN will produce a consistent subset, positive only Q-Sets will yield selective subsets, that are closer to the original Voronoi boundaries. Accordingly, the error rate obtained with positive-only Q-Sets is consistently smaller than that obtained with RNN. It is higher than that obtained with CNN, although the error rates all converge as they decrease exponentially when the training set becomes very large. The training time required for positive-only Q-sets also lies between the very short training time required for CNN and the time required for RNN.

As expected, the general case Q-set heuristics require a great deal of training time, and provide only modest improvements. The error rate was very close to that obtained with positive-only Q-sets, with the error rate sometimes increasing slightly when 1 error was allowed during training. The number of prototypes obtained was higher than that obtained with RNN, but less than that obtained with positive-only Q-Sets. It must be said that although the training time required for general case Q-Sets heuristics will always be considerable higher than that for the positive-only

case, our implementation of the G2P procedure is very inefficient. Matlab is notoriously inefficient when “for” cycles are required, as is the case with G2P. Thus, a C/C++ implementation would decrease the relative difference in training time between these methods and the others. The same argument also applies to the differences between the positive only Q-Set and the remaining methods, since the Matlab implementation uses 8 byte floating variables to store binary values, and varying size matrices to store sets. A lower level implementation, as mentioned before, will greatly increase the performance of Q-Set heuristics. Nevertheless, the asymptotic behavior is the same, no matter which implementation is used.

1.6.2 – The iris dataset

The Iris dataset is probably the best known benchmark dataset in the world. It was originally proposed as a classification problem by Fisher (Fisher 1936), based on data collected by E. Anderson. We were not able to find Anderson’s original article, published in 1935 in the Bulletin of the American Iris Society, volume 59, pages 2-5, titled “The Irises of the Gaspe peninsula”. The dataset contains measurements of 4 different characteristics (sepal length, sepal width, petal length, and petal width) of 150 different irises. The iris is a plant that has 3 different species, namely Iris Setosa, Iris Versicolor, and Iris Virginica. The dataset includes 50 samples of each species.

This dataset is available at the Machine Learning Repository at the University of California at Irvine, but as pointed out in (Bezdek, Keller *et al.* 1999) some errors have crept into the datasets that have been circulating amongst researchers. In this thesis, we use the original dataset.

Applying the Condensed Nearest Neighbor (CNN) algorithm to the Iris dataset, just as it is presented in the original paper, produces 24 prototypes. Reduced Nearest Neighbors will bring the number of prototypes down to 16. The Q-set positive-only heuristic will produce 17 prototypes, while the general case Q-set heuristic, allowing no errors, will produce only 15.

We also performed leave-one-out validation on the Iris dataset. The results are presented in Table 7. Since for classification a single pattern is used each time, the processing time is too small to be measured reliably, and thus is not included in the results. Similarly, for each test there is either one or no errors, so an analysis of variance is pointless, and the total number of error obtained in the 150 trials is shown in parenthesis. Due to the particularly good results obtained using the

general case Q-Set heuristic with one error allowed, we also included the results obtained when allowing 2 errors (QSet-N₂).

Method	N° Prototypes	Error rate	Training time / s
NN	149.0 ± 0.0	4.0 (6)	0
CNN	23.7 ± 1.5	8.0 (12)	0.14 ± 0.03
RNN	15.9 ± 0.5	8.0 (12)	0.26 ± 0.04
QSet-P	16.9 ± 0.4	8.0 (12)	0.06 ± 0.02
QSet-N	13.9 ± 0.5	7.3 (11)	2.06 ± 0.21
QSet-N ₀	14.9 ± 0.4	6.7 (10)	1.11 ± 0.09
QSet-N ₂	12.0 ± 0.4	8.0 (12)	2.47 ± 0.15

Table 7 - Leave-one-out cross-validation for the Iris Dataset. Together with the error rate, the actual number of errors is shown in parenthesis

As can be seen in Table 7, the positive only Q-Set heuristic continues to have a performance between that of CNN and RNN. The number of prototypes obtained by this heuristic is much less than that of CNN, and only slightly more than that obtained by RNN. Since the dataset is quite small, the error rate for all three methods is the same. The most revealing result when comparing these methods, is that the Q-set approach takes considerably less time to train. This happens because while Q-Set techniques search a Boolean space, CNN and RNN perform that search in the original space, that is now 4-dimensional. As the number of dimensions grows, the evaluation of distances will be ever greater, and that difference will increase. Even if all distances are computed beforehand and stored in memory, the Q-Set approach will still be faster for it only needs to compare binary values, as opposed to the possibly real-valued distance measurements required for CNN and RNN.

More important still, the general case Q-Set heuristics, both allowing errors and not, achieve the lowest number of prototypes, and maintain or actually improve the error rate. Unfortunately, that is still obtained with a very high training time.

The reasons why Q-Set approaches perform so much better than in the case of Hart's double F problem can be traced to two main factors. The first, already mentioned, has to do with the higher

dimensionality of this problem. The second, is that while in Hart's problem there is no margin between the two classes (patterns of one class may be arbitrarily close to patterns of another), in this case there is a fair distance between patterns of different classes. Thus, small shifts in the Voronoi boundaries generated by the prototypes will not have a significant impact in the error rate.

1.6.3 – The straight line problem

In the above mentioned problems we were not able to compare the results with those obtained using Q-Sets with an exact minimization technique. While there are techniques and software for finding exactly the smallest prime implicant in problems with many variables, it was not practical to include them in our comparisons. Therefore, we wrote a Matlab routine using branch-and-bound to perform that search. This routine could easily find the smallest prime implicants in problems with up to 40 variables (or prototypes). The branch-and-bound technique is extremely sensitive to the starting point, and thus the time required to find a solution has very high variance when the technique is applied to a series of similar problems. Thus, while sometimes we could obtain results using up to 58 variables in just a few minutes, other times we had to abort the process after running it for 24 hours.

These limitations led us to compare Q-Sets with exact minimization and other methods only for a very simple problem, that we called the straight line problem. In this problem, we generate 2-dimensional patterns with uniform distribution in the unit square limited by (0,0) and (1,1). Those that lie in the left side of that square (i.e., with $x < 0.5$) are considered to belong to class 1, and the others to class 2.

We forced the number of training patterns of each class to be equal, and performed a number of trials, increasing the total number of training patterns from 4 up to 40. The complete results are presented in Appendix B, and summarized in Figure 34, Figure 35, and Figure 36.

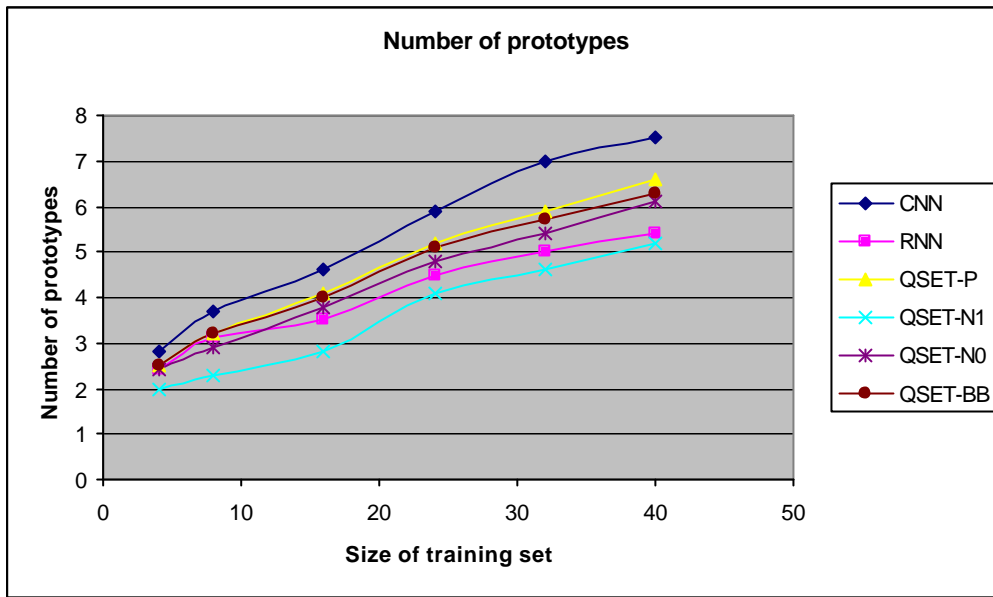


Figure 34 - Number of prototypes used for the straight line problem

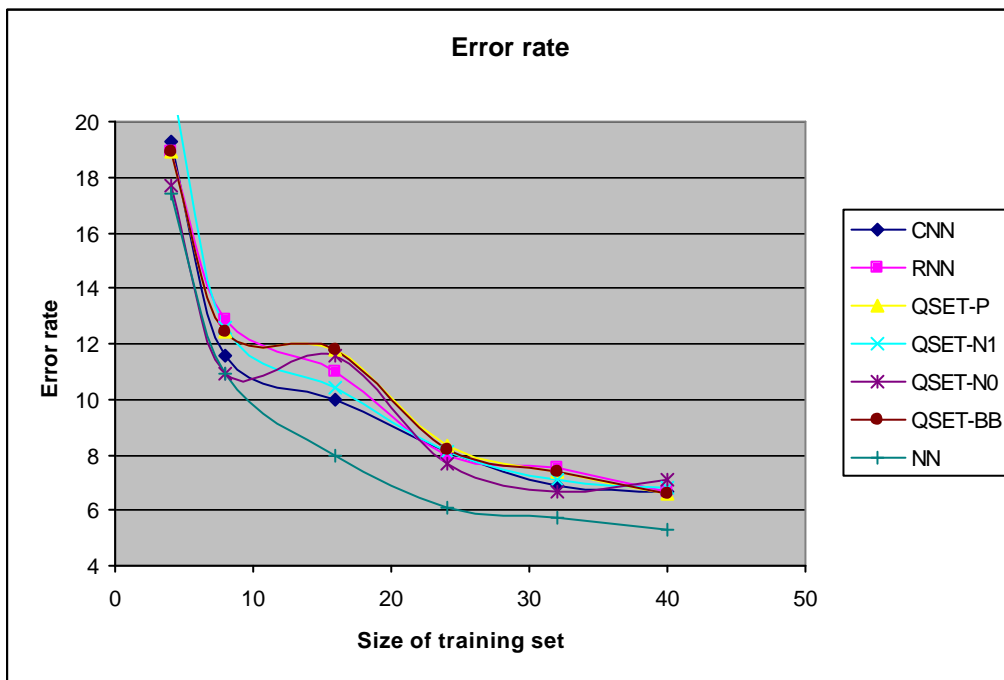


Figure 35 - Error rate for the straight line problem

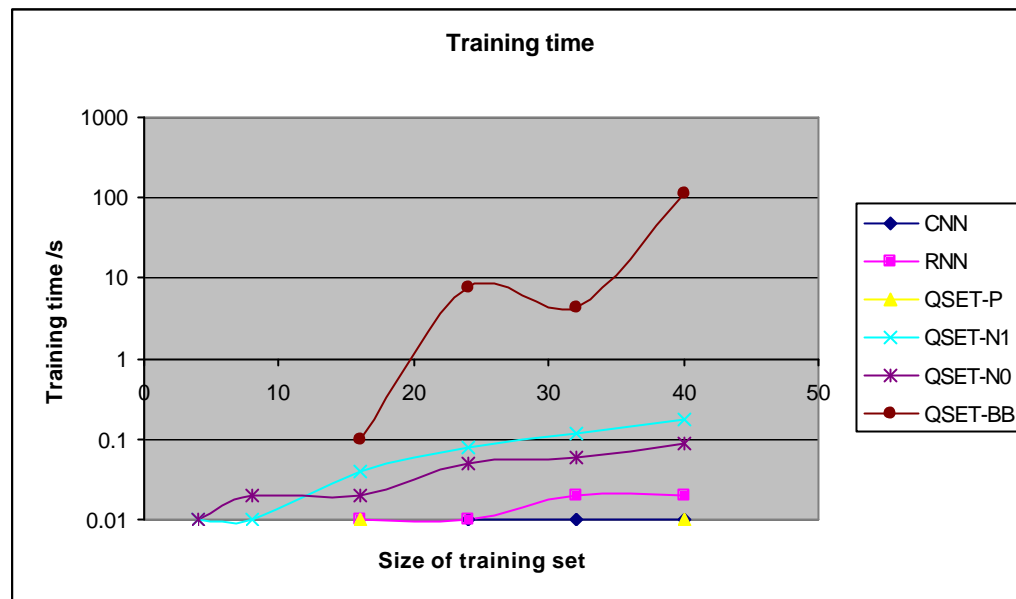


Figure 36 - Training times for the straight line problem. Note that the time axis uses a logarithmic scale to be able to show very different training times. Due to this, when the training time is too close to zero (as is the case for QSET-BB, CNN, RNN and QSET-P when the training set has fewer than 16 patterns), the value is not represented in this graph.

For this very simple problem, it is clear that the positive-only Q-set heuristic solution is very close to the true minimal selective subset. Therefore, the enormous amount of time required to find the optimal solution is defiantly not worthwhile. This small problem also shows that RNN will generally produce prototypes that do not constitute a selective subset.

1.7 – Extensions and applications of Q-set theory

We shall now overview some ways in which Q-sets may be used, and point to ways in which they can be extended.

1.7.1 - Choice of candidate prototypes and training sets

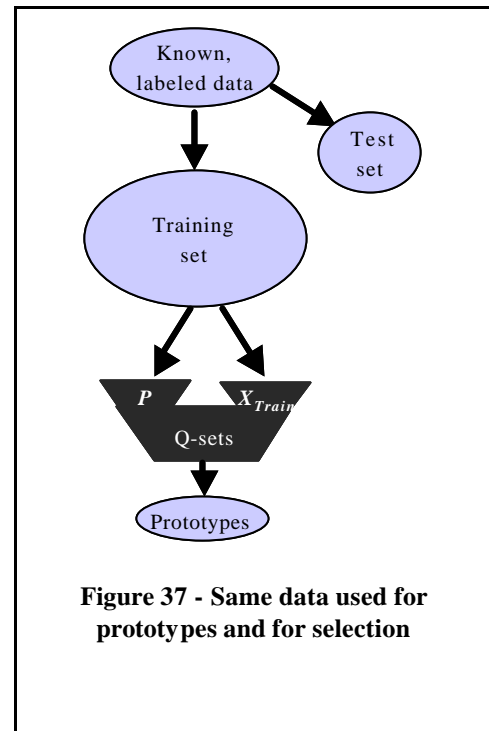
To apply Q-set minimization we need to have a set of candidate prototypes, and a set of training patterns. In many cases, such as the comparisons in section 1.6, we assumed that they were the same. There are however many other possibilities, so let us see what are the consequences of each choice.

1.7.1.1- $X_{Train} = P =$ Available known data

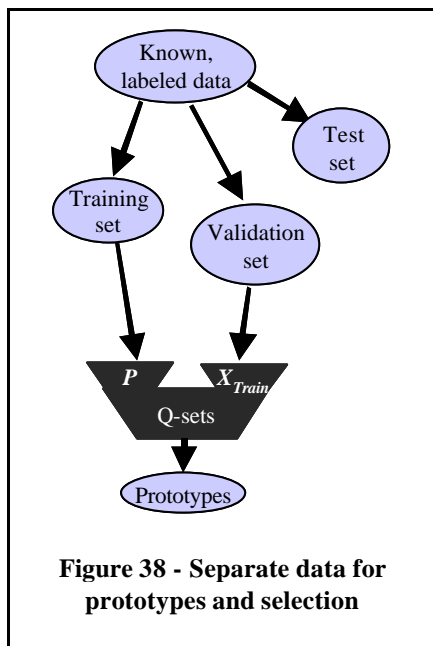
A common approach is to take the whole original training data and use it both as candidate prototypes and training data. A good reason for considering this approach is that it maximizes the use of all available information. Moreover, we will always have a solution for the general Q -set case, because we can guarantee that $R_0(x) = \{x\}, \forall x$. This happens because each pattern is closest to itself than any other, and has the same label as itself.

From a computational point of view, this approach has the inconvenient that it is very costly. The heuristic positive-only Q -set algorithm is quite fast, and can be used for very large data sets, but is nonetheless $O(n^2)$. However, the optimal approach, and the general case approach may not be viable when the data set is large.

From a classification point of view, considering the same patterns as prototypes and training set has a major drawback, that is common to the original nearest neighbor classifier: it is very sensitive to outliers. An outlier will not be removed from the prototype set, because there is a training set pattern (the prototype itself), that will be incorrectly classified if the prototype is removed.



1.7.1.2 - Available known data divided into disjoint X_{Train} and P



From a purely classification point of view, we should never use twice the same data when designing a classifier, so we should divide the available data into two separate sets: one to be used as the P prototypes, and the other to be used as X_{Train} patterns. Providing there is enough data, this should be the preferred approach.

It must be noted that, in coherence with the nomenclature used in Part I, the P set is what was referred to as *Training Set*, since it is used to actively provide the classifying prototypes, and what in this chapter we called X_{Train} is closer to the concept of *Validation Set*, since it is not actively used to construct the

prototypes, but just to select them. We will, however, keep the notation that we have used in this chapter, and continue calling X_{Train} the *Training set*.

By having disjoint sets of prototypes and selection patterns, we may have patterns in X_{Train} that cannot be correctly classified, and must thus be ignored. In the positive-only approach this is done automatically, but when considering the general case, care must be taken, since the correctness function may or may not be satisfiable.

Although the error rate in X_{Train} may be higher, this approach will provide a better generalization, since outliers and very rugged borders between classes will be smoothed out.

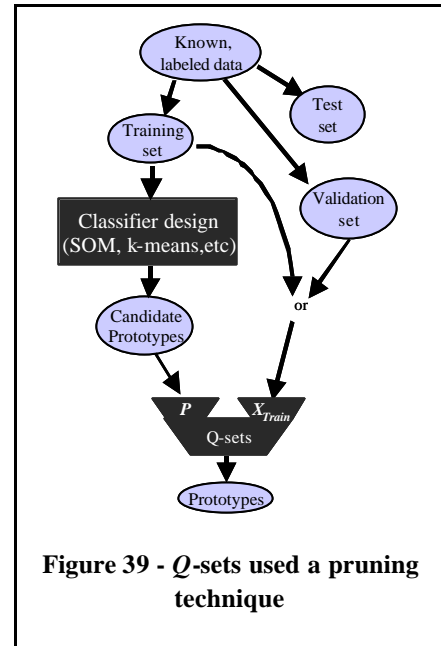
1.7.1.3 - P provided by another classifier design technique

There are many prototype-based classifiers that do not provide even locally minimum consistent subsets, and all of these can be pruned by the Q -set technique. The idea is to use the prototypes provided by those methods as candidate prototypes, and then use the original training data, or another set of validation data as X_{train} to select the best prototypes.

Techniques such as k-means (Duda, Hart *et al.* 2001) , Probabilistic Neural Networks (Specht 1990), Radial Basis Function Networks (Powell 1987) (Broomhead and Lowe 1988; Poggio and Girosi 1990)

, and Self-Organizing Maps (Kohonen 2001), require that the used specify a a-priori number of centroids, kernels, or neurons. Too few of these will result in a severe degradation in performance of the classifier, and so the number of prototypes is usually greatly over-dimensioned. A pruning algorithm may then be used, and the Q -set technique performs that job rather well.

Using some clustering technique before applying Q -sets has the additional advantage that it removes outliers and smoothens the data distribution, thus eliminating, or at least reducing the sensitivity to outliers.

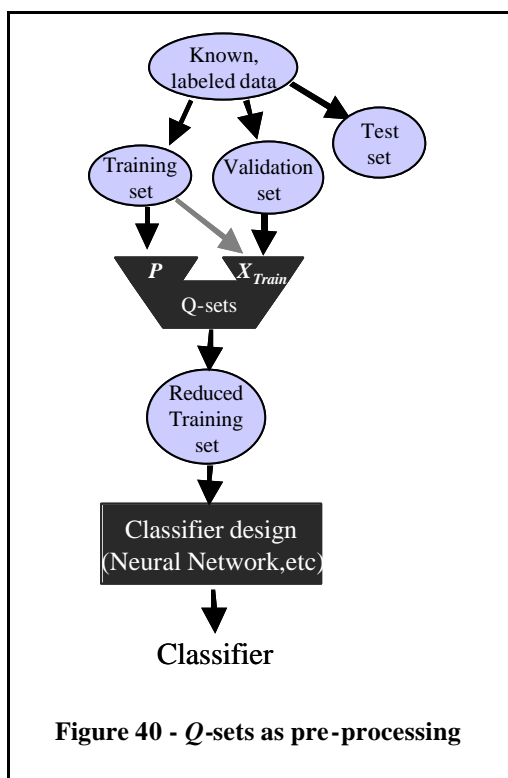


We find the use of Self-Organizing Maps (SOM) particularly useful for this task, as we have shown in (Lobo, Swiniarski *et al.* 1998). As mentioned in Part II, a thorough and general theoretical description of the behavior the SOM is difficult, but it is generally accepted that there is a *magnification factor*, usually estimated at $d/(d+2)$, d being the dimensionality of the problem. This *magnification factor* means that a SOM will represent sparsely populated regions of the input space with greater detail than the densely populated ones. This is particularly suited to Q -sets, since SOM pre-processing under-represents the densely populated areas where the choice between basically equivalent prototypes would take a long time, it smooths the borders between classes, removing the outliers, and it still has a good representation of those borders. Moreover, as argued in (Lobo, Swiniarski *et al.* 1998), the SOM algorithm can effectively be applied to very high dimensional data, or very large data sets, without requiring unreasonably high computational resources, and without having the numerical stability problems that can affect other techniques. As is normal when dealing with a supervised learning situation, we may still use the LVQ (Kohonen 2001) algorithm on the SOM before applying the Q -sets, so as to further separate the prototypes of different classes.

One particularly simple pre-processing step before applying Q -sets is sample the original data to select the candidate prototypes. If done correctly, we will be left with a set of prototypes that have a distribution proportional to that of the original data, and from which it will be much simpler to select the best ones.

One final note on using Q -sets as a pruning algorithm must be made: some techniques already provide at least locally minimum prototype sets, and thus it is no use to try and prune them. Such is the case, for example of Reduced Nearest Neighbors (Gates 1972), or Selective Nearest Neighbors if we use positive-only Q -sets.

1.7.1.4 - Q -sets used as pre-processing



Besides being used as post-pruning method, Q -sets can also be used as a pre-pruning method, that is it may be used to select only a few data patterns that will later be used to design another classifier. While not directly concerned with the way we select the data (any of the above techniques can be used) we chose to include this section here since it is tightly related to data set manipulation.

It could be argued that if Q -sets provide us with a classifier, that from a certain point of view is optimal, it would not make sense to use its output to design another classifier. However, we may not want to use a prototype-based classifier, and prefer some other type, such as a discriminant function, or a Multi-Layer Perceptron neural

network. These other methods may have distinct advantages for certain applications, such as providing an easily understandable distinction between classes, or a smoother border between them. However, training these classifiers on the original, large, data set may have inconveniences, such as require a lot of computing power. Therefore, some sort of sampling of the training data may be necessary. If we use the positive-only approach, the selected prototypes will be close to the border, and thus be good patterns to train the classifier.

Sampling the training set with a prototype-based classifier design technique has been used by (Plutowski, Cottrell *et al.* 1996; Mitiche and Lebidoff 2001; Choi and Rockett 2002), and shown to be effective in reducing the computational effort of training neural networks.

The idea of using a condensing algorithm as a pre-processing step for initializing another algorithm has also been used before, as for example in (Kim, Lee *et al.* 1993), where Tomek's (Tomek 1976) rule for CNN is used to initialize a LVQ network.

1.7.2 – Extensions to fuzzy Q-sets

A lot of effort has been made in using fuzzy set theory in classifier design (Bezdek, Keller *et al.* 1999). There are many reasons to do so, but one of the most important is that these classifiers provide, in a very natural fashion, a degree of certainty to the given result. It is thus appropriate that we point ways in which fuzzy set theory may be integrated with the Q-set approach.

We feel that the best way to introduce fuzzy sets in this framework is to basically leave it out until the final classifier is produced, and only then assign a membership function to each of the final prototypes, based on its distance to the nearest prototype of the opposite class. This keeps the process of selecting the prototypes fast and efficient, and introduces the fuzziness only where it is important for a correct interpretation of the results. One can, however, note that this is not a true integration of fuzzy set theory into Q-sets.

An interesting way to use fuzzy set theory is to consider that the Q-sets are not a crisp set of prototypes, but a fuzzy set, where the membership is a function of the similarity between the patterns and each prototype. We may even consider that the R-sets have a membership of 0, and the order of the partial Q-set imposes a damping factor on the membership (alternatively, the size of the preceding R-sets may do the same). The q -functions generated would be functions not of $h(\mathbf{p})$, but of $h(\mathbf{p}) \cdot \text{Membership to } Q\text{-set}$. From that point onwards, we may substitute the Boolean sum and product by the fuzzy equivalent.

After we do that, we will be left with an expression for the correctness function that does not necessarily compute to 0 or 1, irrespective of the assignments we make to each of the indicator functions. Instead of having the problem of minimizing the classifier cost subject to the restraint that the correctness must be 1, we will have a multi-objective maximization problem, for we will want to minimize the cost and maximize the correctness.

This last approach has several advantages very dear to the fuzzy set community, namely it propagates the “goodness” of a prototype as a classifier throughout the processing, and allows for a graceful degradation in performance as we force a smaller set of final prototypes.

It does however have the very big disadvantage that it requires far more computing power to achieve a result, and this result is consequence of more or less arbitrary assignments of importance (cost versus correctness) or “goodness” (membership in Q -sets).

1.7.3 – Incremental Q-sets

The Q-set methods described are designed for a “one-shot” design of the classifier. Many real world systems can benefit a lot from what is called incremental learning or incremental update. The basic idea is to change the classifier whenever relevant new information arrives.

Let us see how we may adapt the positive-only Q-set to incremental learning. There are two new facts that may occur: the addition of a new prototype, and the addition of a new selection pattern. In any case, we must keep the original Q -Sets.

1.7.3.1 – Adding new patterns to X_{train}

If we plan to add new patterns for selection, we must keep the original prototypes, even though they are not used for classification. We will not have to change the calculated Q -sets ever again, so we have no need to keep the original patterns of X_{train} .

When a new pattern is added, its Q -set must be computed, which is a relatively fast procedure. Next, it must be checked whether any of the prototypes of that Q -set is in the classifying prototypes. If any one is, then no modification is necessary, and we simply store the new Q -set together with the others. This is a necessary step, since there may be another simplification of the original correctness function that yields a solution that has none of the prototypes of this new Q -set.

If the new patterns is not correctly classified by the selected prototypes, i.e., if none of the prototypes of its Q -set are present, then an update of the classifier is required. This can be done in two ways: a temporary update, or a recalculation of a prime implicant. In the first case, we will

simply add one of the prototypes of the new Q -set to the selected prototypes. The resulting selection will still be a prime implicant, and thus a local minima of the cost function, but it may not be the minimum size one. The classifier that we had previously selected is one of a series of terms of the DNF of the correctness function. When we multiply this function by a disjunction of literals (the new q -function), we will be adding 0 or 1 literal to each of the terms, depending on whether that literal already existed or not. Without being too formal,

$$Correct(X_{train}, h) = \sum \prod p_k = \prod p_{kq} + \dots + \text{Present Solution} + \dots \prod p_{kn} \quad (82)$$

\Leftrightarrow

$$\begin{aligned} Correct(X_{train}, h) \cdot q(x_{new}, h) &= (\sum \prod p_k)(p_a + p_b + \dots) = \\ &= \prod p_{kq} \cdot p_a + \dots + \text{Present Solution} \cdot p_a + \dots \prod p_{kn} \cdot p_a + \prod p_{kq} \cdot p_b + \dots \\ &\dots + \text{Present Solution} \cdot p_b + \dots \prod p_{kn} \cdot p_b + \dots \end{aligned} \quad (83)$$

The question now arises as to which of the prototypes of the new Q -set should be added, since any of them will provide a locally minimum solution, with equal size. One again, the greedy option of including the one that appears more times in existing Q -sets seems to be the best. The number of occurrences of a prototype in the Q -sets may be seen as a measure of the probability of being in the Q -set of patterns of that class, so choosing it will maximize the probability that another pattern (that may be added later) will need it.

However, if we value keeping the class boundaries more than lowering the number of prototypes, it can be argued that the prototype that appears less times in the Q -set is closer to the boundaries, and thus should be selected.

1.7.3.1 – Adding new prototypes to P

Adding new candidate prototypes, while being very valuable from the classification point of view, is the less favorable option from the computational point of view. When doing this, we must keep not only the original Q -sets and prototypes, but also the original selection patterns of X_{train} . Together with the Q -sets, we must also keep the similarity value of the last element of the set (the one less similar to the pattern), which we shall call $s_{good}(\mathbf{x})$, and the similarity of the first element of $R_I(\mathbf{x})$ (the most similar pattern of the wrong class), which we shall call $s_{bad}(\mathbf{x})$.

When the new prototype is presented, we must compute the similarity between it and all the stores selection patterns of X_{train} . For each of these, if that similarity is less than $s_{bad}(x)$, no further action is necessary. If it is less, than some updating is necessary. If the pattern in question has the same class as the new prototype, than we only have to add that new prototype to it's Q -set. However, if it has a different class, than the new Q -set for that pattern will have to be reduced. We must therefore compare the similarity of the new pattern with the similarity to all prototypes on the Q -set, and keep only those that are more similar than the new one.

PART II

CHAPTER 2

Binary Self-Organizing Map - BSOM

2.1 – Introduction

Although many variants of SOM exist, very few are designed to work with binary data. We did not find in the literature a detailed discussion of specific problems of adapting the SOM algorithm to binary data, so we prepared it and present it here. We will start by discussing the problems and advantages of using SOMs for binary data. We will then propose an adaptation of the SOM algorithm for binary data, we present the results we obtained with our implementation. Finally, we will briefly review the existing implementations of SOM that use binary data.

The main reason why so few attempts have been made at using binary data with SOMs is that at first sight it seems impossible. SOMs rely on the principle that one can slowly and smoothly make the units move towards the centers of data clusters, and that there can be a topological ordering of the data.

Binary vectors will always be on the vertices of binary hypercubes, and thus a smooth approach between two vectors, along any axis, is impossible. It also seems counterintuitive to try and find a topological ordering amongst points on such vertices.

However, we believe both these theoretical difficulties can be overcome. If two binary vectors differ in a number of bits, then we can make them closer by changing only some of those bits. It is true that the approach between those vectors will be done in quantized jumps, but providing the number of bits is large, those jumps can be viewed as relatively smooth. Furthermore, there is no real reason for not it is possible to find topological ordering amongst binary data, since distance metrics exist for these data.

The advantages of using a binary SOM are considerable. On one hand, it makes it possible to directly use data that are by nature binary, without pre-processing or encoding. On the other hand, a binary SOM can be much faster than a conventional SOM. A binary SOM may be implemented directly in hardware (see Part I-Chapter 4), but more important, can be efficiently implemented in Assembler, making use of the multimedia instructions available in many modern processors. In the popular Intel Pentium architecture, for example, these multimedia instructions can make logical operations on 80 bits at a time, making it possible to train extremely high dimension binary SOMs efficiently.

2.2 - Binary SOM algorithm

The basic SOM algorithm was described in detail in Part I of this thesis, but can be summarized as follows:

For a given training pattern x :

1. Calculate the distance between each SOM unit and the training pattern x . (*Calculation phase*)
2. Find the neuron with smaller distance, and call it the winner W . (*Voting phase*)
3. Change the network neurons with a function G , which depends on the learning rate \mathbf{a} , the distance d to W (in the output plane), and the neighborhood function F . Due to the nature of the neighborhood function, only the neurons closer to W (in the output space) will be changed. (*Update phase*)
4. Update the learning rate \mathbf{a} and the neighborhood function F according to some rule. Repeat steps 1 to 3 for the next training pattern, until some stopping criteria is reached.

When using binary data two problems arise with this algorithm:

- a) Which distance metric is more appropriate.
- b) How should the updating be done.

The first problem is quite easy to solve, and we chose to use the Hamming distance between the patterns, since this is a common distance metric for binary patterns. It must be noted however, that in this case using a Hamming distance is not very different from using Euclidean distance: if we consider each component of the feature vector to be an axis, the Euclidean distance is simply the square root of the Hamming distance. When using Hamming distances the distance increases linearly with the number of different bits, while the using Euclidean distances would result in smaller increments in the distance when the number of different bits grows.

The second problem is slightly trickier. In the original algorithm, the neuron being trained can adjust its weights so that it moves slowly towards the input pattern, along the axis defined by the two patterns. When using binary valued patterns this is not possible, since the coordinate in each dimension can only be 1 or 0, making it impossible to take small steps, and limiting the directions along which the patterns can be updated.

One solution would be to allow the neurons to be real-valued. In this case, it would be possible to use the standard updating rule, and any standard SOM software (including Kohonen's original software) could be used for training. After training, the map neurons could be converted back to binary-valued patterns, so as to enable a faster classification. The main problem with this approach is that training would be as slow as a conventional SOM, and we wouldn't be using the full potential of a binary implementation.

Another solution would be to multiply the learning rate by the Hamming distance and the neighborhood function value to obtain the number of bits to update. If the number of binary features is very large (such as in our problem), the number of bits to update will be fairly large, and this method could provide fairly smooth steps. If a neuron differed in only one bit from a given input pattern, it would never be updated by it, and thus there would always be a certain quantization error. As for the direction of those steps two possibilities arise: we can deterministically choose which bits to update (thus giving more importance to those bits), or we can use a random or pseudo-random choice, that would on average take us along the path towards the final target pattern.

Our solution to this problem is to probabilistically change the bits in which the input pattern and the neuron differ, according to the value of the learning rate and neighborhood function. For each neuron W , the product u of the learning rate α by the neighborhood function F is computed. Then, for each bit in which the neuron W and the input pattern x differ, a random number $rand$ (with a uniform distribution in the interval $[0,1[$) is generated, and compared with the product u . If $rand < u$ that bit of the neuron is changed, to assume the value of the corresponding bit in the input pattern x .

On average, the direction of movement is along the desired path, and the number of bits changed is proportional to the learning rate.

Resuming, there are three possible approaches in the pure binary input space:

Number of bits to change	What bits to change
Fixed	Fixed
Fixed	Random
Random	Random

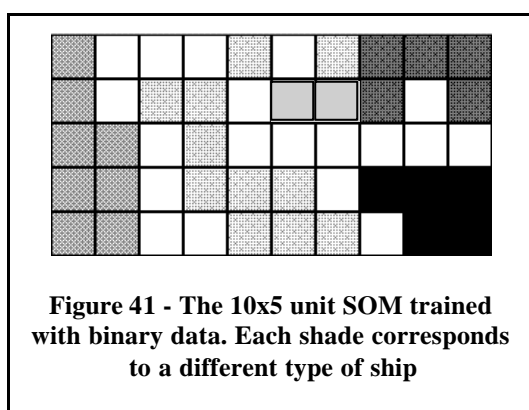
Table 8 - Proposed solutions

These solutions were implemented in our version of SOM, presented in Part III-2.

2.3 – Results obtained with BSOM

Our implementation of BSOM was written in C++, without trying to take full advantage of the processor's architecture. We did this because we did not have time to optimize the code, and because we wanted to try the BSOM approach before committing too much time to its optimization. Thus, we cannot present fair experimental comparisons between execution time of BSOM and conventional SOMs.

We tested the BSOM with binary data extracted from the ship noise spectra described in Part III-3. As mentioned there, it consists of 165 patterns with 2048 binary features each. First the whole data set was used to train the SOM, with each of the three update rules proposed.



Surprisingly, the final SOM obtained was identical for all our experiments with the 3 rules, apart from the axis symmetries that are inherent to SOMs. The mapping obtained is shown in Figure 41. Although by no means conclusive, this shows that the BSOM algorithm is very stable, and all the update rules proposed are approximately equivalent. Since the fastest rule is

the first, that modifies a fixed number of bits in a fixed order, that was the one used in subsequent experiments.

We proceeded to use the BSOM in subsequent experiments, where we divided the available data into various training and test sets to cross-validate the results. One more, we observed that, apart the mentioned symmetries, the mappings obtained were almost identical, varying in only one to four unit labels.

2.4 – Other work

A simple way to implement a binary SOM is to use the conventional SOM algorithm during training, with real valued units, and then threshold the final weights to obtain purely binary units. Such an approach is used by (Gioiello, M., G. Vassallo, *et al.*, 1992) when constructing a LVQ map to process binary data. Using the conventional Euclidean distance is not very different from using a Hamming distance, as seen previously, and the use of real valued units makes the updating trivial. All the speed-up benefits of using a purely binary system are lost during training, but at least the final network can benefit from them when processing new data.

Most of the SOM implementations for binary data are designed for processing images. The techniques used for determining similarities and updating images (i.e. making them more similar to a target image) are not applicable for general binary patterns, at least directly, since they rely on specific 2-dimensional information.

A recent example of the use of SOM for binary images is provided by (Pujol 2001), that follows up on previous work by the same research group, namely (Takacs and Wechsler 1998). A modified Hausdorff distance is used to compare images. A rather complicated update rule is used, that relies on derivatives in order of the x and y coordinates of a certain function of the image features. While achieving remarkable results, the technique is not readily adaptable to generic binary data, since it relies on relationships between the 2-dimensional coordinates of the pixels. Furthermore, both the Modified Hausdorff distance and the update rule require considerable computing time, defeating one of the main advantages of the BSOM.

However, the most common approach to using SOM with binary images involves some sort of preprocessing that renders real-valued features, such as is done in (Tanomaru and Inubushi 1995).

Although not dealing with SOMs, a recent paper (Girolami 2001) proposes a way of adapting the closely related GTM (Bishop 1995) to binary data. Another one (Coultrip 1998) uses a VSLI implementation of a classifier for binary data, based on Parzen windows.

PART II

CHAPTER 3

Parallel implementation of SOM over PVM

3.1 – Introduction

Although neural networks are intrinsically parallel algorithms, they are not easily implemented on distributed architectures because the strong interactions between neurons impose a very high communication overhead (neural networks are also called connectionist models).

One of the neural models that has been implemented with more success onto parallel architectures is Kohonen's SOM (Kohonen 2001), because, as seen in Part I-4, it requires very

little communication between units. However existing implementations have traditionally used either dedicated VLSI chips (Rueping 1994), or parallel machines that tend to be expensive and non-standard (Przytula, Prasanna *et al.* 1993).

Over the last few years, a system called Parallel Virtual Machine (PVM) (Geist, Beguelin *et al.* 1994), has been developed that enables a programmer to use networked computers (running different operating systems such as UNIX and MS-Windows 95) in a manner very similar to a single UNIX machine, using common languages such as C. Thanks to PVM, existing computer networks, no matter how heterogeneous, can easily be programmed. Moreover, simple PCs running MS-Windows (which abound in most organizations), can be put to work during otherwise unproductive times, such as nights and weekends.

Our motivation for using this approach was to be able to train very large SOMs on the University's computer laboratories, that contain large quantities of networked PC computers, running either MS-Windows or Linux. A more far reaching application would be to use the large networks of superstores (including the all the registering machines that are PCs) to cluster data from the previous day sales during the night. With this approach, it would be possible to use SOMs that otherwise could only be trained in reasonable time on supercomputers.

3.2 - Distributed SOM algorithm

The basic SOM algorithm was described in detail in Part I of this thesis, but can summarized as follows:

For a given training pattern x :

1. Calculate the distance between each SOM unit and that training pattern x . (*Calculation phase*)
2. Find the neuron with smaller distance, and call it the winner W . (*Voting phase*)
3. Change the network neurons with a function G , which depends on the learning rate α , the distance d to W (in the output plane), and the neighborhood function F . Due to the nature

of the neighborhood function, only the neurons closer to W (in the output space) will be changed. (*Update phase*)

4. Update the learning rate \mathbf{a} and the neighborhood function F according to some rule. Repeat steps 1 to 3 for the next training pattern, until some stopping criteria is reached.

Many different distributed versions of Kohonen's SOM are possible, each being more adequate for a certain machine architecture. For implementing in PVM, we think the most adequate is the following.

Algorithm 13 - Algorithm 13 - The distributed SOM algorithm.

Given	
N_p	Number of processors, or PVM host
C	A coordinator process
N_t	Number of training patterns $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$
N_n	Number of units forming the SOM
Do	
1	Send the SOM units to the processors, so that each processor receives approximately N_p / N_n units
2	Send all N_t patterns to all N_p processors
3	For each pattern \mathbf{x}_i
4	In each of the N_p processors do
5	Find the BMU (Best Match Unit) within that processor
6	Send the coordinates of the BMU, together with the similarity measure, to the coordinator C
7	Wait for the coordinator C to choose the global BMU, and receive that information
8	Update the local units according to the SOM rule
9	In each of the N_p processors update de learning parameters
10	Repeat steps 3 to 9 until the stopping criteria is met
11	Send all units back to the coordinator C

Steps 1 and 2 form the initialization phase, that requires a lot of communication amongst processors. Step 5 consists of finding the local winner in each processor. Steps 6 and 7 are the voting phase, that does require some communication between processors. Step 8 is the update phase, that is again purely local.

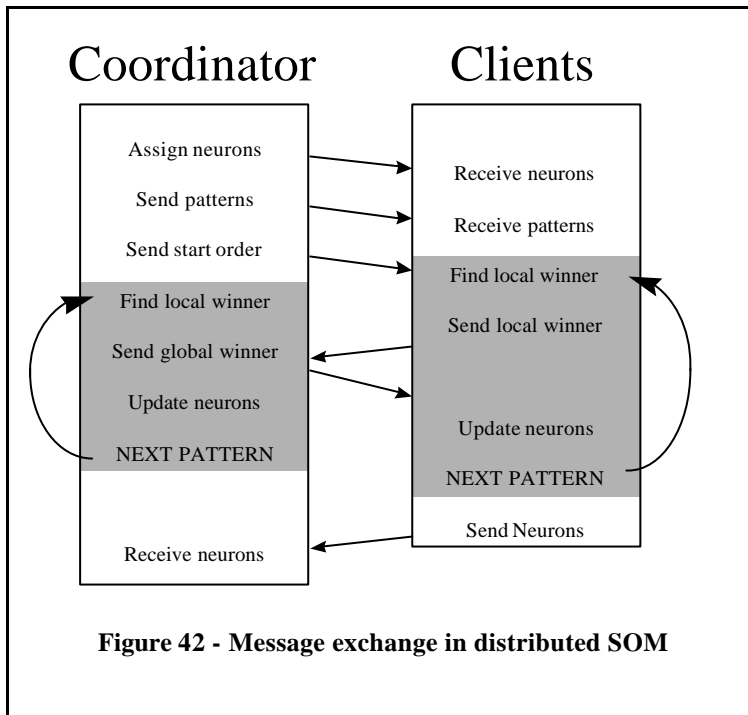


Figure 42 - Message exchange in distributed SOM

A graphical representation of the algorithm showing the messages involved is given in Figure 42.

The initial and final phases of the algorithm (represented in white in Figure 42) are executed only once, and thus have very little influence in the overall performance. Most of the time is spent in the main loop (represented in gray in Figure 42), which iterates

through the three main phases: calculation of the distances, voting for the global winner, and updating the units.

The calculation phase of the algorithm (finding the winner) is inherently parallel, and its computational load can be spread evenly across the network if each processor has roughly the same number of units.

The voting phase is the only one that requires communication (and synchronization) between processors, because the global winner must be known to all for the algorithm to proceed. If there was no coordinator, each processor would have to send information about its local winner to all other processors. While PVM does support a broadcast mechanism, this would translate to a multicast at the data link level, thus originating $N_p(N_p-1)$ messages per iteration. With a coordinator, each process sends only one message to the coordinator, and it in return sends only one message back, thus originating only $2(N_p-1)$ messages. Furthermore, the coordinator can piggyback additional information on the return message, that can be used to select the next training pattern, change the training parameters, etc.

During the update phase, each processor will have to calculate the distance between its local units and the global winner (in the output plane), and then update only the units in the winner's neighborhood. The computational load will be evenly distributed only if all processors have

roughly the same number of units in this neighborhood. Thus it is very important to assign the units evenly during step 1 of the distributed algorithm. When the neighborhood radius is large, the computational load will easily be distributed. However, when the neighborhood radius is small, it is difficult to guarantee that all processors will have the same number of units, and thus some processors will have to wait for others. It must be pointed out that in this case, the number of units to update will be small, so the difference in processing time amongst the processors will also be small.

3.3 - Experimental results

During our experiments, we used networks of up to 12 PCs. Each was a Digital Venturis FX, with a Pentium running at 100 MHz, with 16 Mb of RAM (255 K cache). The computers were connected with a coaxial cable, using 10Base2 level 2 protocol (10 Megabits per second), and TCP/IP as the level 3 protocol. The computers were running the MS-Windows 95 operating system, WPVM 2.0 (Alves 1997) and Microsoft LAN Manager peer-to-peer network clients and servers. The computers had all screen-savers and anti-virus checkers disabled and were not running any other software during these tests.

As the speedup obtained by using the distributed SOM depends critically on the amount of processing required before each synchronization, we used very large pattern vectors, with 1024 features, and then varied the number of neurons on the map.

We used square maps with 5×5 , 10×10 , 20×20 , and 40×40 neurons. Although square maps tend to slow convergence, we used them because they have a shorter boundary than rectangular maps (for the same number of units) and thus are less affected by discontinuities of the neighborhood function on those boundaries. This discontinuity effect would also affect the smaller maps more than the large ones if we used the same initial radius in all tests. To avoid this, we used an initial neighborhood function radius equal to each map's side. So as to make the radius decrease smoothly, we force it to decrease only 1 unit each time the whole training set patterns are presented. The map with 5×5 neurons will thus have only $5 \times N_t$ iterations of the patterns, while the one with 40×40 will have $40 \times N_t$. So as not to make the simulation too long for larger maps, we use fewer training patterns for these maps. In the end, each map requires exactly 4 times more calculations than the previous one.

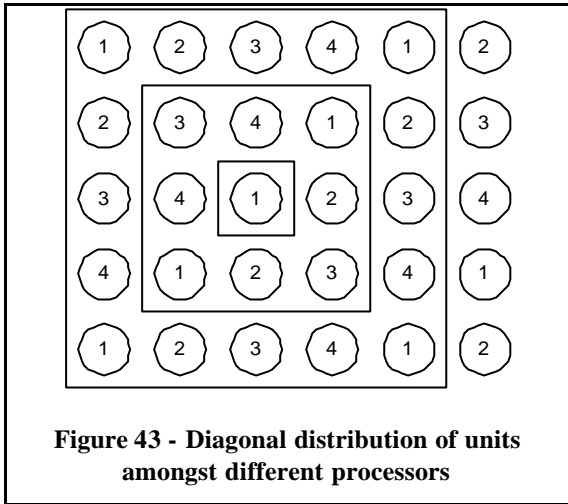


Figure 43 - Diagonal distribution of units amongst different processors

The number of patterns used for training does not influence the performance or speedup of the algorithm, so we use only 120 patterns for the 5×5 map, 60 for the 10×10 , 30 for the 20×20 and 15 for the 40×40 . While this number of patterns (and iterations) would be far too small for a useful classification or clustering, it is sufficient to prove that the system works reliably. The number of training patterns in the smaller

maps has to be greater than for the larger maps, because otherwise the time intervals would be too small to be measured reliably.

During these simulations we distributed the units amongst the processors in such a way that each processor has one or more diagonal lines of units. This distribution, although not ideal, provides a reasonable equilibrium of units by processors for rectangular neighborhoods (see Figure 43 - every uncut neighborhood has the same neuron load per processor at each radius).

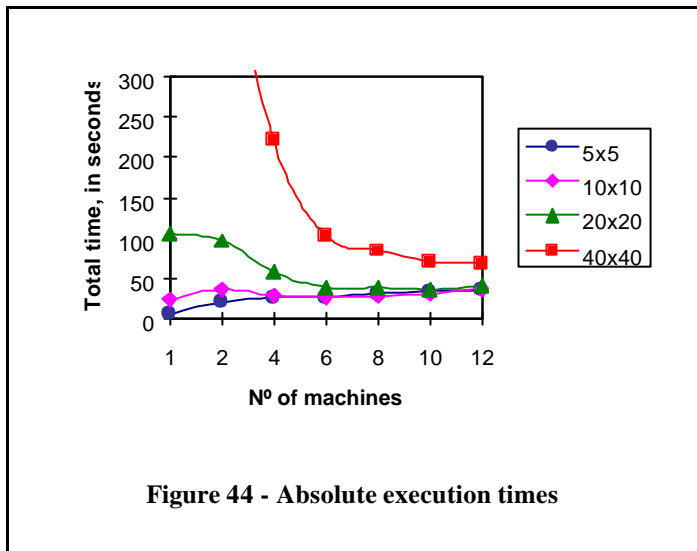
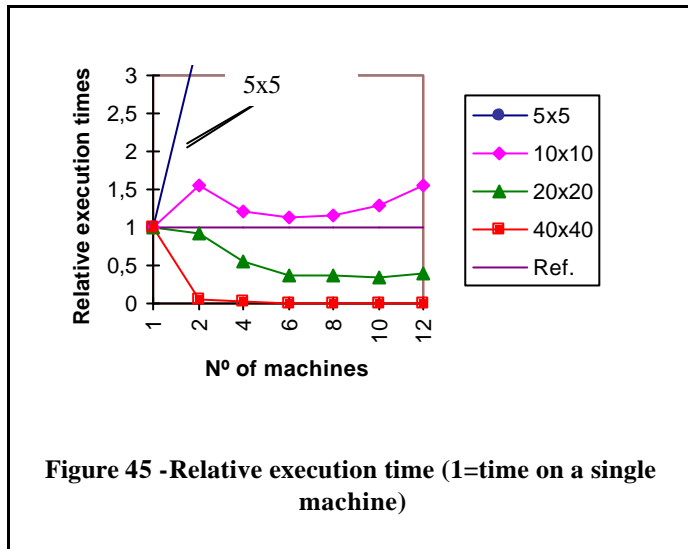


Figure 44 - Absolute execution times



The execution times were measured within the programs (with a call to a system timer), so as to measure only the time spent on the iterations (step 3 of the distributed algorithm, and shaded area in Figure 42).

The numerical results are presented in Table 9 and in Figure 44, we can see a graph of the absolute execution times,

while in Figure 45 we can see the relative times, that is, the speedup.

N° of PCs	Size of map			
	5x5	10x10	20x20	40x40
1	6	24	106	15268
2	21	37	97	623
4	27	29	58	222
6	27	27	39	103
8	32	28	39	85
10	34	31	36	71
12	36	37	41	69

Table 9 - Execution times (in seconds)

3.4 - Conclusions

The results presented confirm the claim that the SOM can be efficiently distributed on an ordinary computer network. However, depending on the workload, we may obtain overwhelming gains (as in the 40 × 40 map), moderate but consistent gains (as in the 20 × 20 map), or even high losses (as in the 5 × 5 map).

There are a couple of extremely high running times corresponding to the 40×40 map running in a single machine or in a group of two. After that there is a sudden break and then the running times decrease smoothly. This initial peak is due to the machine configuration we are working on, namely because we have 16 Mb of RAM and a 40×40 map takes up to 13 Mb RAM, forcing the operating system to use the disk as swap-space. We used machines with this configuration because we needed to have a reasonable pool of highly similar computers to achieve fair comparisons, and these were the ones available. Nevertheless we consider this as an advantage instead of a shortcoming, since these were the most common machines around any office or university lab at that time, and allowed us to expose another very important fact when using distributed processing - the efficient use of each machine's memory. Using our distribution model, one can take advantage of each machine's local memory along with the corresponding processing power, effectively avoiding a RAM/HardDisk swapping situation which terribly slows down the SOM processing.

In the more general case, the total execution time will decrease smoothly (except sometimes for the transition from 1 to 2 machines), and then start to increase slowly. It's not reasonable to expect an unlimited gain as you throw more and more machines into the pool because of the increase in network load.

When distributing a process, there will be a minimum overload on the total running time, due to the network. In our tests this can be seen by observing that the highest jumps upwards happen when there is a switch from a single machine to two machines. The distribution is profitable only when there is a significant workload to be distributed, thereby overcoming this minimum network overload. After that initial step, all other machine extensions walk along an almost smooth curve, reflecting the converging equilibrium between each machine's designated workload and the network overload due to an increasing number of traded messages.

The overload due to the network will increase linearly with the number of machines (each additional machine will be responsible for 2 new messages), until the network starts to saturate due to collisions and/or sending queues. The workload per machine, on the other hand, will decrease hyperbolically, so at a certain point a minimum execution time will be reached, and adding a new machine will not improve the overall performance.

PART III

Application

PART III

CHAPTER 1

Ship noise and target identification

1.1 – Introduction

The main objective of this thesis is to enable a submarine to identify the ships that are near it by analyzing the sound they produce. This is a crucial problem for submarine operation, and as we progressed in our work we found many other areas of application where the same techniques can be used, both for military and civilian purposes.

To solve this problem, it is necessary to understand how and why ships produce noise, how that noise is propagated in the ocean, how it is mixed with noise from other sources, and how it is received by the submarine. A graphical representation of this framework is shown in Figure 46.

There has been a lot of research in underwater acoustics, driven not only by this application, but mainly because of its importance to communications (Green 1997) and underwater imaging. Many excellent textbooks exist on this topic such as (Medwin and Clay 1998), (Kleppe 1989) or (Kinsler 1982). A good reference is the Handbook of Acoustics (Crocker 1998).

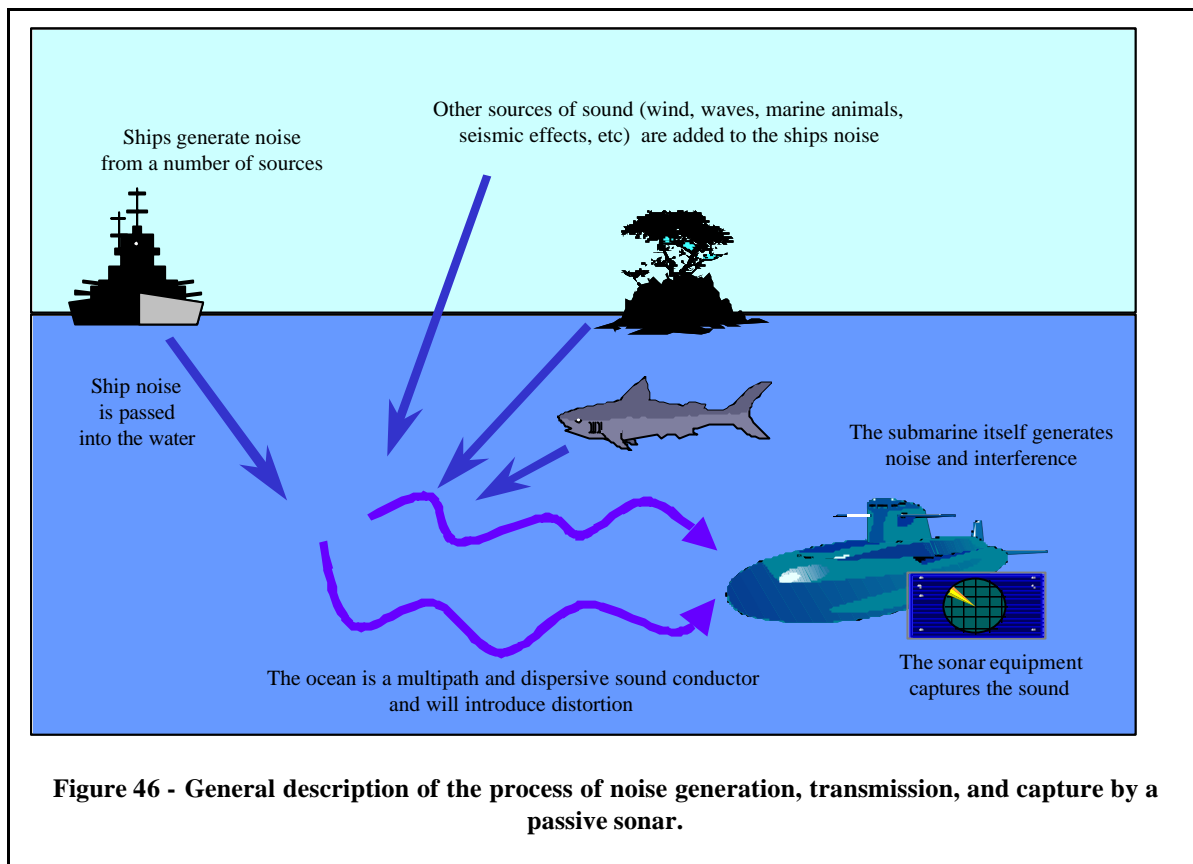
As for the specific problem of characterizing the sound radiated by ships and its recording by sonars, there is far less bibliography. The most cited textbook is undoubtedly (Urlick 1982) which is still used by many navies. Since it is a re-edition of a book written in 1967 and because a significant amount of research has recently been done, we would recommend (Coates 1990) as a solid and general purpose textbook on underwater acoustics for sonar related problems. A short but detailed account of radiated noise with a lot of experimental data can be found in (Collier 1998) which has a naval architect's perspective. For a more military and strategic perspective, although scientifically correct and complete, we would recommend (Stefanick 1987). This last book gives a great deal of attention to the problems that were crucial during the Cold War but its appendices provide a quick reference for all underwater acoustics issues with military interest. For a more in-depth study of the physics of sound generation and propagation, we would recommend (Ross 1987). For the signal processing aspects of sonars (Nielsen 1991) or (Burdic 1991) would be recommended.

1.2 – The basic problem

Ever since the first submarine was used, during the American War of Independence, their detection and identification has been one of the most important issues for any navy. On the other hand, submerged submarines have always had serious problems in recognizing and identifying the underwater world around them. Let us now see what means they have at their disposal and how they have been used.

Visual detection is all but impossible underwater. Ocean water is quite opaque, and visibility is at best a few dozen meters, which is far too little to be of any use. Moreover, there is hardly any

light at the depths at which most submarines operate. Due to these factors, regular military



submarines do not have any windows (or portholes) to observe their surroundings under water.

Most modern ships use electromagnetic sensors intensively, such as radars. Unfortunately, ocean water is a reasonable electric conductor and, as such, does not allow easy propagation of electromagnetic waves. For VHF/UHF radio-frequencies, the typical depth of penetration in water (skin depth) is of only a few centimetres. For normal navigation radar frequencies, sea water is almost completely opaque and gets more and more opaque as frequencies rise.

Very low frequency (VLF) and ultra low frequency (ULF) radiation can penetrate the ocean to a certain extent. In fact, these frequencies have been used successfully for communication with submarines. However, they have very long wavelengths and thus produce a high degree of uncertainty in the location of targets. They are also subject to strong diffraction and reflection effects that, together with a strong attenuation and the need for enormous antennas, render them useless for use by a submarine.

Sound waves, however, can be used instead of electromagnetic waves and provide the much needed “eyes” for submarines, as indeed they do for many marine animals, such as dolphins and

whales (Roitblat, Moore *et al.* 1989) (Council 1994). Water, given its high density and low elasticity, is an excellent sound conductor.

In 1490, Leonardo Da Vinci described an instrument that could be used to hear underwater sound and detect approaching ships (Urlick 1982). Ever more sophisticated versions of this apparatus, which was based on a design of hollow tubes capped with flexible membranes, were used until the beginning of the 20th century. They generated the first scientific studies on underwater acoustics, on Lake Geneva in 1827, and enabled a human operator to determine a ship's direction of approach with an error of less than one degree.

With the discovery of the piezoelectric effect and electric/electronic engineering, mechanical devices gave way to electronic ones based on hydrophones, which have been used ever since World War I. In the first submarines, the crew simply listened to the sound surrounding the submarine, in hope of identifying the engine "roar" of any approaching ship. This is what is now called passive sonar⁹.

With the development of electronics, a more sophisticated technique was devised, whereby a short burst of sound (called a *ping*) is generated, and the time between its generation and the arrival of its echo is measured. This enables the operator to extract two important pieces of information. The time interval will enable us to know the distance to the source of the echo provided we know the speed of sound in the water at that time. The magnitude of the echo will give some information about the object's size, rigidity, and attitude, i.e., a sandy bottom or a school of fish will give a faint echo while a rocky bottom or a large steel ship will give a strong echo. A device that uses these principles is what is called active sonar. Active sonars are widely used. Warships use them to detect targets, fishing ships use them to find fish, and almost every ship uses them to determine the water depth. When used for this purpose, they are known as sounders.

⁹ Sonar is the acronym for "Sound Navigation and Ranging". The name originated in the British Royal Navy, when the first electronic prototypes were used to determine distances. Nowadays, the term is used to describe any naval system that uses sound as a means to detect, localize, or identify any object.

Modern active sonars are quite sophisticated pieces of equipment. Thanks to the development of signal processing and computing power, they now use elaborate signal processing techniques to perform beam forming and noise cancellation. The pings have also evolved to multifrequency (or coloured) signals that, when reflected, can bring back much more information about the target.

Despite the undeniable advantages of active sonar, their use has several drawbacks. From a military perspective, the main one is that active sonars reveal the presence and position of those who use them. Therefore, their use by a submerged submarine would defeat the submarine's most important advantage, which is its stealth. Active sonars also give little information about the target. A few characteristics of the target can be determined by the vessel's echo (Group 1988), namely rigidity, shape, and speed, but much more can be revealed by listening to the target's own distinctive noise. In addition, there has been concern that the widespread use of sounders and active sonars interferes with biological life and, in particular, with the navigation systems of marine mammals.

Passive sonars have therefore become the preferred means of surveillance for submarines and for submarine hunting, also known as anti-submarine warfare. During the Cold War, the United States Navy installed a network of passive hydrophones to monitor the movement of Soviet submarines in strategic locations, first in the North Atlantic but later worldwide. That network, known as SOSUS, an acronym for "SOund SURveillance System", is still operational, although no longer permanently monitored as it was for so long. SOSUS has also been used for civilian purposes such as monitoring whales and earthquakes (Nishimura 1994). The interest in passive sonars for fishing is also increasing, both as a means of identifying species and for monitoring fishing stocks (Mueller 1993).

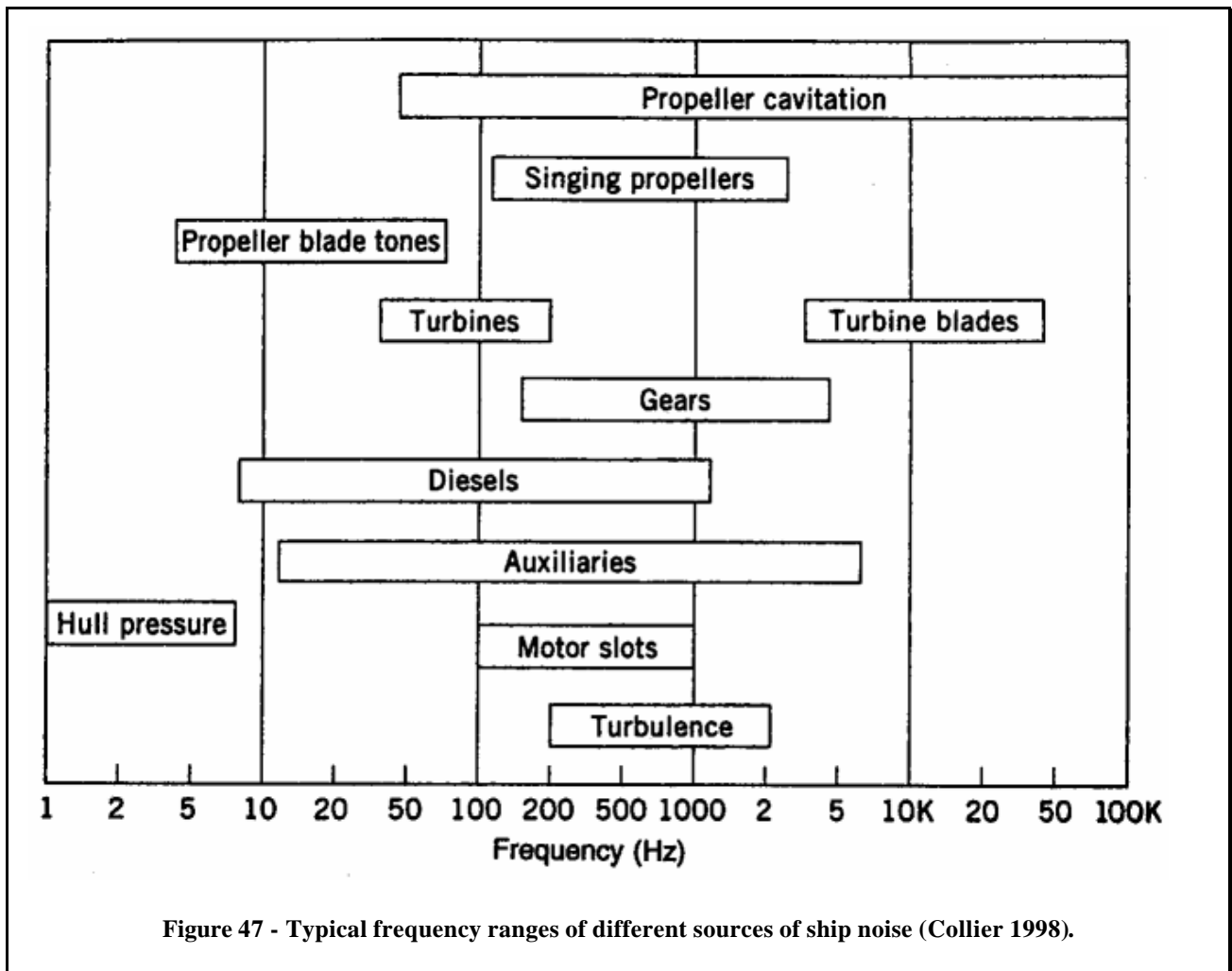
Although papers on the subject have not, as far as we know, been published, we envision passive sonar systems as backup for vessel traffic monitoring and anti-smuggling surveillance. These systems traditionally rely on radar and, when necessary, on visual contact. However, radar visibility can suffer degradation under heavy rain and radar cannot be used to positively determine the identity of a ship. This is usually done by radio-contact between the surveillance station and the ship. Smugglers will try to jam the radars with chaff which can be discreetly dispersed by a small plane towing publicity banners. They will also use very fast small boats, having a tiny radar cross-section that is easily lost in the midst of sea clutter. These boats are however very noisy and will easily be spotted by passive sonar, no matter what the atmospheric

conditions. Networks of hydrophones on the sea bottom of shallow waters are relatively inexpensive to install and operate, and therefore there is potential for widespread use of passive sonars and target identification systems such as the one developed in this thesis.

1.3 - Sound generated by ships

There are four main sound sources on a ship:

- a) Machinery (main propulsion and auxiliary machines)
- b) Propellers (or other forms of in-water propulsion)
- c) Hydroacoustic noise generated by the flow of water on the hull
- d) Other noise generated within the ship, specially under the water line

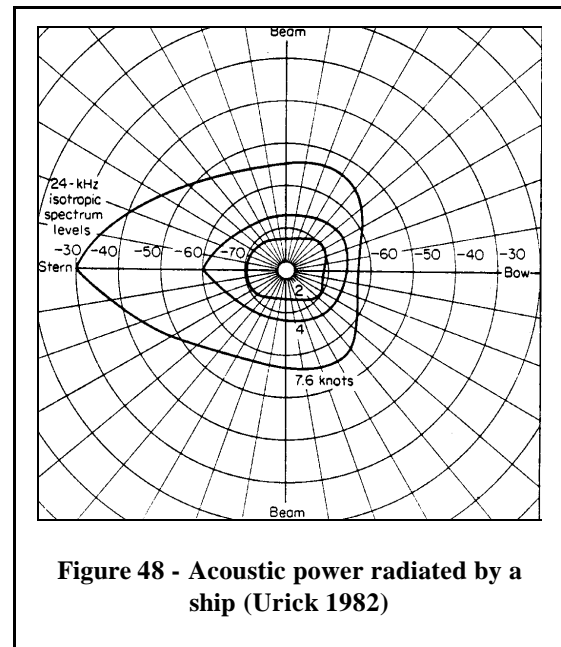


Each of these sources has a typical frequency band (see Figure 47) and exhibits different behaviour under different conditions. Most of the information is in the 10 Hz to 2 kHz range, although information also exists at other frequencies.

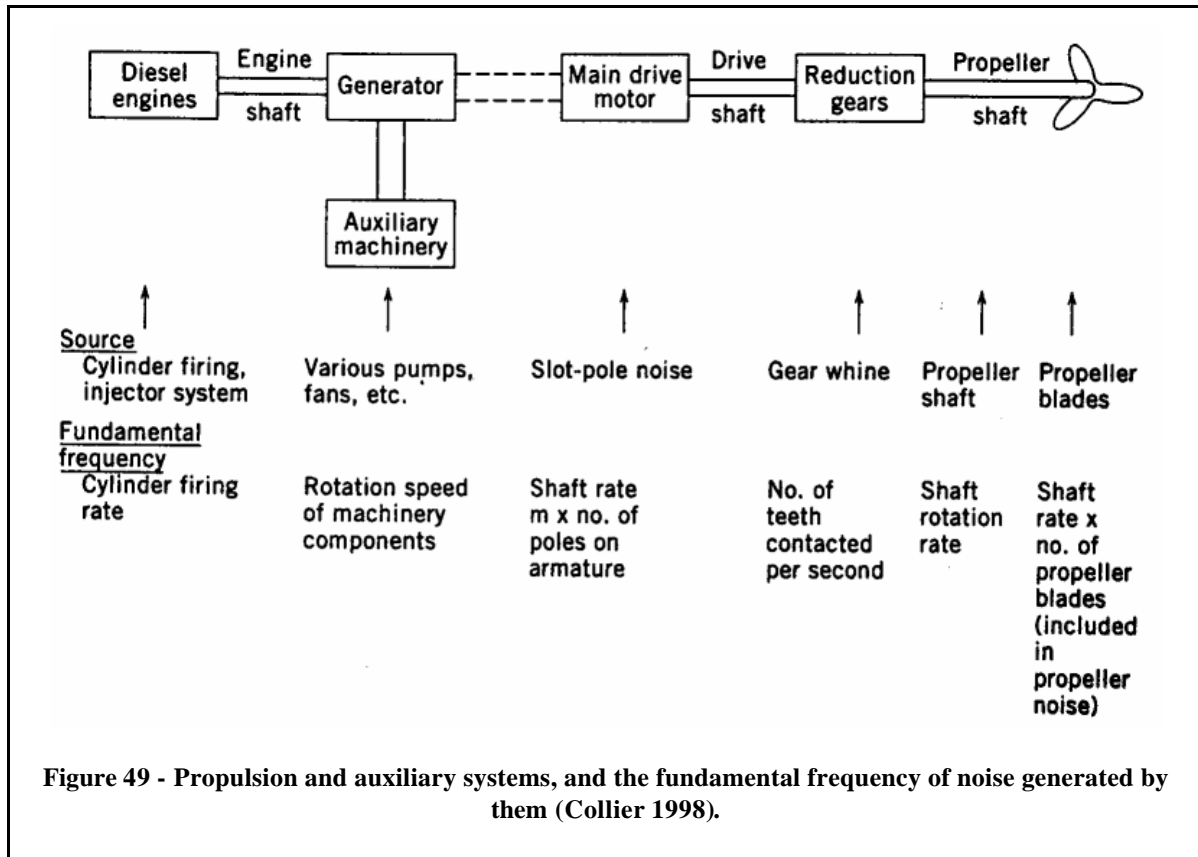
The volume and characteristics of noise generated by a ship will depend on the direction from where it is heard. Mainly due to the propeller, ships tend to radiate more noise to their stern than to the bow, as can be seen in Figure 48.

1.3.1 - Machinery noise

Under normal operating conditions, machinery noise is dominant over other sources in most ships. Different types of machinery can generate quite a variety of noise, as shown in Figure 49. Diesel engines, the most common type of engine, have a number of cylinders and the firing rate of these will determine the dominant frequency of the noise generated. However, very slight imbalances always occur between the cylinders, and a small power peak is usually observed at the basic frequency of individual firings (Ross 1987). By comparing these two frequencies, the number of cylinders of that particular engine can be estimated. Turbine engines tend to be noisier than diesels but can be strongly damped if necessary. Their main fundamental frequency is rotation speed, due to slight imbalances between the blades. There will also be a strong component at a frequency corresponding to the number of blades multiplied by the rotation speed, since this is the frequency at which hot air hits various components. Electric motors and generators, either for the main propulsion or for auxiliary systems, will generate noise at the basic shaft rate. They will also generate noise at basic shaft rate multiplied by the number of poles on their armatures. Of the three most common types of machinery, electric engines are by far the most silent. Conventional submarines use these engines while submerged, making them very difficult to detect. After engines, the next most significant sources of noise are the reduction gear boxes, that make the coupling between the propulsion machines and the propeller shaft. Under certain circumstances, they may even produce more noise than the engines. The fundamental frequency corresponds to the number of teeth contacted



per second. Some types of engines, such as electric, can work at the relatively low rotation rates of propellers, thereby forgoing the noisy reduction gears.



Noise generated by the ship's machinery reaches the ocean water only after traversing its structure and the hull/ocean interface. This transmission process has a huge impact on the sound. Most ships have shock absorbers (or dampers) on the engine mounts. These reduce the tear and wear and increase crew and passenger comfort. From a military perspective, shock absorbers are essential in decreasing the ship's acoustical signature, and thus increase its survivability in a war scenario. The transmission of sound through the structure is an important part of naval architecture. As far as we are concerned, the most important issues are that the transmission process generates many harmonics due to non-linearity in many joints, and that internal compartments can act as resonators, strongly distorting the sound's spectra.

Most of the noise generated by machinery is concentrated at the precise frequencies described above or at their harmonics. It is thus called tonal noise or narrow band noise, and appears as narrow peaks in the spectra of the ship's acoustical signature. As the operating conditions of the ship changes, different machinery will have different behaviours. The machinery associated with

the main propulsion will generate noise with a higher frequency as the ship's speed increases, while many auxiliary machines, such as generators or pumps will not change their acoustical signatures. The exact frequencies at which these auxiliary machines generate sound, and the stability of those frequencies, can reveal important information about the exact type of machine, its maintenance status, and its age. Naturally, the ship's signature will vary considerably as different machines are turned on or off.

Machinery will also generate some broadband noise. Pistons will slap on the sides of the cylinders producing irregular noise and, together with the shaft movement, will induce a multitude of vibration modes in many different engine parts. This will give rise to a generally broadband signal (Coates 1990).

1.3.2 - Propellers

Propellers will generate very different sounds depending on whether they are cavitating or not, and on the level of cavitation (Urick 1982). Cavitation is the process that occurs when, due to sudden changes in pressure, water vaporises and forms small bubbles. The bubbles will collapse back into liquid state letting off a characteristic click. Recent design changes in propellers, a lot of it due to research in aeronautics, has drastically reduced the cavitation of propellers under normal circumstances. Unfortunately, total elimination is very difficult, since the propeller must produce forward thrust, generating large forces that will inevitably lead to large changes in pressure. To reduce cavitation, most modern submarines have large propellers with many blades, a design that generates no macroscopically perceivable cavitation. However, as speed increases, any propeller will start to cavitate, generating a distinctive loud broadband noise and the bubbles will become larger. Although their collapse will make a louder noise, the main lobe of the broadband noise generated will actually move to lower frequencies. This happens both because their greater size will allow larger wavelength and because there will be less bubbles.

Poorly designed or damaged propellers and shafts, or damaged bearings, will start to resonate at certain speeds, letting off a very loud and distinctive noise known as "singing". While the causes of singing can be corrected, given enough time and money, they are sometimes unavoidable in the short run.

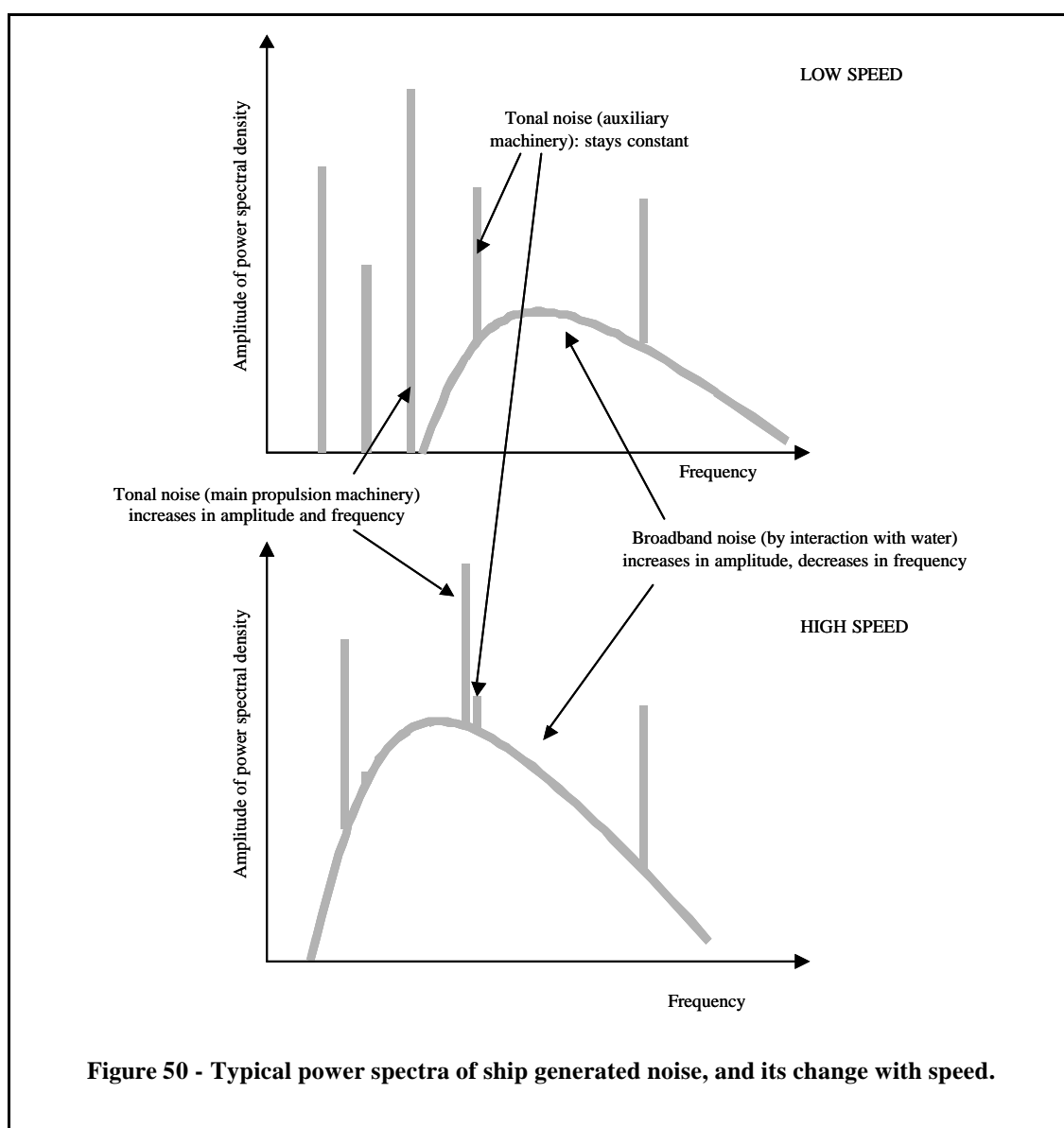
1.3.3 - Hydroacoustic noise generated by water flow on the hull

The movement of a ship through water it will generate noise for many reasons. On one hand, it generates very low frequency waves, due to the simple fact that it pushes water in the bow, sucks it in the stern, and pushes it down and sideways in the bow side panels. The wake of turbulent water will generate higher frequency noise, both by itself, and as it slaps on the sides of the hull. Finally, small irregularities in the hull will give rise to cavitation as water rushes past them, resulting in noise with even higher frequencies. The wake will also interfere with the propeller making it suffer pressure changes. On the other hand, the propeller will induce noise in the hull as it pushes water into it with varying pressure. As speed increases, the amplitude of the hydroacoustic noise will rise considerably but, for the reasons explained in the previous section, will tend to have its peak at lower frequencies.

1.3.4 - Other sources of noise

All noise generated within a ship will eventually find its way into the ocean. Some is made directly by people inside the ship as they go about their daily chores. In a submerged submarine, all these factors must be controlled, and modern submarines have insulating materials on the inside hull to dampen noise. Loose or improperly fastened objects, such as dangling keys or improperly secured fire extinguishers, will also produce considerable noise when a ship is rocked by the waves. Finally, vibrating pipes (due to irregular flow), discharge of refrigeration water, above or below surface exhaust of gases, and other similar effects all add up to produce considerable noise.

Recently, there has been a lot of research into irregular or isolated sounds (or transients) produced by ships or submarines, that may reveal important aspects of their operations. A typical example is the opening of the torpedo hatches. It is a “one-shot” sound, but it will indicate that the submarine is preparing to take offensive action. Firing canons or missiles will also produce distinctive noises. One irregular sound that is very difficult to avoid is the one produced by rudders and their associated machinery. To keep a ship’s course, constant adjustments must be made to the rudders, turning on an off the motors that move them, and causing sudden mechanical stress on the machinery and water flow.



The general behaviour of typical ship noise spectra with speed is summarized in **Error!**
Reference source not found..

1.4 - Transmission of sound to the sonar equipment

After reaching the water, the sound generated by a ship will be propagated in quite strange ways through the ocean, where it will be mixed with a variety of other sounds, until it finally reaches a sonar equipment. Let us now see each of these effects.

1.4.1 - Ambient noise

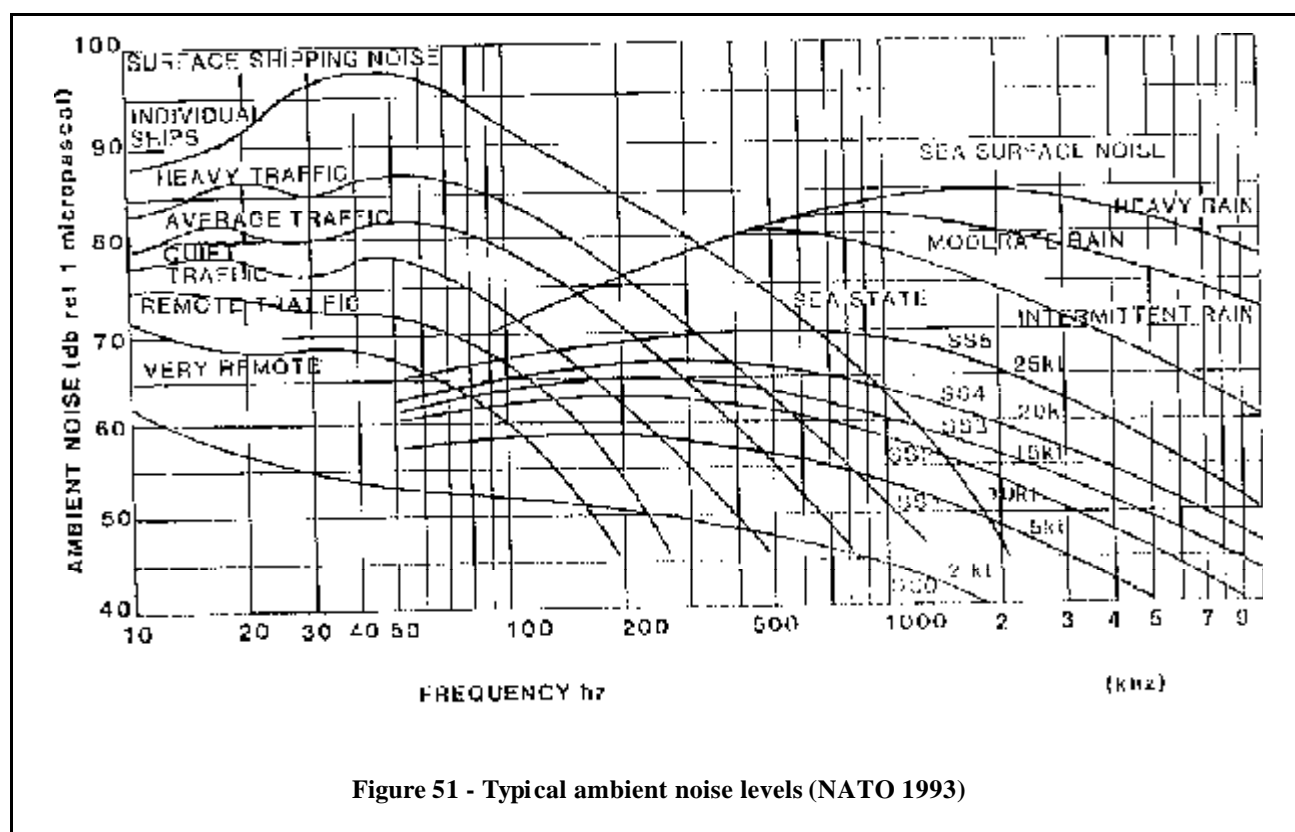
Although known romantically as “the silent world”, the underwater environment is actually quite noisy, due to a number of different causes (Urlick 1986):

- a) Human intervention. Due to the easy propagation of sound in water, even very distant shipping noise will be felt. Even though individual ships may not be distinguishable, their noises will blend together and produce a broadband humming sound. Shore facilities and dredges can also produce distinctive sounds. Since this sound (as other ambient sounds) has to travel long distances, its spectra is strongly distorted by the transfer function of the ocean itself. Typically, the higher frequencies are greatly filtered, and the general effect is a lower frequency hum than the one that would be observed if the sources were closer (Dyer 1998).
- b) Surface agitation (Dyer 1998). Surface agitation is mainly due to the interaction between the atmospheric wind and the ocean water. Higher wind speed implies a correspondingly higher level of sound produced by waves, spray, and bubbles.
- c) Biological sources (Tyack and Howald 1993). Marine animals and even plants can generate considerable noise. Many marine mammals deliberately generate sound for navigation and other purposes, varying from short clicks to long and drawn out whale songs. Many other marine animals perform regular movements, so as to be able to swim or even just move water through their gills, and this movement produces a distinctive sound. One of the most notorious is the “croaking shrimp” that violently snaps while it is moving through the water, producing a very well known sound (Lohse, Schmitz *et al.* 2001).
- d) Seismic activity (Keenan and Dyer 1984; Goodman and Yamamoto 1988; Gerstoft 1994). Although major earthquakes or volcanoes are rare, there is quite a lot of low intensity seismic activity going on at any one time. Due to the particular sound transmission

characteristics of the ocean, some sort of seismic activity can be heard almost anywhere on the planet.

- e) Ice breaking (Xie 1992). Temperature changes in polar ice causes great mechanical stress that leads to cracks in ice. These cracks produce a loud noise.
- f) Rain (Pumphrey, Crum *et al.* 1989; Scringer, Evans *et al.* 1989). Rain, together with snow and ice storms, will induce considerable noise.

A good characterization of ambient noise is very important for anti-submarine warfare, and charts with expected ambient noise levels are routinely distributed at navy briefings. One such chart, known as “Knudsen curves” is presented in Figure 51. A very detailed account of the spectra of ambient noise is given in (Wenz 1962), and a recent paper by (Andrew, Howe *et al.* 2001) shows and discusses some of the changes that have occurred over the last 40 years.

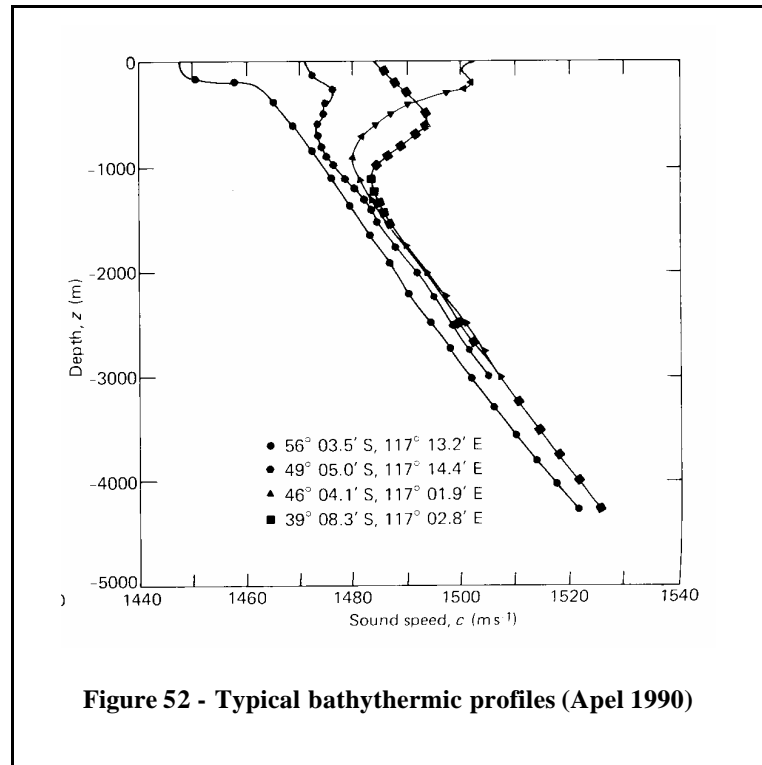


The ambient noise is not homogeneous in all directions, varying with azimuth and vertical angle of arrival. Thus, when planning naval exercises, it is important to have a characterisation of ambient noise in all directions.

1.4.2 – Propagation of sound through the ocean

Propagation of underwater sound is the most important and probably best studied aspect of underwater acoustics.

One of the most important parameters to determine the behaviour of sound in water is its propagation speed. The speed of sound in the ocean depends on the density of the water, and it can vary enormously with depth, location, and time. Ocean water has high and varying concentrations of various organic and inorganic substances, mainly salts. Furthermore, even small changes in temperature will induce changes in density and thus in sound speed. The greater changes



in these parameters occur in the vertical direction, since the ocean is composed of a series of layers of water with slightly different characteristics. Knowledge of these layers is extremely important for anti-submarine warfare. The variation of sound speed with depth is known as “bathythermic profile”. Special devices exist to determine it, called bathythermographs, which can be expendable (i.e. thrown overboard and lost once used), or towed. The temperature gradient tends to be less pronounced during mornings, allowing very long range detections. During afternoons, a strong temperature gradient will tend to deflect the sound downwards. Unfortunately, it is impossible to know the bathythermic profile along all the path between emitter and receiver, since it changes along that path and internal waves (between the layers) will make it vary with time.

As sound traverses across water with different densities it is reflected and refracted, giving rise to curvilinear paths. Many models and computer programs exist to determine these paths, known as ray-paths, but all these are only approximate. A specially adverse effect of these curvilinear paths, is that multiple paths may (and usually do) exist between any two points. These multiple

paths will interfere with one another, generating areas where the sound cancels out or multiplies itself.

As it travels through the ocean, the sound will also reflect on the bottom and on the surface. These reflections will further contribute to the existence of multiple paths, and can distort the sound introducing phase changes. The sea surface is continuously moving, and thus reflections on it will suffer slight Doppler shifts. The type of bottom (sand, rough or smooth rock) will induce scattering in the sound waves, and water itself is a dispersive medium, scattering sound in all directions. Finally, small bubbles of air will further interfere with the propagation, giving rise to a medium far from the linear and time invariant model that some computer programs use. A detailed account of the various aspects of sound propagation, including losses, distortions, and ray-paths, may be found in (Giellis 1983) or (Urlick 1982).

1.4.3 – Reception of the sound

After propagating in the ocean, the sound will eventually be received by the sonar's hydrophones, and processed. Sonars will usually have many individual hydrophones in order to perform beam forming and cancellation of spurious effects.

The position and quality of the hydrophones can influence tremendously the received signal. At a very low level, molecular agitation due to temperature will induce spurious noise in the hydrophones (Dyer 1998). At a more macroscopic level, pressure waves due to water movement will also induce noise. This effect, together with technological limitations in the choice of hydrophone sensors, makes it very difficult to obtain reliable measurements of sound at very low frequencies (Tims and Henriquez 1979). The static pressure on the hydrophones will also condition their performance, which generally improves as the pressure rises due to less cavitation.

Finally, one of the most important factors in the reception is the receiver's self-noise.

Isolated hydrophones, on the seabed or suspended from buoys, are naturally the ideal sensors, for they have no machinery induced self-noise, and are affected only by possible residual interference of their electronic amplifier systems or mounting mechanisms. This type of hydrophone is actually quite common, thanks to the use of sonobuoys by Maritime Patrol

Aircraft. Sonobouys come in many types and shapes, but are all basically a hydrophone, a signal amplifier, a buoy (from which the former two are suspended), a radio-frequency modulator and an antenna. They are dropped by aircraft into the ocean, usually in groups or “fields” so as to be able to perform triangulation. After hitting the water, the hydrophone itself is lowered to a pre-defined depth, and the captured signal is transmitted by radio to the listening station on the aircraft or nearby ship. For safety and security reasons, they will usually sink to the bottom after a few hours. Although we present no results in this thesis, we used some of our techniques on sonobuoy recordings made by the Portuguese Air Force.

Submarines are the next best platform for sonars. They are usually designed to be silent, and when submerged (specially at great depths) produce little hydroacoustic noise. Submarines, as surface ships, can operate two types of sonars: hull mounted sonars, and towed arrays. Hull mounted sonars are very sensitive to self noise generated by the submarine. This self noise can reach the hydrophones both through the water (using direct paths or reflecting off the surface and bottom) and through the hull itself. Towed arrays are by far the best means of receiving sound, both because they are not as influenced by the self noise and because their greater distance between individual hydrophones allows better beam forming. Unfortunately, towed arrays are quite expensive and cumbersome to operate, require very low speeds of the towing vessel, and require estimating the exact position of each hydrophone (Jesus, Felisberto *et al.* 1994; Jesus, Felisberto *et al.* 1996).

Surface ships are the worse platforms for passive sonar operation, specially when using hull-mounted hydrophones. As the hull is constantly rocking in the waves, and is by nature near the surface, hull-mounted hydrophones will suffer badly from noise induced by water movement and cavitation near them. Furthermore, surface ships tend to be much noisier than any submarine.

However, in the early stages of our project, we did perform some passive sonar recordings with hull-mounted hydrophones (using the French designed “Diodon” system that has since been deactivated by the Portuguese Navy), and obtained encouraging results (Lobo 1995).

After reception, the sound usually goes through a signal processing pipeline that will perform beam forming, to obtain directional readings, noise cancellation, and feature extraction. It can then be presented to sonar operators in a number of forms. The one that is still most reliable is as sound. Passive sonar operators go through a lot of training to be able to identify different ships.

The quality of their identifications depends critically on their skill, experience, and state of mind. It can be very difficult to distinguish between hundreds of very similar sounds, specially under the stressful conditions under which they must operate. It is therefore remarkable how a good sonar operator can accurately identify targets, even with low signal to noise ratios. Training manuals and recordings for these operators are highly classified, but exist in almost any navy, and contain a number of useful tricks and decision charts to help them. The sound can also be represented graphically on screens or paper. The most common technique is to use spectrograms, known in the submarine community as Lofargrams. These are simply successive spectra of the received signal with some sort of colour coding. As time progresses, any tonal noise will give rise to identifiable lines on the spectrogram. Changes in these lines usually correspond to changes in their Doppler distortion, thus revealing the closest point of approach to the target. Although highly classified, there is evidence that automatic identification systems are used by major navies, and it is suspected that some use neural networks, similar to those used in this thesis.

1.5 – Previous work in this area

As mentioned before, the identification of underwater noise sources has been studied for a long time. After the Second World War and with the advent of the “Cold War”, it became a very intensively researched subject. It also became a very sensitive one, and thus highly classified. Even so, some publicly available papers have appeared dealing with this subject, and we shall consider them here. We shall start by reviewing the research done on neural networks for sonar processing, since it is closely related to the work we developed. We shall then briefly see other approaches. A lot of research has been done on signal processing techniques for sonar processing, but since it is not our main concern, we will just mention some of it.

1.5.1 - Neural Networks for Sonar Signal Classification

Over the last few years, and despite a period during which anti-submarine warfare no longer seemed to be the main priority for most navies, quite a few papers have been presented where neural networks have been used to classify sonar data. Although many of them do not deal with passive sonar, which is our main interest, we will nevertheless mention them because they do provide relevant contributions.

When Rumelhart, Hinton and Williams (Rumelhart, Hinton *et al.* 1986) re-discovered the Backpropagation algorithm for training networks (it had been described 12 years earlier by Paul Werbos (Werbos 1974)), the USA Department of Defense took a great interest in neural networks, and DARPA sponsored quite a few projects in the area of neural networks. Due to the fact that the “Cold War” was reaching a peak with President Regan’s push for SDI (Strategic Defense Initiative, also known as “star wars”), most of the research effort was highly classified. This is why in the 1988 DARPA study report (DARPA 1988), the use of neural networks for sonar signal classification is mentioned, but not described in any detail.

The first paper describing the use of neural networks for sonar classification was published by Terrance Sejnowski in 1988 (Sejnowski and Gorman 1988). In that paper a system to separate cylinders (supposed to be mines) from rocks using an active sonar was described. The simple FFT coefficients of the sonar echoes were used as features for a Backpropagation network with 60 input neurons, 25 hidden layer neurons, and two output neurons. The error rate was a remarkably low 0.2%, even though human operators could do no better than 9%. These impressive results sparked a lot of interest in the field.

In the same year, a South African team (which uses submarines and sonars exactly equal to those in use in the Portuguese Navy) published a paper (Lourens) with a model for describing the cavitation around the propeller, attempted to model the gearbox, and did some classification based on AR models. Although at the time a fully automatic classification was still beyond the foreseeable horizon, the same team went on to produce some very interesting classifiers of passive acoustic signals (Lourens), that have, as ours, been used on real sonar data.

In 1990, several events led the end of the “Cold War”. With the threat of deep sea warfare largely gone, the USA DoD lost most of its interest in classification of sonar data, which had several implications: on one hand, research grants for this area were substantially reduced, but on the other, the security clearance for research was lowered, leading to a large increase in published works, and to the use of the developed technology in different areas (such as fish school classification, Autonomous Underwater Vehicle (AUV) navigation, ship traffic monitoring, etc). It is ironical, although very fortunate, that when technology was finally getting ready to meet the challenge, the main motivation disappeared. The USA DoD did however maintain an interest in the area, and recently it regained more importance, especially for identifying potential targets in coastal environments. As for smaller navies, such as the Portuguese, their interest never

diminished, for coastal warfare (also known as brown water warfare) was always their prime mission.

In 1991, the “IEEE Conference on Neural Networks for Ocean Engineering” was held in Washington DC. There was one session dedicated only to “Classification of Acoustic Signals”, where several relevant papers were presented, exploring most of the then popular neural network paradigms, such as Hopfield Networks and BAM (Van-Houtte, Deegan *et al.* 1991), Backpropagation (Casselman, Freeman *et al.* 1991; Russo 1991), Linear Vector Quantization and Radial Basis Functions (Ghosh, Chakravarthy *et al.* 1991). Some papers also presented a comprehensive comparison of different statistical and neural network based approaches (Pridham and Hamilton 1991) (Solinsky and Nash 1991). While some authors managed to have very low error rates on specific data sets (7% in (Russo 1991), down to 0% in (Pridham and Hamilton 1991)), the more realistic and general evaluation of (Solinsky and Nash 1991) showed error rates between 15% and 39%. Virtually all authors used the FFT coefficients as features (never more than 180), with some notable exceptions where wavelet coefficients were used (Ghosh, Chakravarthy *et al.* 1991). While most papers dealt with basically stationary signals (DARPA standard dataset 1), one ventured to classify transient signals (DARPA STDS-Standard Transient Data Set) (Casselman, Freeman *et al.* 1991), but still using FFT coefficients.

The DARPA standard dataset 1 was a collection of very different sounds. There were 6 different classes, two of them being short non-stationary signals, the other 4 being clear tonal sounds. Depending on the preprocessing, various feature vectors could be obtained, but (Ghosh, Chakravarthy *et al.* 1991), for example, obtained 42 training patterns and 179 test patterns, not evenly distributed amongst the classes. In this thesis, we do not compare our results with the DARPA sets because the data is too uneven to obtain what we feel is an honest evaluation of the classifier algorithm.

In the same year (Burton 1991) published a very interesting paper where he classified transient sonar signals (ice-breaking) with a vector quantizer (not Kohonen’s LVQ, but the one described in (Linde, Buzo *et al.* 1980)). He used 15 point cepstra (equivalent to 6 ms) with a 83% overlap, a rectangular windowing function (thus no window), and 64 codebooks. The data set consisted of 110 patterns evenly distributed amongst 7 classes, and using the leave-one-out technique (Breiman, Friedman *et al.* 1984) he obtained a 29% error rate.

In 1994, a paper even more similar to our work was published (Hemminger and Pao 1994) (only recently have we found out that it was published the same month (Lobo 1995) was submitted). He used 64 point Welch periodograms, upon which a Vector-Quantizing type of clustering algorithm was applied to create prototypes, followed by a supervised single layer network (named Functional Link Net – FLN (Pao 1989)). The main contribution, however, was the use of the Hausdorff Metric (Essex and M.A.H. 1990), which proved to be a very convenient way to compare spectra. The error rate was an amazingly low 4%, with a relatively low computational load thanks to the simple structure of the neural network.

The dependence of time in the acoustic signals (in this case whale songs) was addressed without much success with Time Delay Neural Networks (TDNN) by (Waibel, Hanazawa *et al.* 1989) but with very good results using a biologically inspired “habituation” pre-processor in (Stiles and Ghosh 1995).

In the 1995 International Conference on Neural Networks (ICNN95), (Fujii 1995) presented a general overview of the use of Neural Networks in Ocean Engineering, in which target identification is mentioned, even though the main focus is to help the guidance system of Autonomous Underwater Vehicles (AUV).

1.5.2 - Other related work

There has been, of course, a lot of work in signal processing, classical statistics and other branches of artificial intelligence, that attempt to classify underwater acoustical signals, or to give significant contributions towards it. A good, if general, overview of signal processing techniques applied to sonar signal processing can be found in (Dwyer 1996), containing 76 references to papers published at the “IEEE/MTS Oceans” conferences during first half of the 90’s, all of them concerning classification and detection of sonar signals.

Various papers have shown that ARMA models, or some variation of them, can provide satisfactory results in some cases. (Huang, Zhao *et al.* 1997) uses a 20-order pole model, compressing the results to a 6 dimensional vector with a Karhunen-Loève transform.

Several techniques have been proposed to extract more and clearer information from the standard spectrograms, or as they are usually called Lofargrams or simply “lofars” (Low Frequency Spectrograms). (Jauffret and Bouchet 1996) proposed a new line extraction algorithm for single

tonal lofargrams (or at least single dominant), while (Oliveira and Barroso 1999) addresses the multi-component case.

It has been shown (Tesei, Regazzoni *et al.* 1994; Lyons, Newton *et al.* 1995) that Higher Order Spectra (HOS) can significantly improve the characterization of the acoustic signature, when compared to the standard Fourier Spectrum.

The importance of a correct characterization of background noise in designing an optimal Generalized Likelihood Ratio Test (GLRT) has also been shown by (Messer 1994).

An information theory approach, using entropy and mutual information has also been used with promising results (Ren and Willis 1995; Broadhead, Pflug *et al.* 1996; Quazi 1996).

Wavelets have been used to characterize sonar signals with good results (Ho, Chan *et al.* 1996), especially when non-stationary phenomena are present.

Various methods have been suggested for breaking up the raw signals into more meaningful features, in particular, separating broadband from narrowband effects can be very useful. Some approaches use median filters to accomplish this task, while others develop optimal filters that use not only the raw signal, but information from the beamforming sub-system (Mehta, Fay *et al.* 1996).

An optimized algorithm for Target Motion Analysis (TMA), that takes into account multipath and Doppler effects was presented by (Blanc-Benon and Bienvenu 1995).

There have also been several contributions that address not only sonar based classification, but the more general task of classification based on all possible sources. (Musman, Chang *et al.* 1990) for example, devised a real time control strategy for gathering evidence for a Bayesian Belief Network, that considerably reduces the amount of computation necessary for a good identification.

The vast amount of data necessary to train a classifier, and in many cases to assist it during classification, have led to the development of specific database techniques. The British Defense

Research Establishment (DREA) has been developing a very comprehensive database (Ebbeson, Ozard *et al.* 1997), as has the Sweedish FOA (Bergsten, Schubert *et al.* 1997).

PART III

CHAPTER 2

The software used

2.1 – Introduction

During the course of this PhD program, a lot of software was written, tested, and used. Some of it was just circumstantial and will not be mentioned in this thesis, but some of it forms a useful set of tools that were very important for our work and may be used by other researchers.

We will first discuss the software written in C/C++ that forms a user friendly program to be used both by researchers and end users, and then series of Matlab routines, that are more appropriate for research only. We conclude with a brief reference to other software that was used.

2.2 – The DSOM program

When the present research project started, there was a possibility that it would be used by the Portuguese Navy in its submarine squadron. It was therefore important that all the software used could be incorporated in an *operational program* (i.e. a program that would be used in everyday operations aboard the submarine). This precluded the use of some very fine public domain software that is distributed under the GNU license, and since we did not yet have a generous budget, we could not use some also very good commercial libraries. We decided to write our own code, from scratch in C/C++, for the following reasons:

- a) We could have complete control over it. We may incorporate any part of it into whatever software we choose to develop.
- b) We would have a better understanding of various problems. Writing our own code from scratch gives us first hand experience of all aspects of the implemented algorithms. This not only weeds out potential bugs in 3rd party software¹⁰, but gives added understanding and sensitivity to the problems associated with the implemented techniques.
- c) Introducing new techniques would be easier. We expected to introduce considerable changes to existing techniques, and having written all the code makes the introduction of these techniques easier.
- d) We could make a user-friendly program, tailored to the needs of the submarine squadron personnel. It was important that even during the development phase of the project, so as to have full support and feedback from the end users, that the program be very easy to use. There are also certain tools and modes of operation with which the personnel is familiar that should be incorporated. It would be very difficult to make such a front end for very disparate software packages.
- e) We like to code¹¹. Being a PhD program in “Informatics Engineering”, we feel that is important to show some proficiency in the most basic skills of computer science: programming.

¹⁰ Even very good software packages, with excellent reputations have bugs. Matlab’s signal processing toolbox, for example incorrectly implements the well known Hamming filter.

¹¹ There is an interesting book, named “Born to code in C” Schildt, H. (1989). Born to Code in C, McGraw Hill., that has a nice explanation of this phenomenon in its introduction.

Naturally, writing our own software from scratch has the big disadvantage of being extremely time-consuming, and prone to our own bugs.

The next choice that had to be made was that of a hardware and software platform. Writing for a generic UNIX environment is the preferred choice for many research projects, since it can easily be ported to a very wide variety of machines, including very powerful mainframes and supercomputers. Unix is also a very stable system well known to most researchers, and a large selection of tested routines and software packages are available. However, UNIX presented a series of disadvantages for this particular project:

- a) The use of UNIX is not widespread amongst most end users: there are not many available machines, and there is little expertise on their operation and maintenance. In the Portuguese navy, that was meant to be the main user of the system, there are quite a few UNIX systems. These are used mainly as dedicated machines on shore based units and large ships, for control and communication systems. As dedicated systems, they would not be available for use with our software, and are not available on submarines anyway. Some years ago, many PC laptops, running a version of UNIX were distributed to patrol craft used in fisheries control, as part of a monitoring project sponsored by the governments fisheries department. Even though the software was pretty good, even simple maintenance tasks were a nightmare for the crews that had to operate them, which led to a relative disinterest and lack of use. Since the crew members are far more proficient in using MS-Windows¹² based machines, the next versions of the system were developed for this operating system, and have been used regularly. The lesson to be learned is that either the system has a professional support and maintenance team and is used as a black box by the end users, or it has to be developed in a system the end users are familiar with.

¹² We choose to use the rather old term MS-Windows, standing for Microsoft Windows, to refer to the family of operating systems produced by Microsoft that have their roots in the first graphic interface produced by the company (MS-Windows), namely MS-Windows 3.11, Windows 95, Windows 98, Windows ME, Windows NT, and Windows 2000.

- b) The graphical interfaces are not very well standardized, and developing software for them can be quite troublesome. Even though there is widespread support for the X-Windows interface, slightly different versions and implementations are used. Thus porting to other platforms is seldom trouble free. Although there are quite a few good toolkits for developing interfaces for X-Windows, the equivalent tools for the MS-Windows environments are generally easier to use. Although too late to have been used in this project, an interesting convergence of the tools available for both environments is the Kylix system by Borland. This Pascal based development system allows, to a certain extent, the development of programs with graphical interfaces that run both under UNIX with X-Windows, and MS-Windows, and is very easy to use since it is based on the popular and very intuitive Delphi system.

On the other hand there were a few good reasons for developing the software in a MS-Windows environment, and in particular for Windows 9x (Windows 95 and Windows 98):

- a) Widespread availability of machines and expertise. All ships in the Portuguese Navy have some PC running MS-Windows aboard. In most cases, these PCs are available to be used, as a secondary mission, to run our software. Thus, the critical problem of getting money for machines to test and prove our system could be avoided. Moreover, MS-Windows based machines are available in large quantities at the Naval Academy, at the New University of Lisbon, Portugal, or any other school.
- b) Easy to use development tools. There are a number of very good development systems available for MS-Windows, and there is widespread support for sound-card programming (which would be a requirement).

The final program proved to be quite reliable, and could perform classification in real time even on some early Pentium PCs, running the original Pentium at 75 MHz clock speed. Due to the fact that it runs on inexpensive and widely available computers, and can use any TCP/IP networked computers to form a PVM cluster, we feel it can have a wide application both in educational institutions (where computer classrooms are available), and in commercial enterprises (where the networked office computers or points of sale are networked and largely unused during nights).

2.2.1 - Overview of DSOM

The program that was developed, was called DSOM for Distributed Self Organizing Map. It had the following requirements, that where all met:

- a) Record data from the sound card. We found that, although there are differences between sound cards, most of them had quite reliable linear transfer functions in the frequency ranges that we would be using. Thus, the program should use the MS-Windows Multimedia Interface to control the sound card and read data directly from it.
- b) Show a spectrogram of the sound being recorded in gray scale. This was required to give the end users an interface they were familiar with. With it, we were able to check that the recording was being done correctly, and a sliding ruler was provided to identify the exact frequency of any feature that turned up on the spectrogram.
- c) Read and write data from sound files (in the Microsoft .WAV format), and from feature vector files in the format used by Kohonen's SOMPAK (Kohonen, Hynninen *et al.* 1995).
- d) Read and write SOM maps in the format used by Kohonen's SOMPAK (Kohonen, Hynninen *et al.* 1995).
- e) Train SOMs either on a single computer, or over a distributed cluster of networked computers. For reasons explained in part I, we chose to use PVM as the distributed platform. The possibility of training a SOM over various computers made training very large SOMs a possibility. Under operational conditions, this could mean using all the vessel's computers to re-train a network during missions, and ashore it meant being able to train the large SOMs in reasonable time.
- f) Show the user a color coded 2-dimensional SOM, highlighting the currently selected winner neuron.
- g) Generate U-Matrices of the SOM maps.
- h) Prune a SOM using Q-Sets.
- i) Be as simple as possible to use by non-trained personnel in operational conditions. This required that with only a few keystrokes, a end user should have a meaningful output from the program.
- j) Be usable for research purposes. This require that various aspects of the software be as customizable as possible, to accommodate experimentation with different techniques. It also required that the core routines, including those involving the PVM interface, be well

documented and written in generic C/C++ so that they could easily be ported to another operating system, namely Linux.

It was decided to use the data format standardized by SOMPAK (Kohonen, Hynninen *et al.* 1995), for both the data pattern files and the SOM maps. We will refer to this format as “the Kohonen format”, and detailed explanation of the format is available in the SOMPAK documentation. Basically all files are text files, where the first lines contains the number of features (and map characteristics if it is a map), and the next lines contain one pattern each, each feature separated with a space, and the optional label as the last value. Comment lines are available, and start with a “#”. We used these comment lines to include information specific to our implementation, namely the type of distance function to be used. Thus, if we want to use a non-Euclidean distance, the second line of the file must be a comment with the information as to which function to use (as we will see later, only Hamming distances are currently available).

The program was written in Borland “C++Builder” a visual development system for MS-Windows that is very similar to the popular Borland “Delphi”, with which most people involved

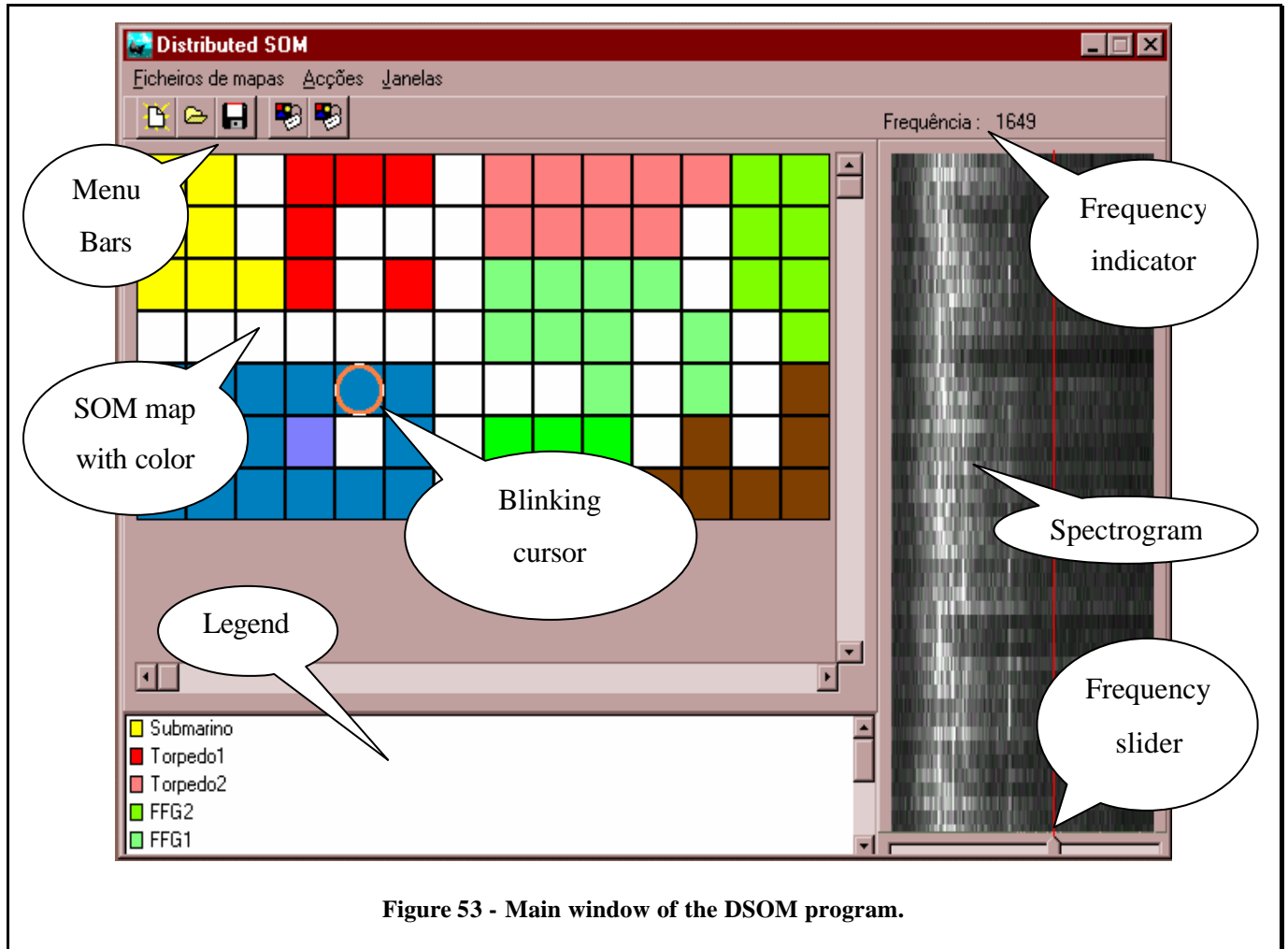


Figure 53 - Main window of the DSOM program.

in the project were familiar with, but that uses C++ instead of Pascal as its base language. The PVM version used was the one developed in the University of Coimbra, that is fully compatible with the original version for UNIX (at the time we started, the other version wasn't even available for MS-Windows).

The task of actually writing and program was almost entirely done by Nuno Bandeira, then in his last year of undergraduate, as his “final project”¹³. His merit and very hard work cannot be overstated, as it demonstrated not only great proficiency in C/C++ and using the sound card windows API, but also programming and debugging PVM with very little in the way of support tools. The following year, Raul Moizão coded the generation of U-Matrices and the Q-Set simplification, adding on his initiative some enhancements to the U-Matrix visualization. A separate auxiliary program, used to initialize the SOMs and named “Drandinit” was written by Cadet Almas. The author of this thesis also wrote a few of the routines, but mainly set our requirements and design guidelines and helped with debugging and testing. The development process is described, in Portuguese, in the students final reports (Bandeira 1996), (Moizao 1997), and (Almas 1998).

The main screen of the program can be seen in Figure 53, and appears as soon as the program is started. The screen is divided into 4 main areas:

- a) Top - **Menu Bars**. The top of the window has the menus which give access to all the program's options, including secondary windows. The most used options (creating a new map, opening a trained map, saving a map, configuring a map, and configuring the spectrogram) are available also with “fast buttons” under the menu bar. Although the meaning of each option in the menu bars is self-explaining, we shall discuss them later.
- b) Center-left – **SOM visualization**. Each unit of the map is represented by a square in this area. The units that have labels are color coded, according to the legend available just below. When the program is running in real time classification mode, the winning neuron is shown by a blinking circle. Left clicking this area will give available

¹³ In the New University of Lisbon, as in most science and engineering schools in Portugal, undergraduate students are required to do what is called a “final project”. This “final project” will require most of their time during a semester to a year, and is treated like a graduation thesis.

information on the underlying unit (its label and coordinates), while right clicking pops up a menu that allows us to re-scale and re-draw the map.

- c) Bottom-left – **Legend**. This area contains the color code legend used in the map. Right clicking this pops up a menu with legend related tasks, such as gathering legends from the SOM, changing the color a given label, creating/editing/deleting labels, and finally loading and saving legend files.
- d) Right – **Spectrogram**. This area shows the spectrogram of the sound being received. The spectra are gray-scale coded, with white representing the highest intensity. Left clicking in this area gives the option to start/stop processing the sound from the sound card, and gives access to a menu with the signal processing options. A sliding ruler is available, that can be moved with the mouse, and gives the exact frequency value (shown above) of any area of the spectrogram. As the processing goes on, the spectrogram slides upward, with the most recent spectra at the bottom.

2.2.2 - Main menu bar

The main menu bar has the following options:

- a. File menu (FICHEIROS DE MAPAS)
 - i. Open (ABRIR) – Opens a SOM stored in Kohonen format.
 - ii. Save (GRAVAR) – Saves a SOM in Kohonen format.
 - iii. Save as (GRAVAR COMO) - Saves a SOM in Kohonen format with a different name.
 - iv. Close (FECHAR) – Stops using the loaded SOM.
 - v. Exit (SAIR) – Terminates the program.
- b. Actions (ACÇÕES)
 - i. Distribute (DISTRIBUIR) – Distributes the map amongst a PVM cluster, sending neurons to other machines. A dialog box will ask how many machines should be used, and the name of the executable in those machines (default="client.exe")
 - ii. Recall (RECOLHER) – Recalls neurons from the PVM cluster, ending the distribution process.
 - iii. Train (TREINAR) – Pops up a dialog box with the various options for training the SOM.

- c. Windows (JANELAS) – Gives access to windows with other functionalities. At the moment.
 - i. Patterns (AMOSTRAS) – Pops up a window that allows processing of data files with patterns or with recorded sounds. This is intended for off-line data processing.
 - ii. U-matrices (U-MAT) – Pops up a window that allows the generation and visualization of different U-Matrices.

In the version used for research purposes, a fourth menu entry was added, named “ABOUT”, that has information about the authors and version numbers.

2.2.3 - Pattern Window

This window allows off-line processing of both data pattern files and sound files in Microsoft’s .WAV format.

The top menu bar gives access to the various options, the middle edit box indicates which data file is being used, and the bottom part shows the labels of each individual pattern.

In the label window, one may select one or a group of labels. Right clicking on the labels pops up menu that allows:

- a) Editing the label.
- b) Visualizing the spectra of that label.
- c) Classifying that pattern on the SOM (also Ctrl-C).
- d) Deleting that pattern.

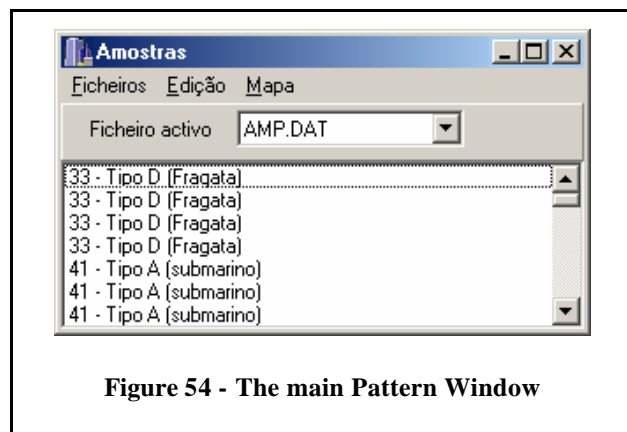


Figure 54 - The main Pattern Window

The FILE menu (Ficheiros), allows for the traditional file operations with data files in Kohonen format, namely Open, Save, Save As, and Close

The EDIT menu allows editing the selected labels (as can be done by right clicking them), and allows 2 options to work with sound files:

- a) Inserting a WAV file. The WAV file will be read and processed according to the selected options, to produce a data pattern file in Kohonen format.
- b) Recording a WAV file. This will produce a WAV file with the sound recoded from the sound card. It is possible to hear the sound before or after saving it to disk.

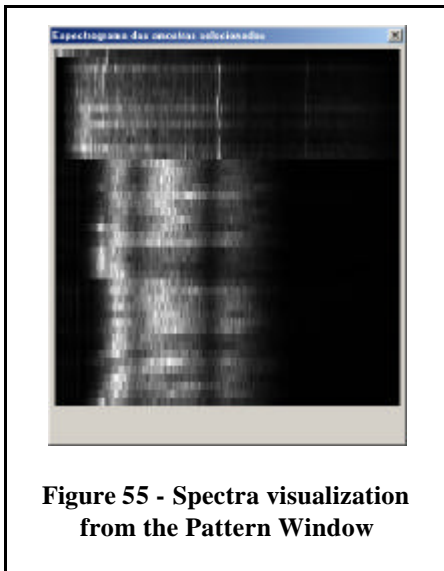


Figure 55 - Spectra visualization from the Pattern Window

The MAP (Mapa) menu has the following options:

- a) Train a SOM. This brings up the standard training dialog box.
- b) Calibrate. This will label the SOM according to the labels of a given set of patterns. For each unit of the SOM, the labels of the patterns that have it as the winner unit are recorded, and the label given to the unit is the most occurring label amongst these.
- c) Classify. This will classify a single pattern (as can be done by right clicking the label).
- d) Classify all. This will classify all selected patterns, and if they have labels, it will calculate the **error rate** (i.e., the percentage of cases when the patterns label was different from its winning unit label).

2.2.4 - Training dialog box

The training dialog box is accessible both at the main window and at the pattern window. It is assumed that the SOM training will always occur in two phases, described in (Kohonen 2001), and parameters for each of the phases can be introduced, as can be seen in Figure 56.

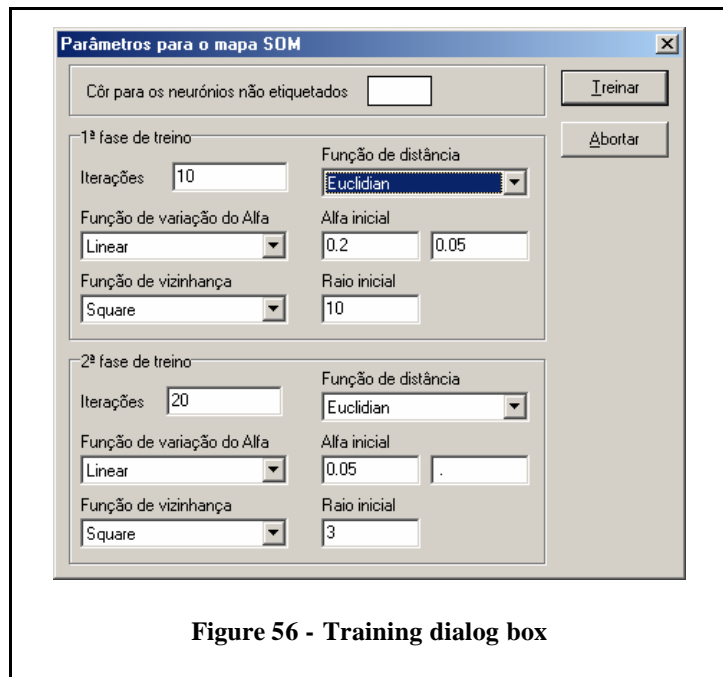


Figure 56 - Training dialog box

The top area defines the default label color to be assigned to the

units. The two areas concerning the 1st and 2nd phase of the training process contain the same information.

The first dialog box defines the number of iterations (ITERAÇÕES). It must be noted that, contrary to Kohonen's convention, where an iteration is considered to be the presentation of a single pattern, each of our iterations is a run through the entire training set. Accordingly, the learning parameters are updated only when a complete run through these patterns is performed. As a consequence, the number of iterations used by DSOM will always be much smaller than the equivalent number of iterations in SOMPAK.

The distance functions currently available (FUNÇÃO DE DISTÂNCIA) are the Euclidean distance and the Hamming distance. When using the Hamming distance, the random bit update rule is used, as described in part II of this thesis.

Although there is a dialog box to change the rate of change of the alpha parameter (Kohonen 2001) (FUNÇÃO DE VARIAÇÃO DO ALFA), only a linear variation is currently possible.

The two neighborhood functions available (FUNÇÃO DE VIZINHANÇA) are “Square” (equivalent to Kohonen’s bubble), and “BinSquare”, which is similar but should be used for binary maps.

Finally, two dialog boxes allow the introduction of the initial learning rate (alpha), and neighborhood radius. It must also be noted that this implementation will force the radius down to 0, so that in the last iteration only the winning neuron is updated. This feature will compensate the annoying outward boundary distortions that the traditional SOM has. Thus, even the units on the extremes of the map will be correctly centered amongst the patterns that have them as winners.

2.2.5 – Differences between DSOM and SOMPAK 3.11

Summarizing what has already been discussed, the main differences between DSOM and SOMPAK 3.11 are (besides DSOM’s extra features):

- a) The way iterations are counted. SOMPAK considers one iteration to be the processing of a single pattern, while DSOM considers the processing of the entire training set to be an iteration. Thus, the parameters are updated more often in the original SOMPAK
- b) The final radius of the neighborhood. SOMPAK makes the neighborhood radius converge to 1, i.e., even in the last iteration some neighbors of the best matching unit will be updated. DSOM makes the neighborhood converge to zero, i.e., in the last iterations, only the best matching unit will be updated. This will reduce the border effects that can be observed in the original SOMPAK.
- c) The comments in the datafiles. DSOM uses the comment lines of the datafiles to encapsulate information about its extra features, such as different similarity measures. Since these lines are only comments, they will be ignored by SOMPAK.

2.3 – MATLAB routines

MATLAB, a shortname for MATrix LABoratory, is one of the most used scientific computing programs. It started as an interpreter that used the LINPAK library of matrix manipulation

routines, but has since grown into a full fledged development system. MATLAB is produced by Mathworks Inc (www.mathworks.com), and is available for UNIX, MS-Windows, and Macintosh (though the Macintosh version has been discontinued). The latest version, at the moment, is version 6.1, that for internal reasons is also known as Release 12. The current version includes, amongst other things, a compiler (to avoid run-time interpretation of the code), a context sensitive editor, a visual debugger with an object inspector, and an excellent Graphic User Interface (GUI). Since its start MATLAB has boasted very good data graphics, and a large variety of MATLAB toolboxes are available from Mathworks for specific areas, such as the Neural Network toolbox, the Financial Toolbox, the Fuzzy Set Toolbox, etc. There are also many toolboxes written in MATLAB by researchers, that are freely available. Of these, we must mention for SOM related tasks the SOM Toolbox for Matlab available “<http://www.cis.hut.fi/projects/somtoolbox/links>”, that was developed by Kohonen’s group, and contains links to many other Matlab packages. For classification tasks, we would recommend NETLAB, available at www.ncrg.aston.ac.uk/netlab, that contains most of the code necessary to solve the problems proposed by (Bishop 1995), and it’s companion book (Nabney 2001) that contains additional exercises and examples.

We started to use MATLAB when the requirement to build an operational program disappeared (due to uncertainty as to the future of the submarine squadron). The main reasons to use MATLAB where:

- a) Prototyping is much faster using MATLAB than using C. The language is more powerful (since it is much more high-level), and the interpreter allows a interactive testing and design of code. In fact, a lot of useful processing is done interactively without ever writing a program.
- b) The graphical outputs of MATLAB are very good and easy to use.
- c) It has a huge amount of toolboxes and routines readily available.
- d) Code developed in MATLAB can be incorporated in stand-alone C programs, using the MATLAB compiler and run-time library. While the code may not be as fast as pure C code, it is still amazingly fast, specially with matrix manipulations, due to very stable and well developed libraries.
- e) It is easy to share MATLAB programs with other researchers. MATLAB does not require compiling, and when the code is properly written, it can safely and easily be used with little or no knowledge of its inner workings.

A lot of our software was originally written for version 5, but runs perfectly on the latest release. Some minor (but very annoying and tiresome) adjustments had to be made to some routines, since the version 6 Delauny triangulation does not reorder the triangles as version 5 did.

All the routines developed by us contain help information. This includes a one line description of the purpose of the routine, a specification of input and output parameters, comments on its internal workings when necessary, and always version and author information for configuration management. Some of the MATLAB batch files we used are also provided as examples.

The complete listing of the functions is given in Appendix E, and we will only list the filenames and purpose of each routine, grouped by purpose.

Q-set related routines

qs_mat_build.m	Build the positive only Q-sets given a set of candidate prototypes and a set of training patterns. The Q-sets are given as a Boolean matrix.
qs_select_heuristic.m	Select prototypes using the positive-only heuristic described in chapter 2 of part II, given the positive only q-sets. This function produces only the numbers (or indexes) of the prototypes to select.
qsgc_mat_build.m	Build the general case Q-sets given a set of candidate prototypes and a set of training patterns. The Q-sets are given as two matrices. The first contains the indexes of the prototypes sorted by proximity, while the second is a Boolean matrix indicating whether they have the same class or not.
g2p.m	Transform general case Q-sets to positive-only Q-sets.
qs_select.m	Select the minimum set of prototypes for positive-only Q-sets, using branch and bound.

Prototype minimization routines

cnn.m	Select prototypes using the CNN rule.
rnn.m	Select prototypes using the RNN rule, given a set of prototypes selected by the CNN rule.

Graphic routines

voronoi_boundary.m	Plot the Voronoi boundary between classes, given the 2-dimensional prototypes of those classes .
class_plot.m	Plot a set of 2-dimensional patterns, using different markers for each class.

Classification and validation routines

knn.m	Classify a pattern using the k-nearest neighbor rule. The routine also produces a vector with the estimated probabilities of that pattern belonging to each class
knn_mat.m	Classify a set of patterns, using the k-nearest neighbor rule
confusionMatrix.m	Calculate the confusion matrix, given the true classes and the assigned ones.
selfClassify.m	Classify each pattern in a given dataset using all other patterns as prototypes, with the nearest neighbor rule
splitData.m	Produce a matrix to split a given dataset into training and test sets for cross-validation.
buildTrainTestSet.m	Build training and test sets, given a matrix produced by "splitdata.m".

Miscellaneous

read_koh.m	Read a file in Kohonen's format (Kohonen, Hynninen <i>et al.</i> 1995).
write_koh.m	Write to a file in Kohonen's format (Kohonen, Hynninen <i>et al.</i> 1995).
remove_col.m	Remove a column from a matrix.
generate_2D_uniform_data.m	Generate d-dimensional data with uniform distribution in a given rectangle.
generate_double_f.m	Generate data for the double F problem (Hart 1968).
generate_straight.m	Generate data for the straight line problem described in chapter 5 of part II.
spectra_wavfile.m	Calculate spectra of data contained in a Microsoft WAV format audio file, using windowing and averaging.
mHamming.m	Correctly implemented Hamming window
hausdorff.m	Calculate the Hausdorff distance between two sets of points (or spectra)
findPrimes.m	Find prime implicants, using Quine-McClusky's method

2.4 - Other software

Besides the abovementioned software that we ourselves developed, we used quite a lot of software that is provided for free use amongst researchers. We will now mention the 2 software packages that were more important for this thesis, and that will certainly be useful for other researchers.

2.4.1 - SOMPAK

The SOMPAK has been developed at Helsinki University of Technology by Kohonen's team at the Neural Networks Research Center. The most popular version is version 3.1, available at <http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml> , that has fairly good documentation (Kohonen, Hynninen *et al.* 1995). It consists of a series of programs that run in command line mode making it very easy to create batch files to perform a given processing sequence. The software is written in C, and compiled versions are available for MS-Windows based environments and for Linux. Since the code uses very standard C with very few system calls, it is easy to compile under any other system.

The original documentation provides a good tutorial on how to use the system, but additional information is available at SOMPAK unofficial site (<http://www.cis.hut.fi/~hynde/lvq>), maintained by Jussi Hynninen, that includes information on how to modify SOMPAK, and recent developments. An additional work-through tutorial with a simple example is available in (Lobo 1998).

Included in the package, are the following programs:

- randinit* – Initializes a SOM.
- vsom* – Trains a SOM.
- qerror* – Calculates the quantization error of a map for a given set of patterns.
- vcal* – Calibrates a SOM, assigning labels to the units, based on a file of patterns.
- visual* – Generates a file with the SOM mapping coordinates of each pattern of a data file.
- Sammon* – Generates a postscript file with the Sammon mapping of the patterns.
- planes* – generates a postscript file with the gray-scale coded weights of one of the features of the map.

umat – Generates a postscript file with a gray-scale U-Matrix of the SOM.

2.4.2 - RoughSetLab

For feature selection using Rough Set Theory, we used a program called RoughSetLab, that runs under UNIX. This program started as a front end to the publicly available Rough Set Library (RSL). The Rough Set Library is a collection of C routines and conventions, to perform various tasks necessary to apply Rough Set theory. The original RSL was developed by M.Gawry, M.Modrzejewski, and J.Sienkiewicz, and documented in a Users Manual (Gawrys and Sienkiewicz 1993) and a research report (Gawrys and Sienkiewicz 1994). It has been continuously upgraded to reflect recent development in Rough Set theory, but since its main purpose is to provide researchers with C routines, it does not have a front end for users. Thus, a few different front ends were developed, including GROBIAN (<http://www.infj.ulst.ac.uk/~ccc23/grobian/grobian.html>), and the RoughSetLab that we used.

RoughSetLab was originally developed as a MSc. Thesis by Frank Muller (Muller 1993), under the supervision of Prof. Roman Swiniarski. Later other students under the supervision of Swiniarki made considerable changes. These included re-writing the core functions in C++, and thus separating it from the RSL library, providing a menu system to assist the users (making its use almost trivial), and making a X-Windows interface that uses MOTIF. For this thesis we used the menu driven version.

Rough set lab allows users to:

- Load a discrete information system
- Load discretized data set, and define a discrete information system
- Load a real-valued data set, and produce a discrete information system. To do this, the number of discretization levels must be given. The program will calculate the various parameters to perform that discretization.
- Find a relative reduct using heuristic.
- Find all relative reducts, using a complete search.
- Find all relative reducts smaller than a given one (using branch-and-bound)
- Find the core.

The Roughsetlab software is kept by Roman Swiniarski at San Diego State University (SDSU).

PART III

CHAPTER 3

The Submarine data

The author of this thesis has been cooperating with the Portuguese Navies Submarine Squadron on ship noise recognition, since 1990. The project started with the construction of a microcontroller based acquisition system, and proceeded with the first digital recordings performed by the submarine squadron personnel and the author. The submarine squadron had been doing analog recordings on tape for a long time, and some of those recordings were cataloged and digitized. Thus, a digital database with “hydrophonic effects” (the technical term used to refer to underwater sound) started to be constructed, and is now kept for military purposes. Most of the data, and all the target and environment information associated with each

recording are classified. However, we were allowed to use some of the recordings for research purposes, provided we kept some basic safety precautions.

All the recordings were performed by the submarine's passive sonar equipment. Some of the data was recorded by high quality tape recorders and then digitized ashore, while other were digitized aboard with PCs. Unfortunately, the exact frequency bands used, together with other important details cannot be discussed in this thesis.

3.1 - First experiments

The first research done with those recordings was the base for a MSc. thesis (Lobo 1995), and a conference paper (Lobo and Moura-Pires 1995), where the experiments are described in detail. The data then consisted of 10 recordings of 6 different ships (two of the ships had 3 separate recordings). Power spectra of small portions of these recordings were calculated, producing a total of 210 patterns with 2048 features each. The main reason for using so many features was that we knew from prior knowledge that the frequency resolution attained with that many features was necessary, and we didn't know which frequencies could be discarded. Another reason was that we were trying to harness the ease of use of a SOM, even with many features, to compensate a lack of feature extraction techniques that required deep expert knowledge in real ship acoustic signatures, that we did not have.

So as to validate the results, we used a class based leave-one-out technique (Breiman, Friedman *et al.* 1984), and obtained 21 training/test sets, each with 200 training patterns, and 10 test patterns, one from each of the recordings. We then repeated the processing steps for each of these training/test sets.

necessary for operational use. Thus the DSOM program, described earlier, was developed, to provide a real-time, user-friendly environment, with the possibility of training large maps over networked computers. The performance of the distributed version of SOM has been discussed in chapter 3 of Part II, and being distributed does not have any impact on classification accuracy, so we will discuss the advantages of distributed processing no more, even though it was used in many experiments. The real-time and user-friendly features of the program will also have no impact on the classification accuracy, and these feature by themselves have also been seen earlier.

Thus, and although the tools were crucial, we shall concentrate only on the classification results obtained.

3.3 – Broadband vs. Tonal identification

As was seen in chapter 1, the noise generated by a ship moving in the ocean can be divided into broadband components, and tonal components, that have quite different causes and behaviors. The way we were using our SOMs, they were sensitive mainly to the broadband noise. This happens because the broadband signal will, by its own nature, span a large number of bins, contributing heavily for the distance measure (which was Euclidean). On the other hand, the tonal components will have an impact on very few features, ideally only one, and thus contribute little to the distance measure.

We decided to use feature extraction techniques to separate the two components, and design separate SOM-based classifiers for each set of features. We then compared the performance of each of these classifiers. When considering tonal noise, we did not know whether the frequency at which the tonal noise occurred was the best feature, or whether the actual amplitude would also be useful, so we tried both approaches.

To compare these techniques, we selected 5 ships available in the database, and extracted 33 patterns for each ship, producing a total of 165 patterns. Each pattern was the 2048 bin power spectrum calculated over a certain time period, using a 4 segment Welch periodograms with 50% overlap, and a Hamming Window, as described in chapter 2 of part I. So as to be able to validate the results, we divided the available data into 11 training/test sets, so that in each of the test sets

we had 3 different patterns of each class. Thus, each training set had a total of 150 patterns (30 of each class), and each test set had a total of 15 patterns (3 of each class).

3.3.1 - Extraction of the tonal signal

The general idea of extracting the tonal signal, is to estimate the broadband signal for each point in frequency, based on the frequency neighborhood, and then set a threshold above which we consider that value as an outlier, and thus part of the tonal signal.

The first step to extract the tonal signal was to obtain the reference broadband signal for each frequency bin. This was done by calculating for each bin, the median power, based on a 64 point neighborhood around that point. This width of neighborhood was reached by experimentation: we steadily increased the neighborhood radius, and visually inspected the spectra obtained, until plausible (smooth yet detailed) broadband spectra were reached.

Assuming that at each point in frequency, the broadband signal is contaminated with white Gaussian noise, that our estimate is unbiased, and that each of the four segments is statistically independent, the spectral density estimate will follow Chi-square statistics with 4 degrees of freedom. Under these conditions, we can impose a desired confidence level to find outliers, and obtain the corresponding threshold value (Kay 1988). If we impose a 99% confidence value, we will obtain a

value of 7dB. Thus, we considered any value more than 7dB above the median as a separate tonal signal. On average, applying this process to the data yielded 0.81% of non-zero components, and a visual inspection of this spectra (see Figure 58) showed that they were plausible tonal signals of the ships.

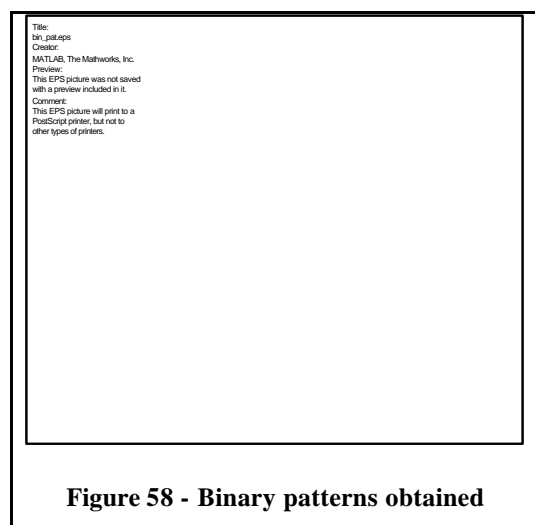


Figure 58 - Binary patterns obtained

3.3.2 – Experimental comparisons

From the original dataset we now have 3 datasets with different features:

- a) **Full** patterns – The original patterns with 2048 features representing the full spectra. This dataset was used to train a standard SOM, using Euclidean distances.
- b) **Amplitude** tonal patterns – The original patterns with the broadband signal removed. The tonal components kept their amplitude values. This dataset was used to train a standard SOM, using Euclidean distances.
- c) **Binary** tonal components – The tonal components of the spectra, with a value of 1 where the amplitude was different from zero, and 0 otherwise. This dataset was used to train a binary SOM, described in Part II.

After training and labeling a SOM, with 10x5 units, for each of the 11 training sets of the 3 datasets, we obtained the results shown in Table 10 and Table 11.

Dataset	Error rate in each training set											Average	σ	Min.	Max.		
	0	1	2	3	4	5	6	7	8	9	10						
Full Patterns	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Amplitudes	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Binary	0,0	2,6	0,0	0,0	1,4	1,1	0,9	0,0	0,7	0,0	0,0	0,6	0,8	0,0	2,6		

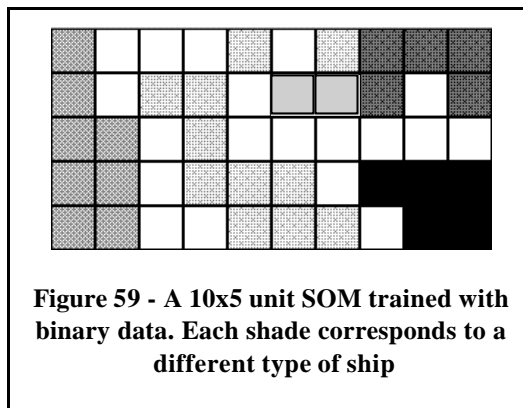
Table 10- Error rates in the training sets

Dataset	Error rate in each test set											Average	σ	Min.	Max.		
	0	1	2	3	4	5	6	7	8	9	10						
Full Patterns	0,0	0,0	0,0	0,0	6,7	6,7	20,0	6,7	40,0	0,0	0,0	7,3	12,5	0,0	40,0		
Amplitudes	13,4	13,4	0,0	6,7	0,0	6,7	0,0	13,4	6,7	0,0	0,0	5,5	5,9	0,0	13,4		
Binary	0,0	1,1	6,7	0,0	0,0	0,0	0,0	6,7	6,7	0,0	0,0	1,9	3,1	0,0	6,7		

Table 11 - Error rates in the test sets

An example of the SOMs obtained is presented in Figure 59.

From the experimental results, we may take a few conclusions:



- a) The standard deviations are extremely high. On one hand, this could be explained by the relatively few number of training/test sets, and the existence of some “anomalous” error rates for some of these sets. Nevertheless, it does reveal that the results obtained with this data are very unstable, and should be used with caution.
- b) The error rate for the training set dose not reach zero when using the binary features. If the classes are not separable, this is desired result, since it reveals that the learning process is not overfitting the data. However, if each pattern is unique, and no 3 patterns are collinear (which cannot occur with binary data), the data must be separable, since we have more features (2048) than patterns (165). The error rate could also be non-zero due to insufficient training, but more training would not lower the error rate. A close look at the data, revealed that in fact the patterns where not unique. This means that either the tonal signal of different ships was identical. Due to the many differences in machinery aboard different ships, this probably means that the method of extracting the tonal signal, despite being useful at it is, could be improved.
- c) The use of the amplitudes of the tonal signal brings no improvement over the use of the simple binary features. Since processing real-valued features takes considerably more resources than binary valued features, the use of the amplitudes of the tonal signal was abandoned.
- d) The results obtained with the binary features were significantly better than those obtained with the full spectra. This shows that the tonal signal is in fact an important feature for the problem at hand, and should be further explored.

It must be noted that some of work done on classification of ship noise mentioned earlier also use the tonal signal to perform classification. (Meister 1993), for example, selects the most significant peaks in the power spectrum, and uses their frequencies and amplitudes as inputs to a Backpropagation Neural Network. For our objective, this method has two main drawbacks: it forces us to select a fixed number of frequencies from the spectrum, and it is not a clustering

method, which we want so as to do some exploratory analysis of the data. Also, as we showed experimentally, the presence of a spectral line seems to be a much more reliable feature than its relative amplitude. This can be explained because the actual amplitude can be strongly distorted by multipath interference.

3.4 – Clustering on a large dataset

To be of practical use, the system we were designing had to be able to work with very large databases, containing many ships. To test the system, both methods and available data, under these circumstances, we selected 2 hours of recordings, containing 33 different ships or types of ships. From these recordings, we extracted 2342 patterns, containing spectra with 2048 frequency bins, obtained using Welch periodograms with 50% overlap and Hamming Windowing. The actual number of patterns available for each class differed considerably, as would be expected from a practical situation, where it is easy to record data from friendly ships, but hard to obtain data from others.

We used the DSOM program to cluster this data with a 30×20 unit SOM, and obtained the U-Matrix shown in Figure 60.

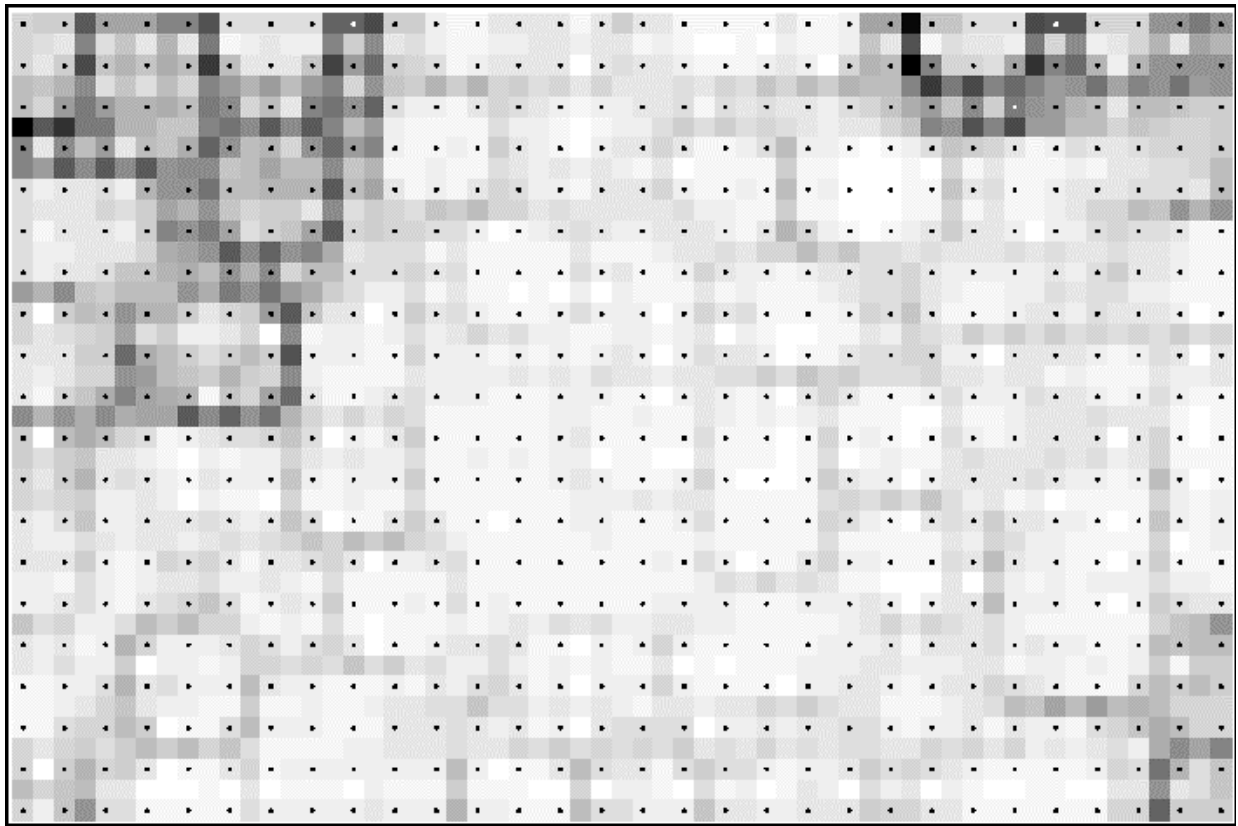


Figure 60 - The U-Matrix obtained after clustering the 33 ship dataset with a SOM.

A careful inspection of the U-Matrix shows 38 reasonably bounded areas, and thus 38 clusters. When labeling the SOM, almost all units were winners for a single class of ship, and only a few had a mixture of classes, and thus a label chosen by majority vote. This led to a 2.4% error rate in the training set, corresponding to 57 patterns. Given the very large amount of data, and the fact that its quality (signal to noise ratio) was sometimes quite bad, the results are remarkably good.

It can also be observed that some classes of ships, namely those that had several recordings, had more than one cluster, corresponding to different environment conditions during the recording. This explains why we obtained 38 clusters for only 33 classes.

Three of the corners of the U-Matrix have very well defined borders, and are quite different from the rest. The upper left corner corresponds to various types of very large merchant vessels (tankers, bulk carriers), that do indeed have a very distinguishable acoustic signature. The lower left corner, corresponds to torpedoes, whose signatures had also proved to be very different from the rest in our first experiments. Finally, the upper right corners contains ships that were recorded

under circumstances that are unusual for the recording platform, and thus are mainly due to considerable changes in the self-induced noise.

3.4.1 – Fusing information from the standard SOM and the binary SOM

We repeated the experiment with the binary version of the SOM, and as would be expected from the previous section, the error rate in the training set was considerably higher. However, even in these circumstances, as long as a classifier performs better than random choice, it can be used to improve the performance of another one (Schurmann 1996; Gama 2000; Alexandre, Campilho *et al.* 2001; Demirekler and Altincay 2002). Since we only have two classifiers (standard SOM and binary SOM), we would only obtain confirmations or ties if we used their final output to vote for the final decision. Thus, we used the following method:

- a) When labeling the SOMs, keep the classes of every training pattern that has a given unit as its winner in the units label. The label thus ceases to be a simple class, but will be a vector with the number of patterns of each class. As discussed previously, this can be taken as an estimate for the probability density at that point.
- b) When classifying, use the label given by the standard SOM, if only class is represented in it. Elsewise (i.e. if the standard SOM is not 100% sure of the class), use the label vectors of both the standard SOM and the binary SOM to decide on the class.

Using this method, the error rate on the training set went down to 1,9%, thus improving slightly the results obtained.

Due to the fact that we were already using all the available data, we could not estimate the error outside the training set. Due to the fact that some classes had very few patterns, a leave-one-out cross validation, besides being extremely time-consuming, would be very biased. Thus we were satisfied with the results obtained, and proceeded to perform experiments with a larger, if slightly less realistic set of recordings.

PART III

CHAPTER 4

Acoustic Tank Data

4.1 - Introduction

The data obtained by the submarine squadron, undoubtedly the one with real practical interest, are not the best to conduct experiments on, or to use in a thesis such as this. This happens mainly due to 3 reasons:

a) Security classification

The data collected by the submarine squadron has great military value, and thus is classified. Some of the results obtained with those data are not classified, and have been

presented earlier, but the raw data itself, and many intermediate and final results remain classified. Thus, in this thesis, we were not able to present all interesting results, and worse than that from an academic point of view, we cannot provide means for other researchers to replicate our results.

b) Uncontrolled environment

Having been recorded at sea, on an “opportunity basis” as a “secondary mission”, over a large span of time, and by different operators, the submarine data often lack a detailed description of the environment and scenario of each recording. The complete description of the environment and scenario is all but impossible in real life situation, since it requires not only a very thorough observation of all variables involved, but an efficient and fluid communication between the recording vessel and the “target”. Indeed, sometimes the actual user-given classification might be wrong, given conditions under which those classifications were made. Thus, it is very difficult to isolate the contribution of different factors towards the performance of the system.

c) Scarcity of data

Although the submarine database is already quite large, there are relatively few recordings available for each separate ship, resulting in a low statistical significance for the results. Those data have to be enough for designing an operational classifier, but an academic project should try to demonstrate statistical significance.

Thus, it was decided to perform recordings on unclassified “targets” in controlled environments. As mentioned earlier, we did not want to use computer-generated data, and the next cleanest data that we could obtain would be in an acoustic tank. Such a tank exists in one of the departments of the Portuguese Navy’s shipyard (named “Arsenal do Alfeite”), where it is used to calibrate sonars. Every year, in July, that tank is emptied for cleaning and maintenance. Thanks to the good will and collaboration of a few officers (namely Captain Ferreira de Sousa, and, Lieutenant-Commander Deusdado), we obtained permission to use the tank for a week in July 1999, just before the tank was to be emptied. We could set up our equipment, as long as we could make the tank available for its main purpose in a few hours if it were necessary. This situation actually occurred one day, when a ship’s sonar array was brought for a quick fix, and we were pleased to find out that it was easy to accommodate both tasks.

A complete and very detailed report of the preparation and recording of the signals is available as a technical report from the Naval Academy, but is written in Portuguese (Lobo and Oliveira 1999). The report is also available in a 3CD set, that contains all the data along with programs to help process it, and various information.

4.1 - Data gathering

4.1.1 - The tank and recording equipment

The acoustic tank is an anechoic tank built under the supervision of the now Rear-Admiral Silva Nunes in 1976, and can be seen in Figure 61. It measures $8\text{ m} \times 5\text{ m} \times 5\text{ m}$, and its walls are covered with a mixture of cork and rubber (with a density of approximately 0.8 g/cm^3), that form small spikes. Floating boards usually cover the tank, to absorb sound in every direction, but we did not use them, both because we wanted to simulate the sea surface, and because they interfered with our equipment. There is a sliding bridge over the tank that was used to hold the hydrophone and other equipment when necessary. One of the corners of the tank has a small compartment, with a sliding door, where the outboard motors were fixed. Other two corners have fixed hoses that are used to fill the tank and recycle its water. On one of the sides of the tank, there is a glass cabin where the measuring equipment is kept and operated.

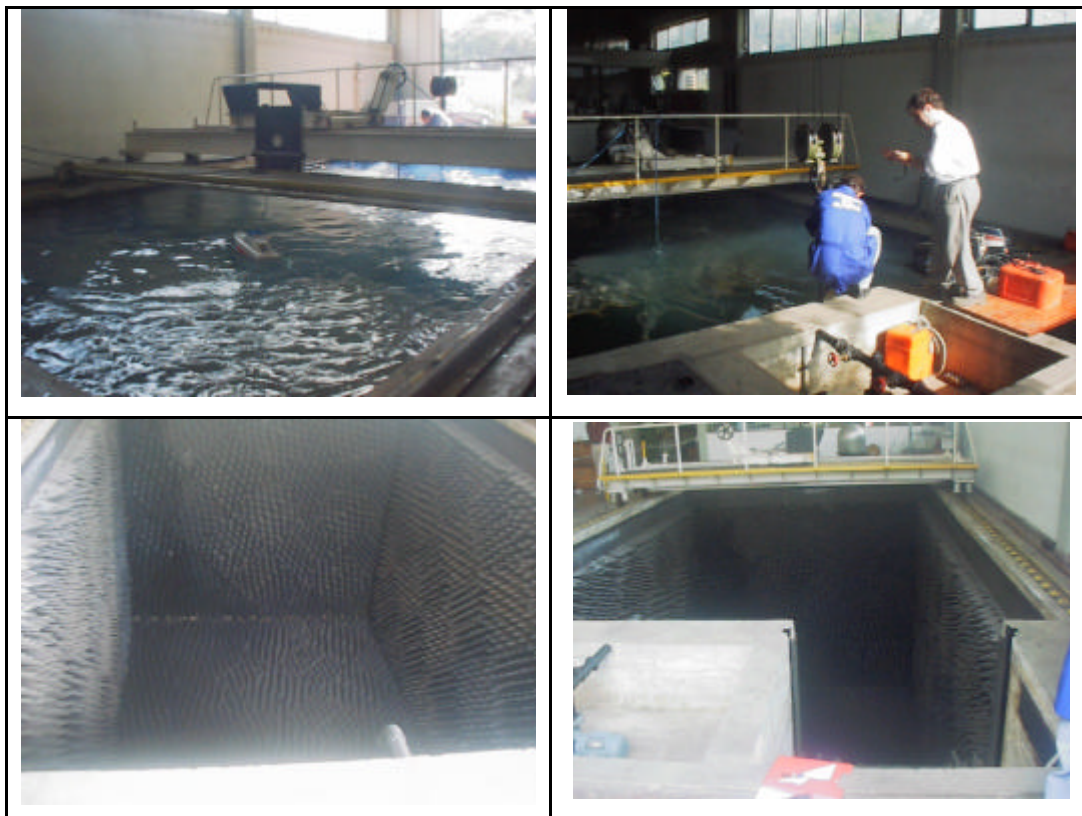


Figure 61 - Various aspects of the acoustic tank. Note the sliding bridge and the outboard motor fixation on the top photographs, and in the bottom ones, the acoustical isolation visible when the tank was emptied.

The sonar division is equipped with high quality hydrophones and amplifiers, build by Bruel. The hydrophone used in our tests was a reference “Bruel & Kjaer 8104” (see Figure 62), which is passive and omnidirectional, weighing 1,3 Kg, and is basically a 12 cm long cylinder with 2 cm diameter. It is sensitive to signals from 0.1Hz up to 200 kHz, and had recently been confirmed to have perfectly flat transfer function up to 20 kHz (the highest frequency we would try to measure).

The signal amplifier is a Bruel & Kjaer 2636, which is a 2 stage variable gain amplifier with various filters. We always used a high quality low-pass filter set to 20 kHz, to act as an anti-aliasing filter. The gain varied from recording to recording, being set manually after observing the signal for a short while, so as to maximize the dynamical range. Most of the time, the gain was around 30 dB.

After the amplifier, we had a high quality HP oscilloscope and spectral analyzer to monitor the signals that were being measured. This allowed us to manually set the optimal amplification to use all the available dynamic range without saturating, confirm that the signals were band limited, and correct any anomalies.

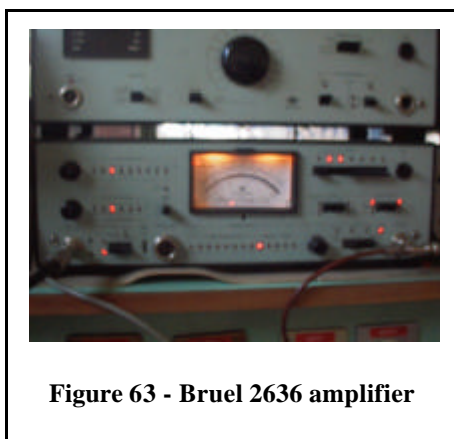


Figure 63 - Bruel 2636 amplifier

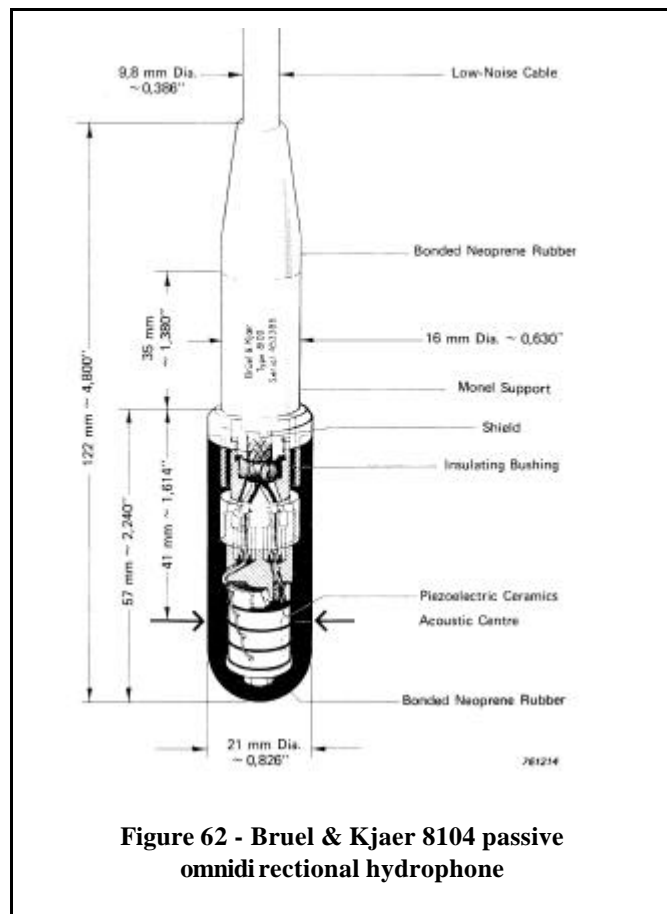


Figure 62 - Bruel & Kjaer 8104 passive omnidirectional hydrophone

Finally, the audio signal was fed to a 16 bit Sound Blaster compatible sound card installed in a Pentium 200MMX computer, with 64 Mb RAM and a 8 Gb hard disk. Lab tests showed that the transfer function of the sound card was reasonably flat in the

50 Hz to 20 kHz range, and could digitize very low frequency signals under 1 Hz. Since the same sound card was used in all recordings, all signals in the very low frequency range were affected in the same way. All recordings were done with a 44.1 kHz sampling rate, in mono channel mode.

4.1.2 - The hydrophonic effects generated

So that the data could be useful for our purposes, we had to generate noise similar to that of a real ship, and then introduce background noise similar to that found in the ocean, and finally, so that the data could also be used for other work, introduce transients similar to those that are interesting from a military or security point of view.

Since the main sources of noise in a ship are its main propulsion engines and auxiliary machines, we opted to use maritime outboard motors to simulate the “target” ships. The background noise was simulated with air bubbles and running water falling into the tank, simulating in some way the effect produced by waves and water movement. Finally, banging objects or shots of air pressure rifles provided the transients.

It must be clearly stated that these effects are not equivalent to the real effects they stand for, and their specific characteristics may differ considerably from them. They do however have the same general behavior, and the techniques developed for them can be re-applied to operational data, to obtain operational classifiers. Similarly, the accuracy results obtained are an indication for the performance of the operational system.

4.1.2.3 - The motors

We used 5 different types of motors: 4 outboard motors, and one small electrical motor of a radio-controlled model boat. Since the electric motor is significantly different from the others, and the amount of data recorded with it are considerably less, only the outboard motors are used in most of the tests with this dataset.

Motor 1

Motor 1 is a 4.5 horsepower Mercury, with one cylinder, and a right hand 3 blade propeller, belonging to the Portuguese Naval Academy's *NRP Vega* Yacht.



Figure 64 - Motor 1, a 4.5 hp Mercury



Figure 66 - Motor 3, a 3.6 hp Yamaha



Figure 65 - Motor 4, a 3.6 hp Mercury

Motor 2

Motor 2 is a 18 horsepower Mercury, with two cylinders, with a right hand 3 blade propeller, belonging to the Officers Club of the Portuguese Navy - CNOCA.

Motor 3

Motor 3 is a 8 horsepower Yamaha, with one cylinder, and a right hand 3 blade propeller, belonging to the Naval Academy's *NRP Polar Yacht*¹⁴.



Figure 67 _ Various aspects of the electric model boat (motor 5)

¹⁴ Curiously, this Yacht is a somewhat imperfect copy of the *America*, that won the first America Cup.

Motor 4

Motor 4 is a 3.6 horsepower Mercury, with one cylinder, and a right hand 3 blade reinforced rubber propeller, belonging to the Officers Club of the Portuguese Navy - CNOCA.

Motor 5

Motor 5 belongs to a small radio-controlled boat that is used in the Naval Architecture classes at the Naval Academy. It is shown in Figure 67, and has a left hand 3 bale plastic propeller. The servos that power the rudder make noise that is louder than the main motor, and so must also be taken into consideration.

4.1.2.4 - The interferences

To simulate the ocean background noise, two devices were used, each with two variants.

The first was to pour water into the tank using the fixed hoses. The pumps are reasonably far away and well isolated acoustically, so that when they were used we were able to hear only the water falling in the tank. We used two different intensities, to simulate different sea states.

The second, was to make air bubbles. These were produced with the help of the compressed air system available throughout the shipyard. A rubber hose was connected to the air outlet, and the other end of the hose was lowered from the sliding bridge of the tank, with a metal weight to take it close to the bottom of the tank. The weight was sufficiently heavy to stop the hose from moving around in the tank, even when air was pumped at full pressure. Again, two different air pressures were used to simulate different sea states.

4.1.2.5 - The transients



Figure 69 - Transient *a*: bursts of compressed air.

Besides what we considered “steady state” interferences, we also wanted to provoke transients. Even the steady state interferences are many times a sum of many distant transients, but isolated transients are important both to identify ships and their operation, and to identify certain

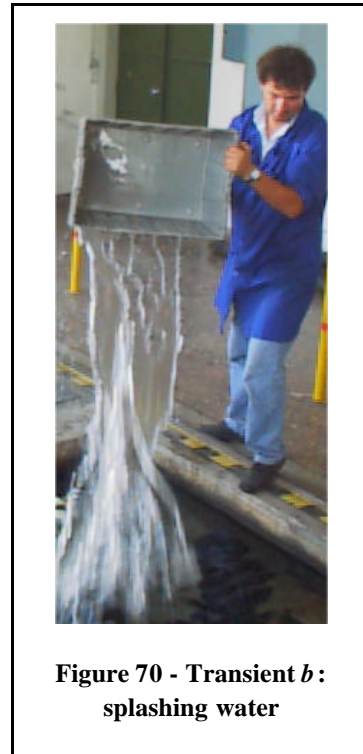


Figure 68 – Transients *c* and *d*: hitting metal tubes

natural sources, as for example marine mammals. In this thesis we are not interested in pursuing the latter objectives, and treat the transients as just another interference in the desired signals, namely the motor noise. This interference is, naturally, quite different from that produced by the “steady state” sources.

We produced 5 types of transients:

- a) Compressed air bursts. Produced with the compressed air setup described earlier, we simply turned the air on and back off very fast, generally producing a single large bubble that broke out into smaller ones. This transient simulated the effect that is produced when a ship produces air discharges under the surface, such as expelling gases or even firing torpedoes.
- b) Water splash. This was obtained by throwing a bucket full of water on the surface. The splash obtained simulated the effect of throwing liquids or even solid objects into the water.
- c) Hit of a metal tube with wood. This was obtained by hanging a metal tube with a rope, and hitting it with a wooden hammer. It simulates a number of effects that are produced by men and machinery inside a ship.
- d) Hit of a metal tube with iron. This was obtained by hanging a metal tube with a rope, and hitting it with a metal hammer. Since both objects are metallic, the sound is distinctly different from transient *c*. Like transient *c* it simulates a number of effects that are produced by men and machinery inside a ship.
- e) Air pressure shot. This was obtained using a air pressure shotgun. The barrel was inserted in the water, so that the air was all expelled through it. It is a much shorter than that produced by transient *a*, though a short bubbly noise can be heard. This noise may simulate a very violent air discharge, such as expelling small objects from a submarine, or dropping something from a ship.



4.1.3 - The recordings

Each of the above effects was recorded individually, and in conjunction with others. In all almost 5 hours of recordings were obtained (4 h 58 min 30 s to be precise). For the comparisons carried out in this thesis, we considered the “targets” to be the four internal combustion motors, named “motor1”, “motor2”, “motor3”, and “motor4”. All recordings where a single of these internal combustion motor was used, with and without various interferences and transients, where used. All the recordings of interferences, both with and without various transients, were used as

background noise, and for the sake of simplicity named “motor5”. We also used some recordings of transients by themselves (without steady state interferences) as background noise. The total recording time used was 4 h 30 min, totaling 1.457 GB, and the actual filenames of the recordings used are in the Appendix C.

When extracting data patterns from the recordings, we chose to use always segments of approximately 3 seconds. This choice was based on our experience and that of sonar operators, and it is the bare minimum required by a human operator, even under very favorable conditions. Using the same time interval, and thus approximately the same amount of data patterns in different experiments makes comparisons easier. Finally, this choice of time intervals provides what we think is a reasonable amount of data patterns. Information about the data used is presented in Table 12.

Effect	Nº of Patterns	Time	Size /MB
motor 1	1263	1 h 03 min	333.606
motor 2	949	47 min	249.587
motor 3	968	48 min	254.763
motor 4	1045	52 min	275.536
motor 5 (background)	1291	1 h 04 min	343.890
TOTAL	5516	5 h 01 min	1.457.382

Table 12 - General information about the Acoustic Tank data. The number of patterns correspond to 3s segments of the original signal. These will later be subject to different feature extraction techniques, to produce the final patterns.

4.2 - Datasets and experiments

Each raw pattern is a segment of a time-series stored in a Microsoft WAV file. Since the sample rate is 44 kHz and each sample requires 16 bits, each pattern consists of approximately 258 K. A number of feature extraction and selection techniques must be applied to both reduce the size of the patterns and strip them from irrelevant information.

4.2.1 - General overview of the signals

Before going into a detailed analysis of the data, we started by plotting the power spectra of the different motors, which are presented in Appendix D. These plots were obtained by computing the power spectra of each raw pattern with 5.3 Hz resolution¹⁵, corresponding to 4096 frequency bins, from 0 to 22 kHz. A Hamming window and 50% overlap Welch periodograms were used, so each spectrum presented is the average of 32 individual spectra¹⁶. Although this resolution is coarser than that used in the experiments described in Chapter 3, it does provide a good overview of the signals and, as we shall see later, is more than enough for an accurate classification.

From these figures, it is clear that the noise produced by the motors is felt in the low and very low frequency ranges. There is almost no visible signal above 12 kHz, and it seems possible to distinguish the various motors using only the frequency range from 0 to 270 Hz.

We chose to use only two different frequency resolutions in our tests. The finer resolution dataset, using 5.3 Hz per bin, was called dataset 1, while the coarser resolution, with 690 Hz per bin was called dataset 2.

4.2.1.1 - Dataset 1

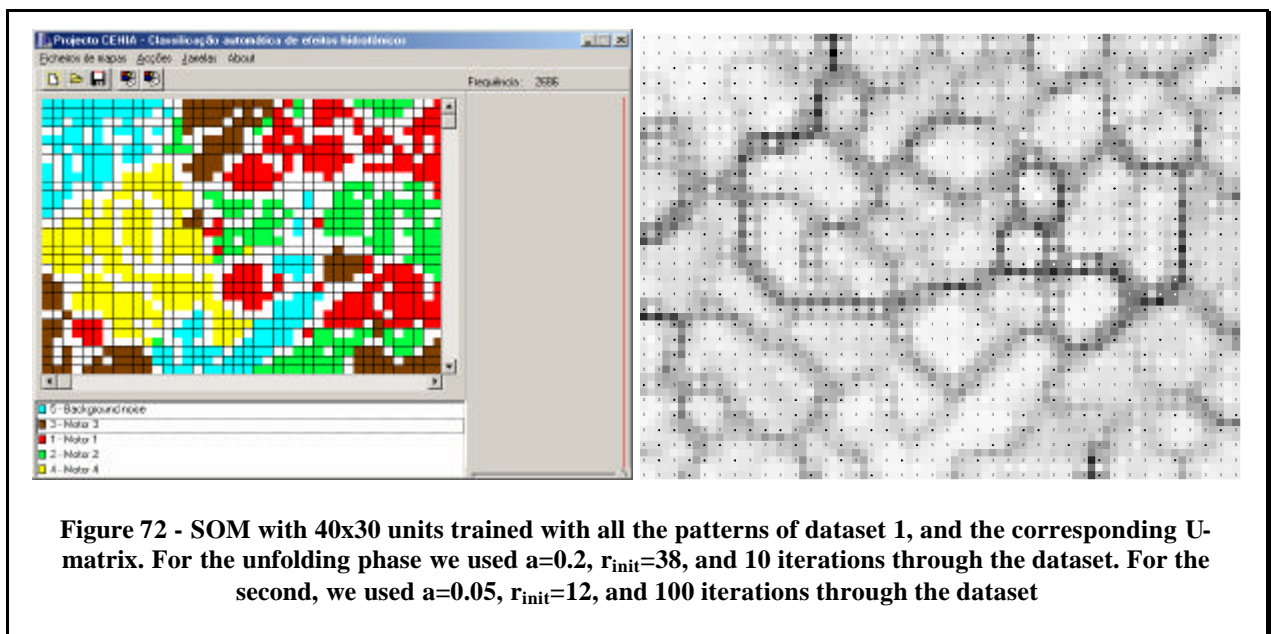
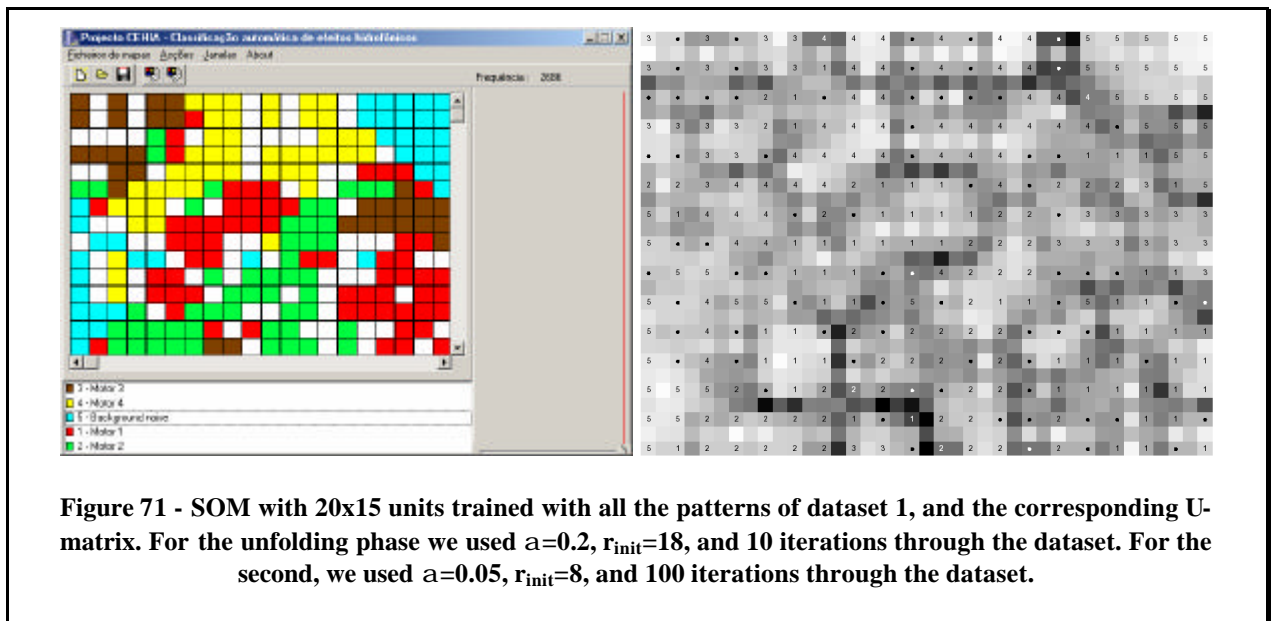
Dataset 1 consists of the power spectra used for the general overview of the signal. Each pattern is thus a vector with 4096 real-valued components, corresponding to the power spectra with 5.3 Hz per bin.

To check if the data clustered according to the desired classes, we trained a 15x20 unit SOM. The results are shown in Figure 71. The resubstitution error obtained when using this SOM as a

¹⁵ For the sake of simplicity we use the term resolution when we really are referring to the width of each bin. Due to the fact that we used a Hamming Window, the true resolution is actually quite lower, and corresponds roughly to 2.6 times more than the frequency width of each bin.

¹⁶ We broke up the original recordings into separate 3 s chunks only when we extracted the actual data patterns. This allows us optimize the usage of recording time by allowing very slight overlaps between data patterns. If we isolated the 3 s chunks before extracting the spectra, we would only be able to average 31 spectra per pattern. Allowing a slight overlap, we can use 32 spectra, which is more convenient.

classifier is only 0.96%, which is remarkably low considering SOM is a clustering technique. This reveals that this representation of the data should make the classes separable. However, a visual inspection of the SOM presented shows the classes are not grouped together, but are instead dispersed over the map. Tracing the origins of the various patterns, we can observe that the interferences (air bubbles, water flow, etc) are having a very strong influence in the distribution of data. This influence would be expected and can only be eliminated using signal processing techniques aimed at canceling it.



To further distinguish each cluster, we trained another SOM that was four times bigger, having 40x30 units. The results are presented in Figure 72, and clearly show well defined clusters, both in the colored SOM map, where there are unlabeled (white) units between each of the labeled groups of units, and in the U-matrix, where there are several well defined clusters. The resubstitution error for this larger map was 0.1%.

To perform cross-validation we randomly selected 10 pairs of training and test sets. All those 10 pairs have exactly the same name number of patterns, and all classes are represented in the same proportion as in the original known set. Each training set consisted of 4599 patterns, and each test set consisted of 511 patterns. The results are shown in Table 13.

Method	N° Prototypes	Error rate	Training time / s
NN	4941.0 \pm 0.0	0.0 \pm 0.0	0
CNN	72.8 \pm 1.5	0.1 \pm 0.2	926.65 \pm 310.36
RNN	63.0 \pm 2.9	0.2 \pm 0.2	63833.85 \pm 2409.19
QSet-P	64.5 \pm 1.9	0.3 \pm 0.3	26756.85 \pm 11325.93

Table 13 - Results of cross-validation on dataset 1.

4.2.1.1.1 - Dataset 1 with small training set

In most practical situations, the amount of acoustic data available for training is quite limited. To simulate this situation, we inverted the role of training and test sets, i.e., we trained 10 different classifiers with 511 patterns, and tested them with the remaining 4599. The results are shown in Table 14.

Method	N° Prototypes	Error rate	Training time / s
NN	549.0 \pm 0.0	0.5 \pm 0.2	0
CNN	48.4 \pm 3.0	1.4 \pm 0.2	51.65 \pm 12.82
RNN	43.1 \pm 3.4	1.8 \pm 0.4	2759.63 \pm 1057.01
QSet-P	43.1 \pm 2.3	1.6 \pm 0.6	195.26 \pm 1.47

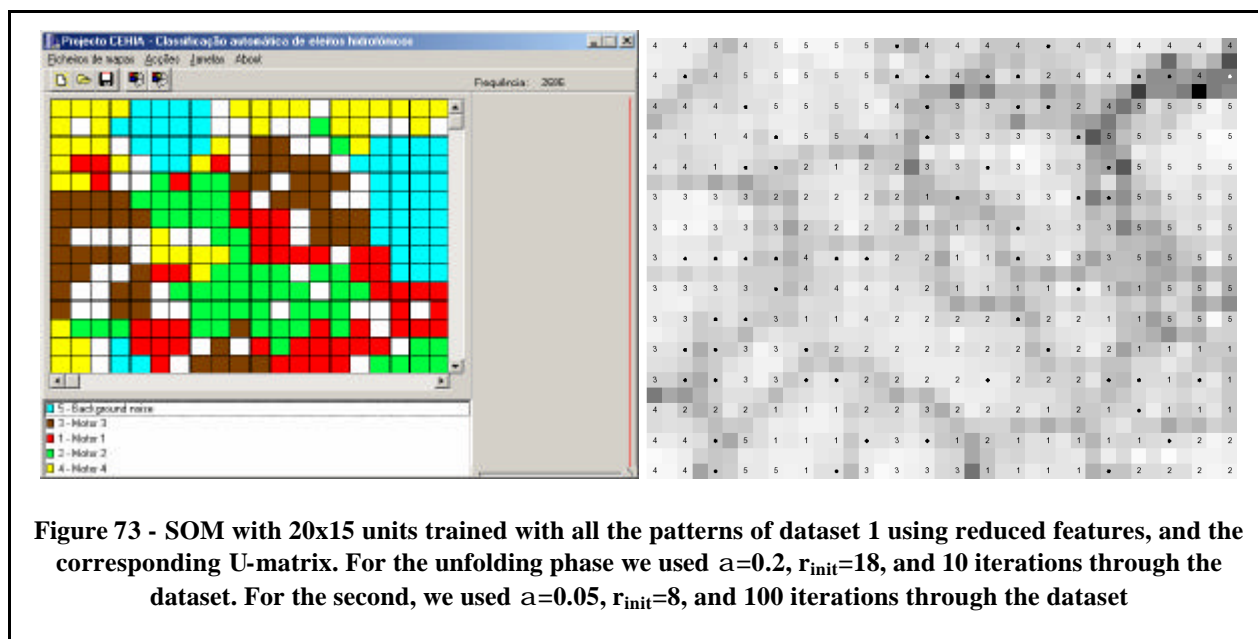
Table 14 - Results of cross-validation on dataset 1, using small training sets.

4.2.1.1.2 - Dataset 1 with reduced features

Due to the very large number of features, rough set techniques cannot be used to find a reduced set of features for classification. Following the classical approach, we tried to use scatter matrix techniques. Using full scatter matrices is computationally very demanding, so we have to work with each feature separately. For each feature, we computed its within class variance (S_w), and its variance between the class means (S_b). We then chose sequentially the 32 features that had greatest value of S_b/S_w , excluding those that had a correlation coefficient greater than 0.8 with any of the already selected features. The features chosen, together with their correlation coefficients, are presented in Table 15.

Order of choice	Bin	Frequency /Hz	Order of choice	Bin	Frequency /Hz	Order of choice	Bin	Frequency /Hz
1	227	1,219	12	204	1,095	23	36	189
2	258	1,387	13	28	146	24	23	119
3	10	49	14	370	1,991	25	45	237
4	267	1,435	15	175	939	26	192	1,031
5	3866	20,854	16	215	1,155	27	43	227
6	324	1,743	17	33	173	28	477	2,568
7	306	1,646	18	273	1,468	29	165	885
8	446	2,401	19	291	1,565	30	284	1,527
9	244	1,311	20	9	43	31	30	156
10	404	2,174	21	522	2,811	32	610	3,286
11	234	1,257	22	12	59			

Table 15 - Features selected from dataset 1, using scatter matrices.



Training a 20x15 unit SOM with this data and using it as a classifier yields a resubstitution error of 2.19 %.

Method	N° Prototypes	Error rate	Training time / s
NN	4941.0 \pm 0.0	0.1 \pm 0.1	0
CNN	100.3 \pm 3.5	0.5 \pm 0.3	9.53 \pm 1.31
RNN	85.6 \pm 3.0	0.5 \pm 0.3	228.04 \pm 13.60
QSet-P	94.9 \pm 2.0	0.4 \pm 0.4	2728.17 \pm 700.96

Table 16 - Results of cross-validation on dataset 1 with reduced features.

Method	N° Prototypes	Error rate	Training time / s
NN	549.0 \pm 0.0	1.2 \pm 0.3	0
CNN	53.5 \pm 3.9	2.4 \pm 0.5	0.76 \pm 0.17
RNN	46.7 \pm 3.6	2.7 \pm 0.5	7.42 \pm 1.16
QSet-P	48.1 \pm 2.3	2.6 \pm 0.5	1.84 \pm 0.04

Table 17 - Results of cross-validation on dataset 1 with reduced features, using small training sets.

4.2.1.2 - Dataset 2

Dataset 2 was obtained by calculating 64 point FFTs of the original signal, using a Hamming window, and 50% overlap between spectra. We then averaged 4096 consecutive spectra to obtain each data pattern. Each data pattern is thus the spectrum of approximately 3 s of the original signal, with 690 Hz per bin. Although such a coarse resolution goes against common wisdom amongst the submariner's community, we will show that fairly good classification accuracy can be obtained with it. The consequence is that far less computing power is required than when fine resolution is used. In a practical application, a low-resolution system may be used as a permanent vigilance system, while a finer resolution system will be used only to identify the targets where the first system has low confidence.

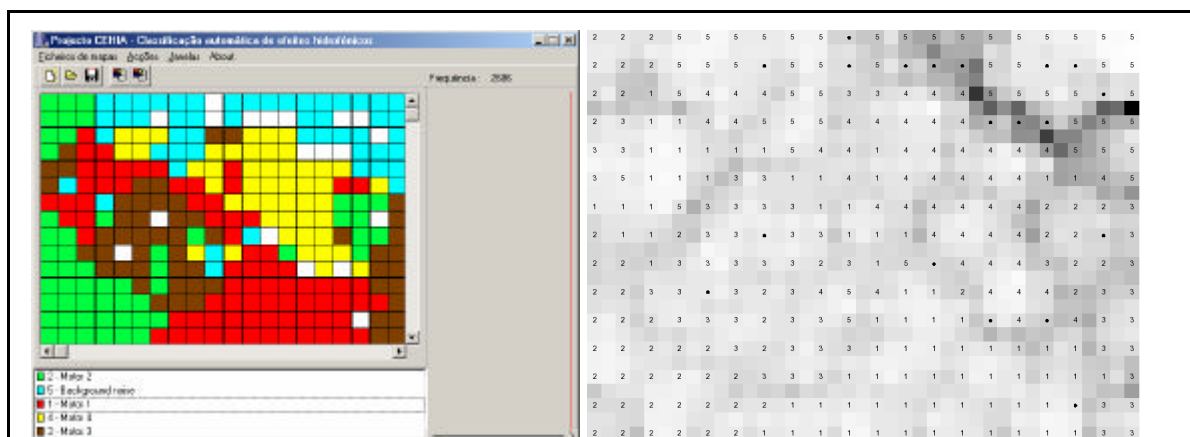


Figure 74 - SOM with 20x15 units trained with all the patterns of dataset 2, and corresponding U-matrix. For the unfolding phase we used $a=0.2$, $r_{init}=18$, and 10 iterations through the dataset. For the second, we used $a=0.05$, $r_{init}=8$, and 100 iterations through the dataset

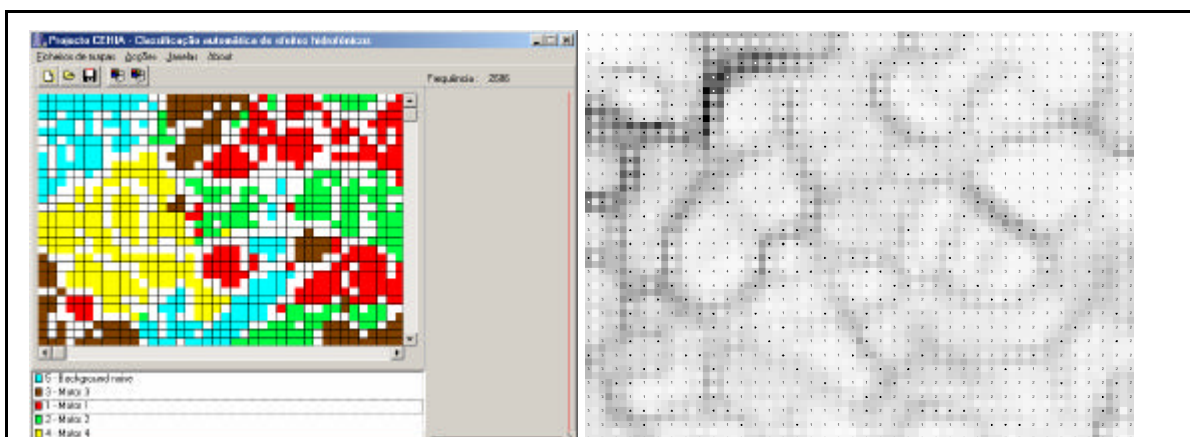


Figure 75 - SOM with 40x30 units trained with all the patterns of dataset 2, and corresponding U-matrix. For the unfolding phase we used $a=0.2$, $r_{init}=38$, and 10 iterations through the dataset. For the second, we used $a=0.05$, $r_{init}=12$, and 100 iterations through the dataset

When we train a SOM with all the patterns of this dataset, we obtain the map shown in Figure 74. The various classes do not seem to form well defined clusters, which would suggest that some pre-processing should be added. Even so, if we use this map as a classifier, we obtain a resubstitution error of 6.6% (339/5110). Considering that this is a clustering technique, that error can be considered quite good.

Applying 10-fold cross-validation, just as it was applied to dataset 1 discussed earlier, we obtained the results shown in Table 18.

Method	N° Prototypes	Error rate	Training time / s
NN	4599.0 ± 0.0	1.2 ± 0.4	0
CNN	266.6 ± 8.4	2.0 ± 0.4	22.99 ± 1.19
RNN	227.2 ± 6.3	2.4 ± 0.6	1517.16 ± 104.17
QSet-P	252.1 ± 7.3	2.1 ± 0.6	1496.04 ± 507.88
SOM (10x)	300.0 ± 0.0	8.4 ± 3.6	80.96 ± 1.47
SOM (100x)	300.0 ± 0.0	6.6 ± 3.6	844.56 ± 21.00

Table 18 - Results of cross-validation on dataset 2.

4.2.1.2.1 - Dataset 2 with small training sets

Using the same 10 fold partition of the dataset, but using only 1/10 of it for training and the rest for testing, we obtained the results shown in Table 19.

Method	N° Prototypes	Error rate	Training time / s
NN	511.0 ± 0.0	4.0 ± 0.5	0
CNN	84.1 ± 7.1	6.0 ± 0.6	0.92 ± 0.14
RNN	73.8 ± 5.4	6.4 ± 0.8	17.10 ± 2.56
QSet-P	77.0 ± 4.9	6.3 ± 1.0	1.66 ± 0.03
SOM (10x)	300.0 ± 0.0	22.6 ± 1.2	12.27 ± 3.03
SOM (100x)	300.0 ± 0.0	18.8 ± 1.4	86.66 ± 1.98

Table 19 - Results of cross-validation on dataset 2, using small training sets.

4.2.1.2.2 – Reduced features dataset 2

From the general overview of the signals, we suspect that it is possible to classify them using only the lower frequency ranges. Since dataset 2 has only 32 features, we can use roughsets to find which are dispensable for classification. Using 10 levels for discretization, the roughsetlab program found 16 reducts. Two of them have 22 features, while the rest have 23. The core consists of 20 of those features, corresponding to frequency bins 1 to 10, 12 to 14, 16 to 20, 23 and 31. The complete results are presented in table Table 20.

Frequency bins selected																															
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	22	23	26	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	22	23	27	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	22	23	28	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	23	24	26	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	23	24	27	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	23	24	28	31	32									
1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	17	18	19	20	21	23	25	31									
1	2	3	4	5	6	7	8	9	10	12	13	14	16	17	18	19	20	23	25	31	32										
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	22	23	26	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	22	23	27	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	22	23	28	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	23	24	26	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	23	24	27	31	32									
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	23	24	28	31	32									
1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	17	18	19	20	21	23	25	31									
1	2	3	4	5	6	7	8	9	10	12	13	15	16	17	18	19	20	23	25	31	32										

Table 20 - Reducts for dataset 2 produced by Roughsetlab, using 10 levels of discretization.

Re-applying the cross validation process applied earlier, we will have the results shown in Table 21 and Table 22.

Method	N° Prototypes	Error rate	Training time / s
NN	4599.0 ± 0.0	1.3 ± 0.4	0
CNN	288.3 ± 6.0	2.4 ± 0.6	18.98 ± 3.33
RNN	243.0 ± 3.4	2.7 ± 0.6	1198.91 ± 54.51
QSet-P	266.9 ± 7.2	2.6 ± 0.6	1027.31 ± 68.74

Table 21- Results of cross-validation on the reduced dataset 2.

Method	N° Prototypes	Error rate	Training time / s
NN	511.0 ± 0.0	4.3 ± 0.4	0
CNN	85.3 ± 6.6	6.3 ± 0.6	0.70 ± 0.11
RNN	71.9 ± 5.3	6.8 ± 0.7	11.15 ± 1.50
QSet-P	77.1 ± 6.4	6.9 ± 1.1	1.22 ± 0.04

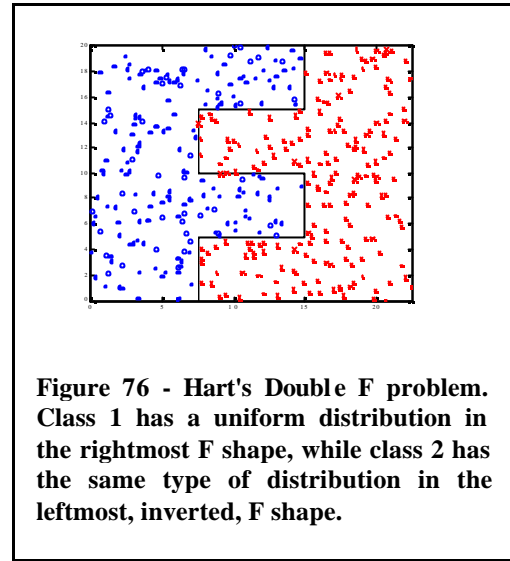
Table 22- Results of cross-validation on the reduced dataset 2, using small training sets

APPENDIX A

Experiments with Hart's double F problem

This appendix presents the detailed results of applying a series of prototype minimization techniques to Hart's double F problem, as described in Chapter 1 of part II.

This problem, initially proposed by (Hart 1968), consists of two classes of bi-dimensional patterns with a uniform distribution in two interlocked F shapes, as seen in Figure 76. The two classes lie in the 22.5×20 rectangle with the bottom left corner at the origin $(0,0)$, and have boundaries defined by the line that joins $(7.5,0)$, $(7.5,5)$, $(15,5)$, $(15,10)$, $(7.5,10)$, $(7.5,15)$, $(15,15)$, $(15,20)$.



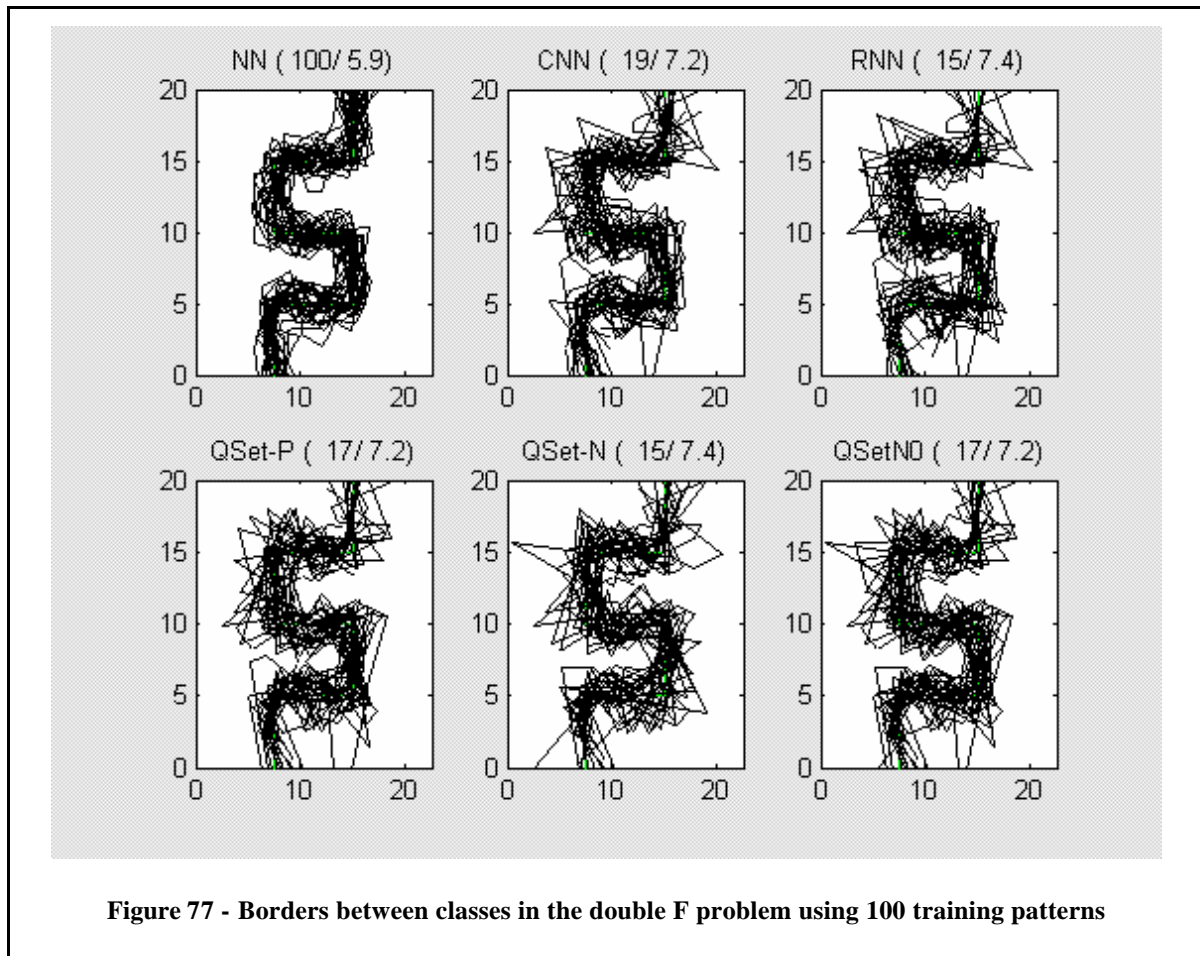
In the following experiments, 30 different datasets, randomly generated with the described probability density function, are used for each size of training set. For each of those datasets, classifiers were designed using standard nearest neighbors (NN), Condensed Nearest Neighbors (CNN), Reduced Nearest Neighbors (RNN), positive-only Q-Set heuristic (QSET-P or Q-Set (P)), general case Q-Set heuristic with 1 acceptable error (QSET-N or Q-Set (N_1)), and general case Q-Set heuristic with no acceptable errors (QSET-N0 or Q-Set (N_0)).

A large test set, consisting of 100.000 patterns, was then classified with each of the 6 classifiers, so as to estimate the generalization error.

The average number of prototypes, error rate, training time, and classification times (for the test set), are presented, together with the standard deviation of those values.

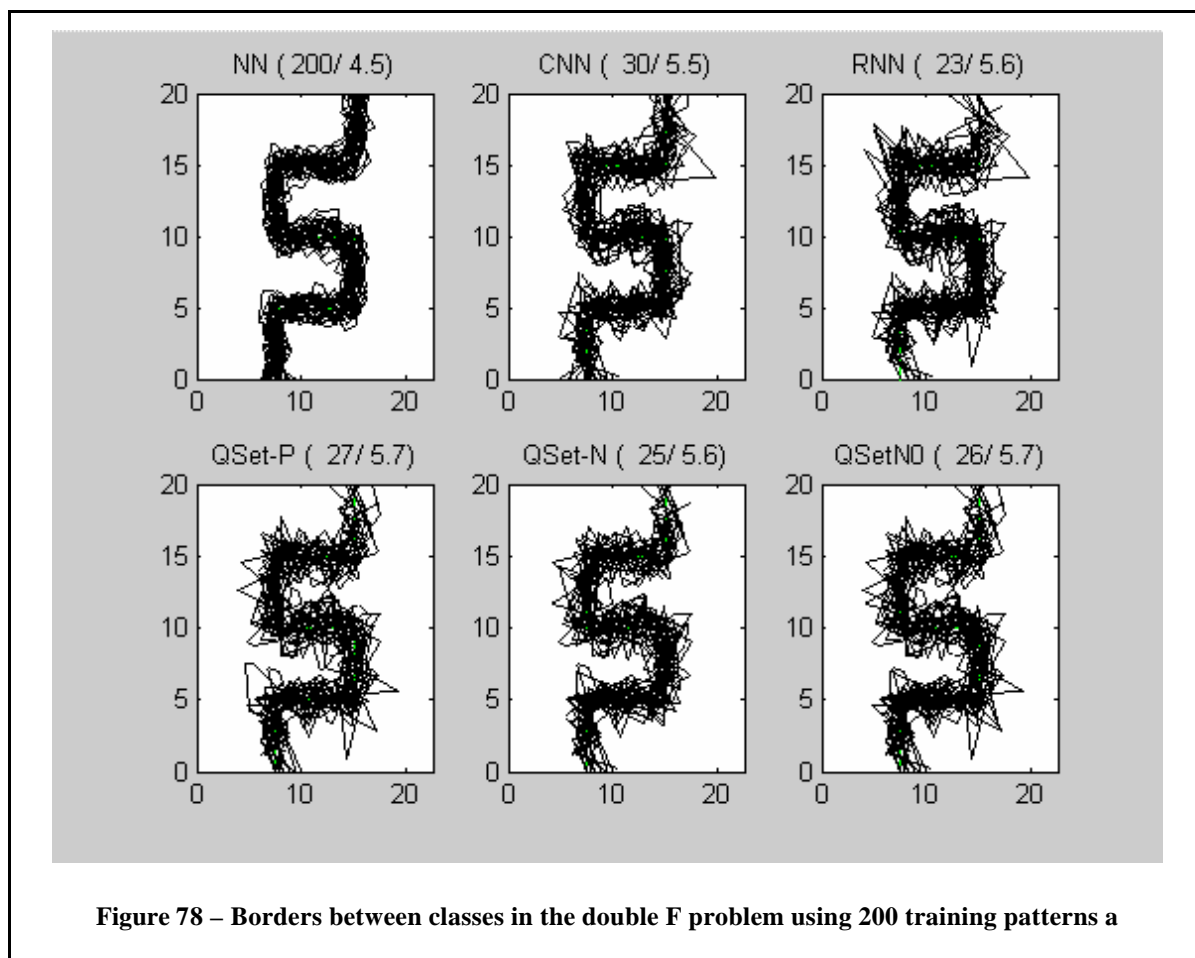
Since the patterns are 2-dimensional, it is easy to graph them, and these graphs can give us insight into the problem. The graphs presented do not show the individual patterns chosen in each case, but superimpose the borders between classes, obtained for the 30 trials performed. Over each of the graphs, the name of the method used is given, followed by the average number of prototypes and average error, in parenthesis.

100 patterns



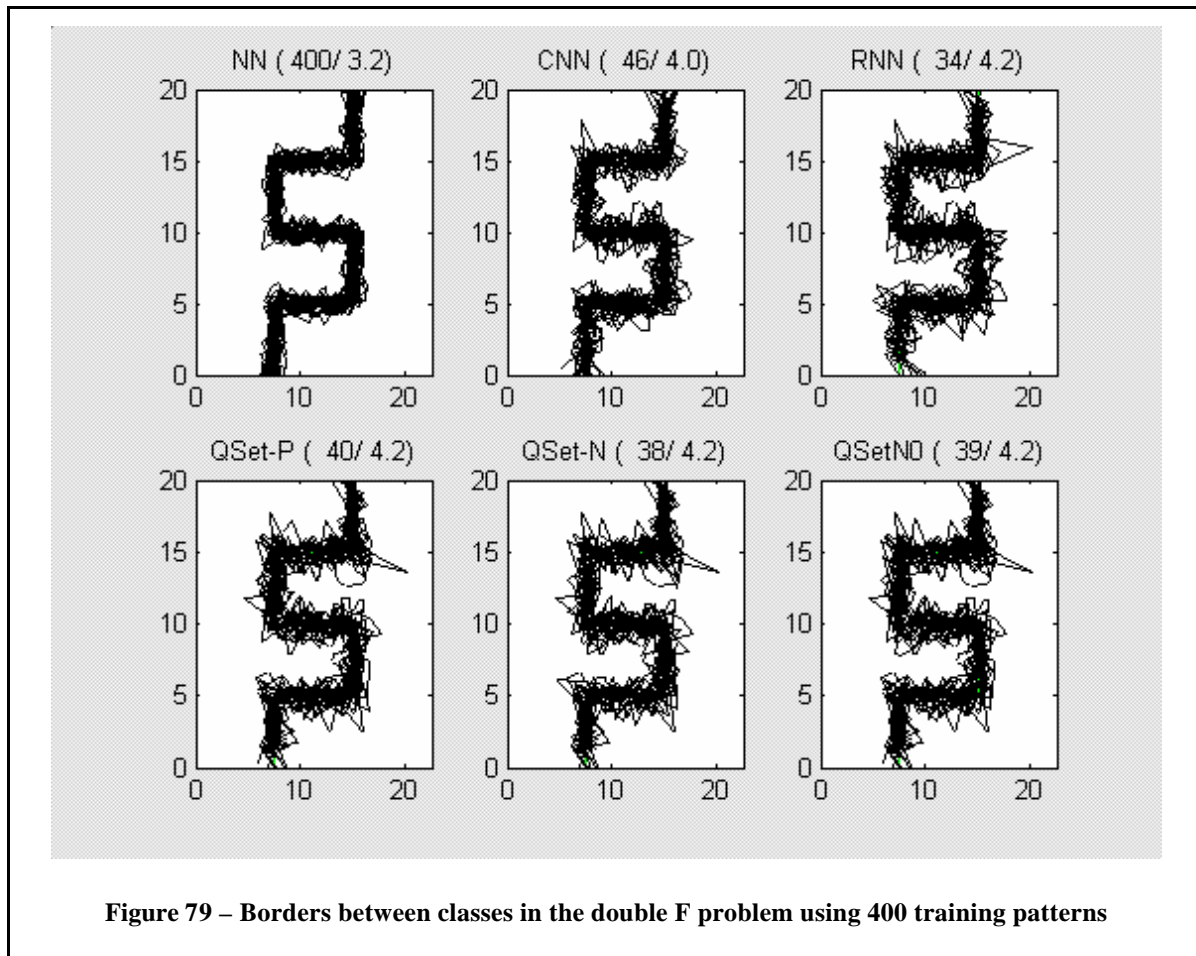
Method	N° Prototypes	Error rate	Training time	Classification time
NN	100.0 ± 0.0	5.9 ± 1.1	0.00 ± 0.00	6.15 ± 0.02
CNN	18.6 ± 3.6	7.2 ± 1.5	0.05 ± 0.02	1.17 ± 0.22
RNN	14.6 ± 3.1	7.4 ± 1.5	0.10 ± 0.04	0.92 ± 0.18
Q-Sets (P)	17.0 ± 2.4	7.2 ± 1.4	0.03 ± 0.03	1.07 ± 0.14
Q-Sets (N ₁)	15.3 ± 2.3	7.4 ± 1.2	0.67 ± 0.23	0.96 ± 0.14
Q-Sets (N ₀)	16.6 ± 2.5	7.2 ± 1.3	0.39 ± 0.21	1.05 ± 0.14

200 patterns



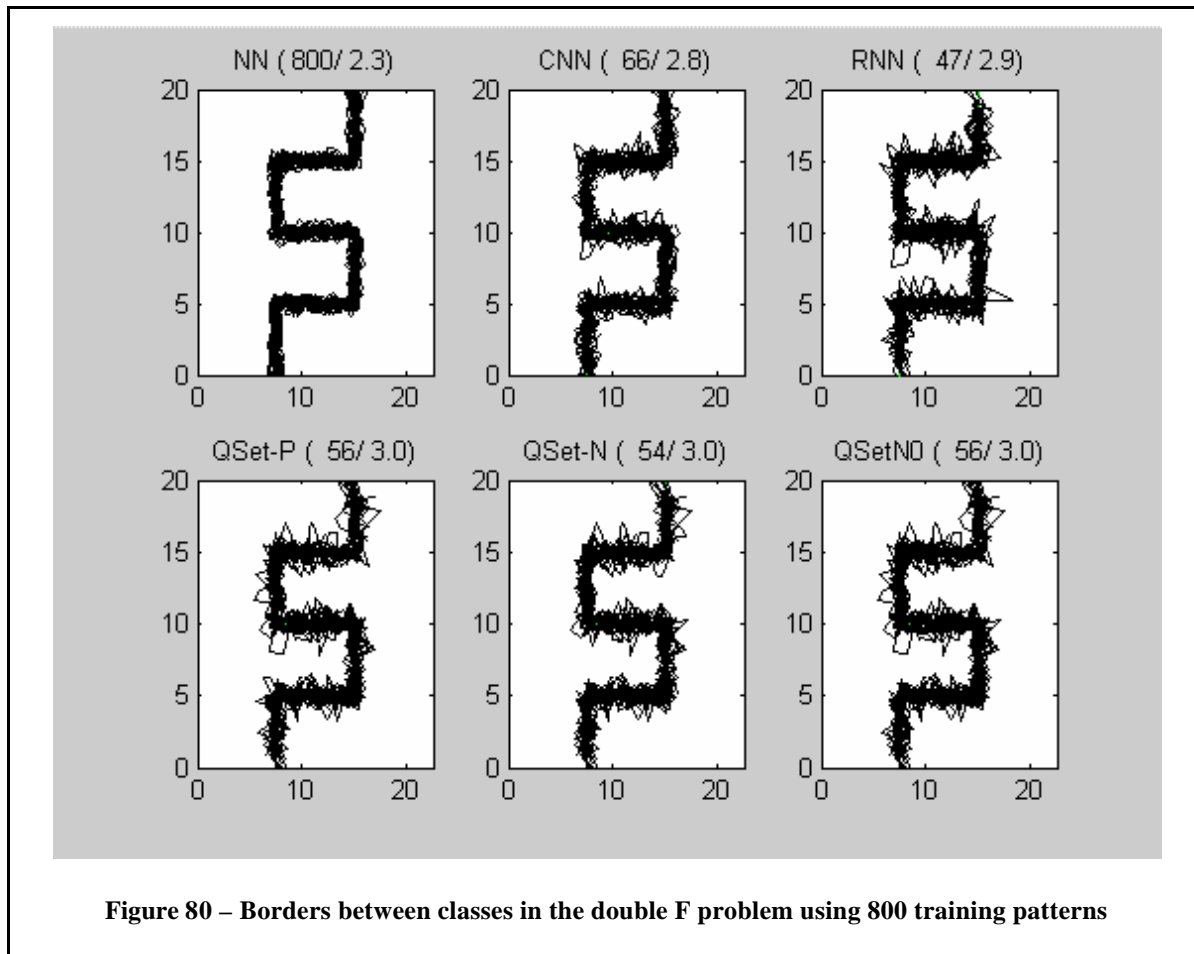
Method	N° Prototypes	Error rate	Training time	Classification time
NN	200.0 ± 0.0	4.5 ± 0.7	0.00 ± 0.00	12.27 ± 0.03
CNN	30.2 ± 3.4	5.5 ± 1.1	0.12 ± 0.02	1.85 ± 0.20
RNN	23.4 ± 2.8	5.6 ± 1.1	0.35 ± 0.07	1.45 ± 0.17
Q-Sets (P)	26.8 ± 3.2	5.7 ± 1.0	0.10 ± 0.02	1.65 ± 0.19
Q-Sets (N ₁)	25.0 ± 3.0	5.6 ± 0.9	3.97 ± 1.65	1.55 ± 0.19
Q-Sets (N ₀)	26.4 ± 3.2	5.7 ± 0.9	2.25 ± 1.19	1.62 ± 0.19

400 patterns



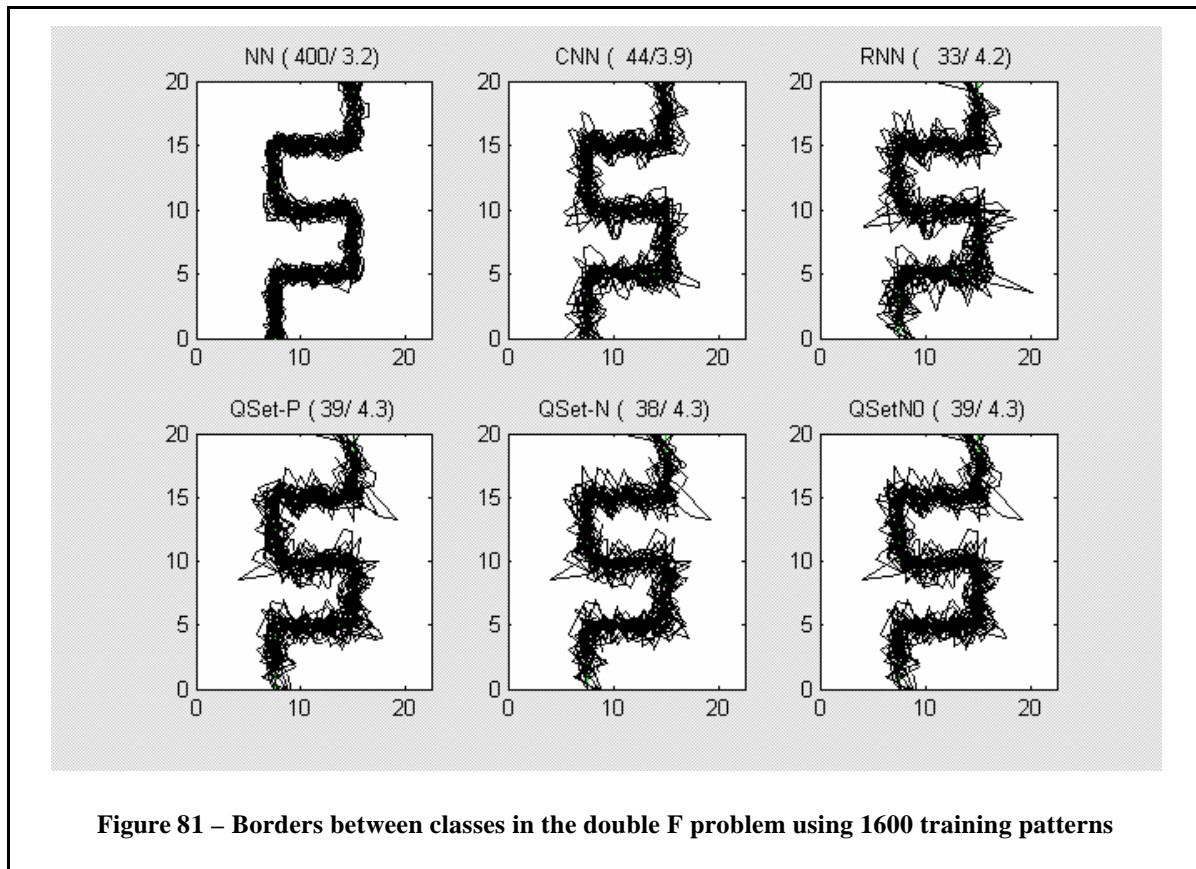
Method	N° Prototypes	Error rate	Training time	Classification time
NN	400.0 ± 0.0	3.2 ± 0.3	0.00 ± 0.00	24.33 ± 0.13
CNN	46.1 ± 6.1	4.0 ± 0.5	0.29 ± 0.05	2.77 ± 0.37
RNN	34.3 ± 4.2	4.2 ± 0.7	1.20 ± 0.26	2.08 ± 0.25
Q-Sets (P)	39.6 ± 4.0	4.2 ± 0.7	0.39 ± 0.03	2.38 ± 0.23
Q-Sets (N ₁)	37.6 ± 4.2	4.2 ± 0.7	24.17 ± 10.72	2.26 ± 0.24
Q-Sets (N ₀)	39.3 ± 4.0	4.2 ± 0.7	12.60 ± 7.85	2.37 ± 0.23

800 patterns



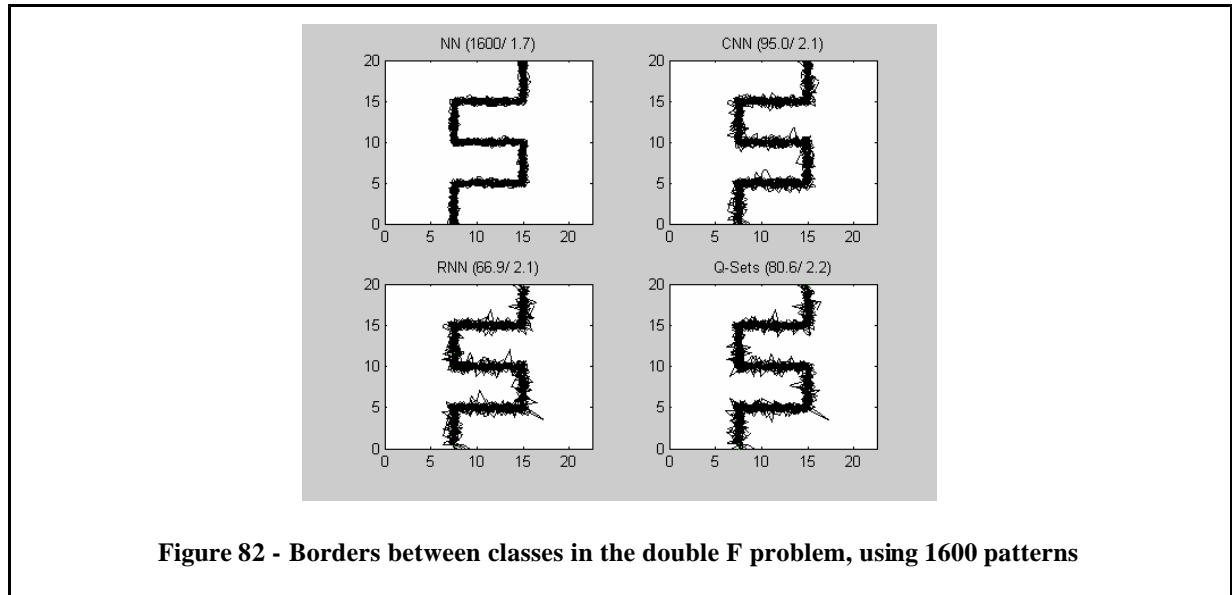
Method	N° Prototypes	Error rate	Training time	Classification time
NN	800.0 ± 0.0	2.3 ± 0.3	0.00 ± 0.00	48.92 ± 0.22
CNN	66.0 ± 7.2	2.8 ± 0.4	0.75 ± 0.11	3.98 ± 0.47
RNN	47.0 ± 5.3	2.9 ± 0.4	3.98 ± 0.78	2.80 ± 0.33
Q-Sets (P)	56.3 ± 4.6	3.0 ± 0.5	1.60 ± 0.06	3.35 ± 0.28
Q-Sets (N ₁)	54.2 ± 4.5	3.0 ± 0.4	146.04 ± 71.77	3.23 ± 0.28
Q-Sets (N ₀)	55.9 ± 4.5	3.0 ± 0.5	74.15 ± 40.27	3.34 ± 0.27

1600 patterns



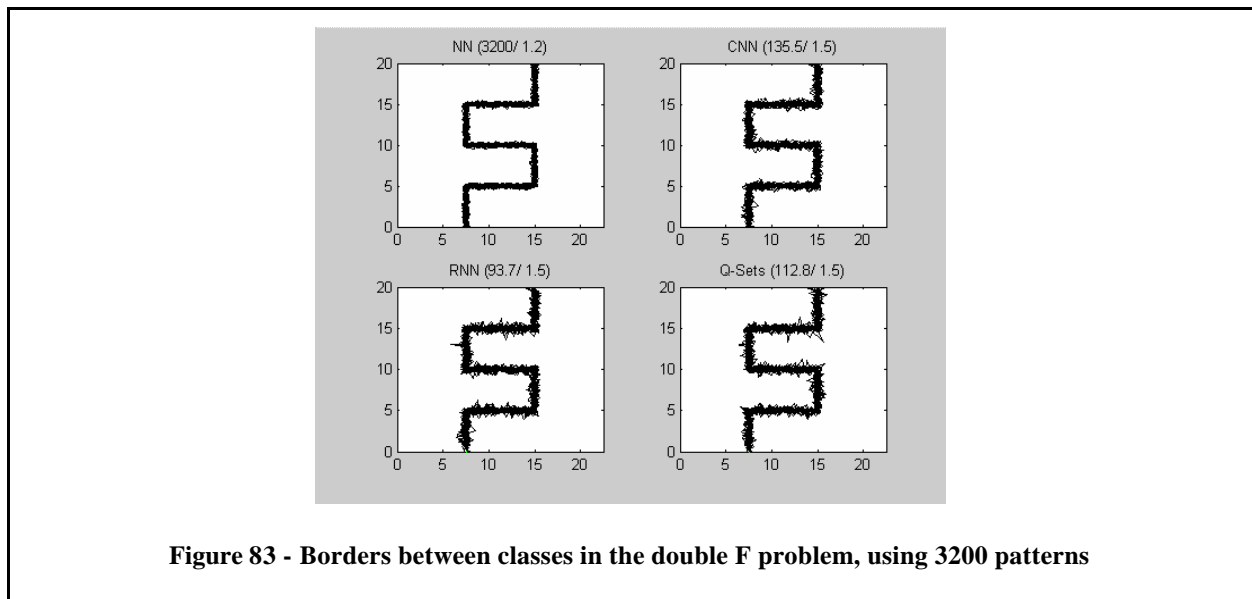
Method	N° Prototypes	Error rate	Training time	Classification time
NN	1600.0 ± 0.0	1.7 ± 0.2	0.00 ± 0.00	109.99 ± 0.34
CNN	93.8 ± 7.9	2.1 ± 0.2	1.83 ± 0.27	5.69 ± 0.48
RNN	66.3 ± 5.3	2.2 ± 0.2	14.20 ± 2.15	3.98 ± 0.34
Q-Sets (P)	80.8 ± 4.9	2.1 ± 0.2	6.93 ± 0.15	4.89 ± 0.30
Q-Sets (N ₁)	78.5 ± 4.8	2.1 ± 0.2	777.00 ± 354.19	4.74 ± 0.30
Q-Sets (N ₀)	80.6 ± 4.9	2.1 ± 0.2	378.99 ± 129.93	4.88 ± 0.30

1600 patterns:



Method	N° Prototypes	Error rate	Training time	Classification time
NN	1600.0 ± 0.0	1.7 ± 0.2	0	116.21 ± 4.10
CNN	95.0 ± 6.8	2.1 ± 0.3	2.43 ± 0.36	5.79 ± 0.43
RNN	66.9 ± 5.5	2.1 ± 0.3	15.52 ± 2.18	4.04 ± 0.37
Q-Sets	80.6 ± 4.3	2.2 ± 0.3	9.95 ± 0.66	4.91 ± 0.27

3200 patterns:

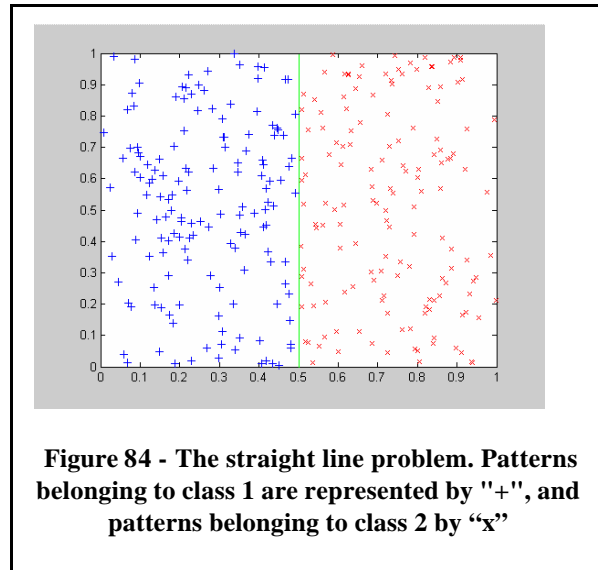


Method	N° Prototypes	Error rate	Training time	Classification time
NN	3200.0 ± 0.0	1.2 ± 0.1	0	385.88 ± 19.08
CNN	135.5 ± 12.2	1.5 ± 0.1	6.57 ± 0.87	8.31 ± 0.72
RNN	93.7 ± 8.3	1.5 ± 0.2	57.44 ± 9.64	5.68 ± 0.49
Q-Sets	112.8 ± 6.9	1.5 ± 0.1	41.40 ± 2.22	6.83 ± 0.42

APPENDIX B

Experiments with the straight line problem

This problem consists of two classes with uniform distribution in the unit square limited by (0,0) and (1,1). Those that lie in the left side of that square (i.e., with $x < 0.5$) are considered to belong to class 1, and the others to class 2, as seen in Figure 84. A summary of the results, and their discussion can be found in Chapter 1.6.3 of part II.



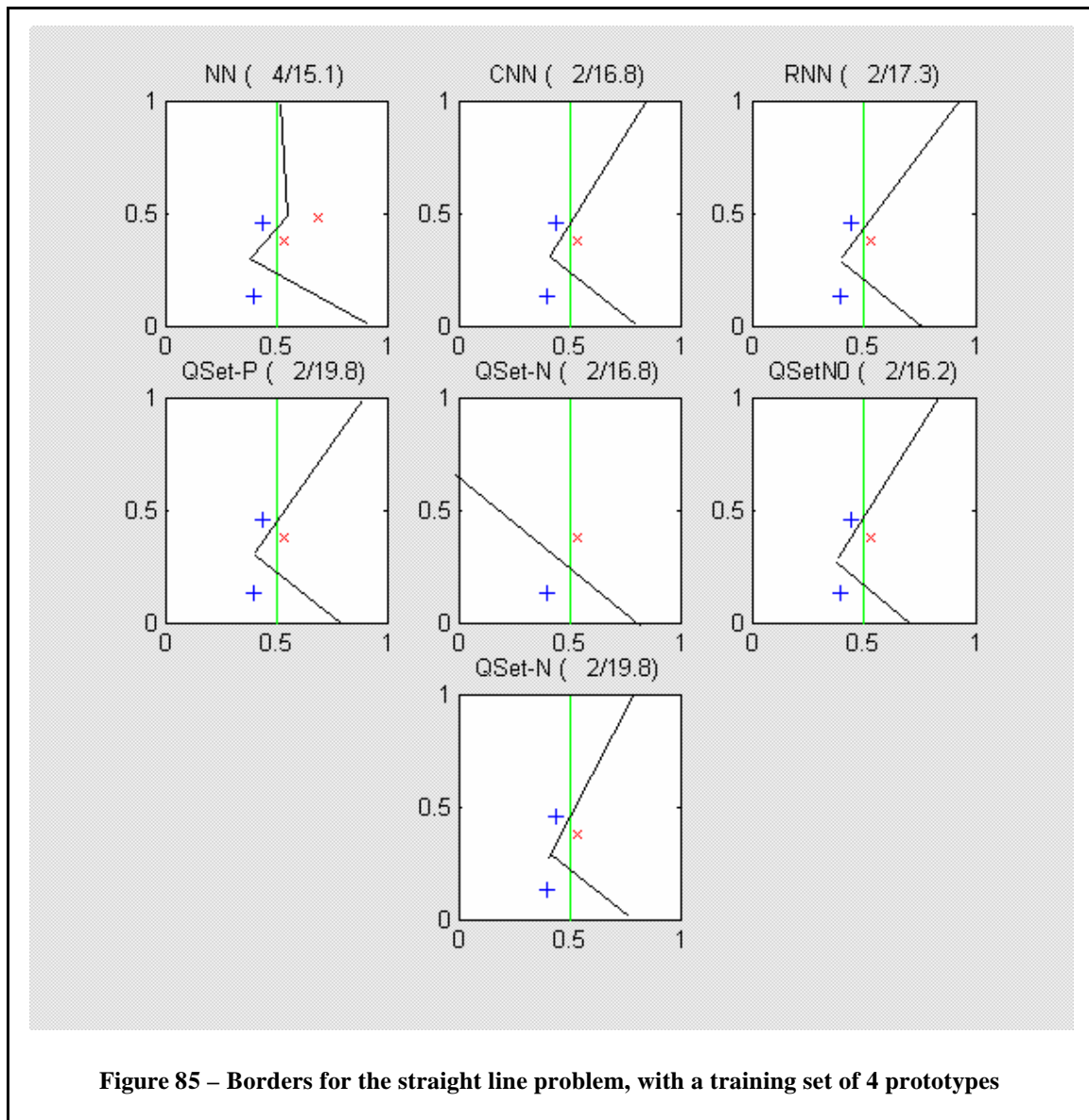
In the following experiments, 30 different datasets, randomly generated with the described probability density function, are used for each size of training set. For each of those datasets, classifiers were designed using standard nearest neighbors (NN), Condensed Nearest Neighbors (CNN), Reduced Nearest Neighbors (RNN), positive-only Q-Set heuristic (QSET-P or Q-Set (P)), general case Q-Set heuristic with 1 acceptable error (QSET-N or Q-Set (N_1)), general case Q-Set heuristic with no acceptable errors (QSET-N0 or Q-Set (N_0)), and positive-only Q-sets with a branch-and-bound search for the optimal solution (QSET-BB).

A large test set, consisting of 100.000 patterns, was then classified with each of the 6 classifiers, so as to estimate the generalization error.

The average number of prototypes, error rate, training time, and classification times (for the test set), are presented, together with the standard deviation of those values.

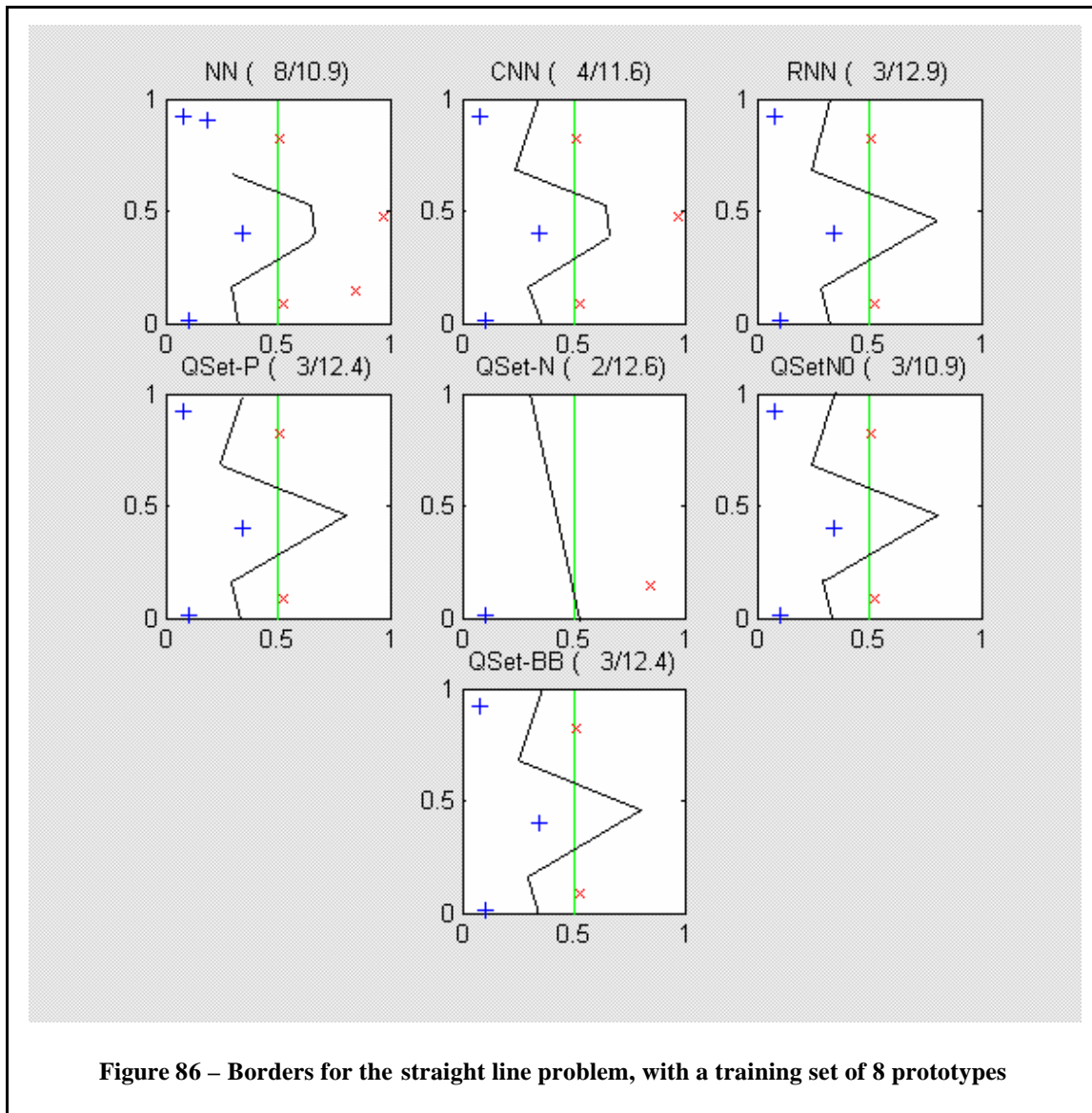
Unlike Appendix A, where the graphs presented superimposed the borders obtained with the 30 trials, we here present only one of those trials. We do however show individual prototypes chosen, and how they generate the border for that particular trial. The complete training set can naturally be seen in the graph of the nearest neighbor classifier. This is only possible here, since for all experiments the total number of patterns is relatively small.

4 patterns



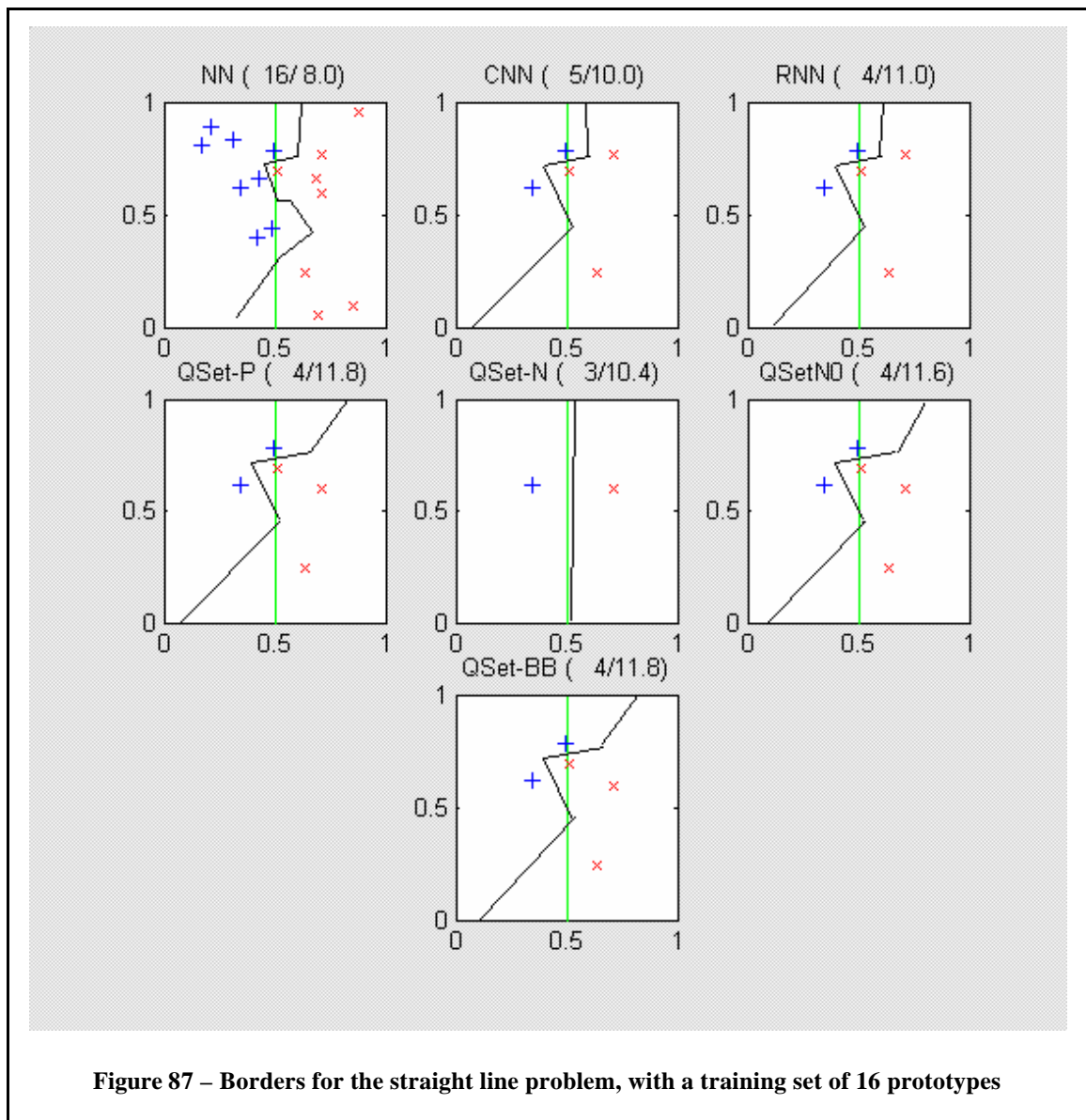
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	4.0 ± 0.0	17.4 ± 8.1	0.00 ± 0.00	0.54 ± 0.02
CNN	2.8 ± 0.8	19.3 ± 9.9	0.00 ± 0.01	0.40 ± 0.09
RNN	2.4 ± 0.7	18.9 ± 11.2	0.00 ± 0.01	0.36 ± 0.08
QSet-P	2.5 ± 0.7	18.9 ± 11.2	0.00 ± 0.00	0.37 ± 0.08
QSet-N	2.0 ± 0.0	21.5 ± 10.7	0.01 ± 0.02	0.32 ± 0.02
QSet-N ₀	2.4 ± 0.7	17.7 ± 10.3	0.01 ± 0.02	0.36 ± 0.09
QSet-BB	2.5 ± 0.7	18.9 ± 11.2	0.00 ± 0.01	0.37 ± 0.09

8 patterns



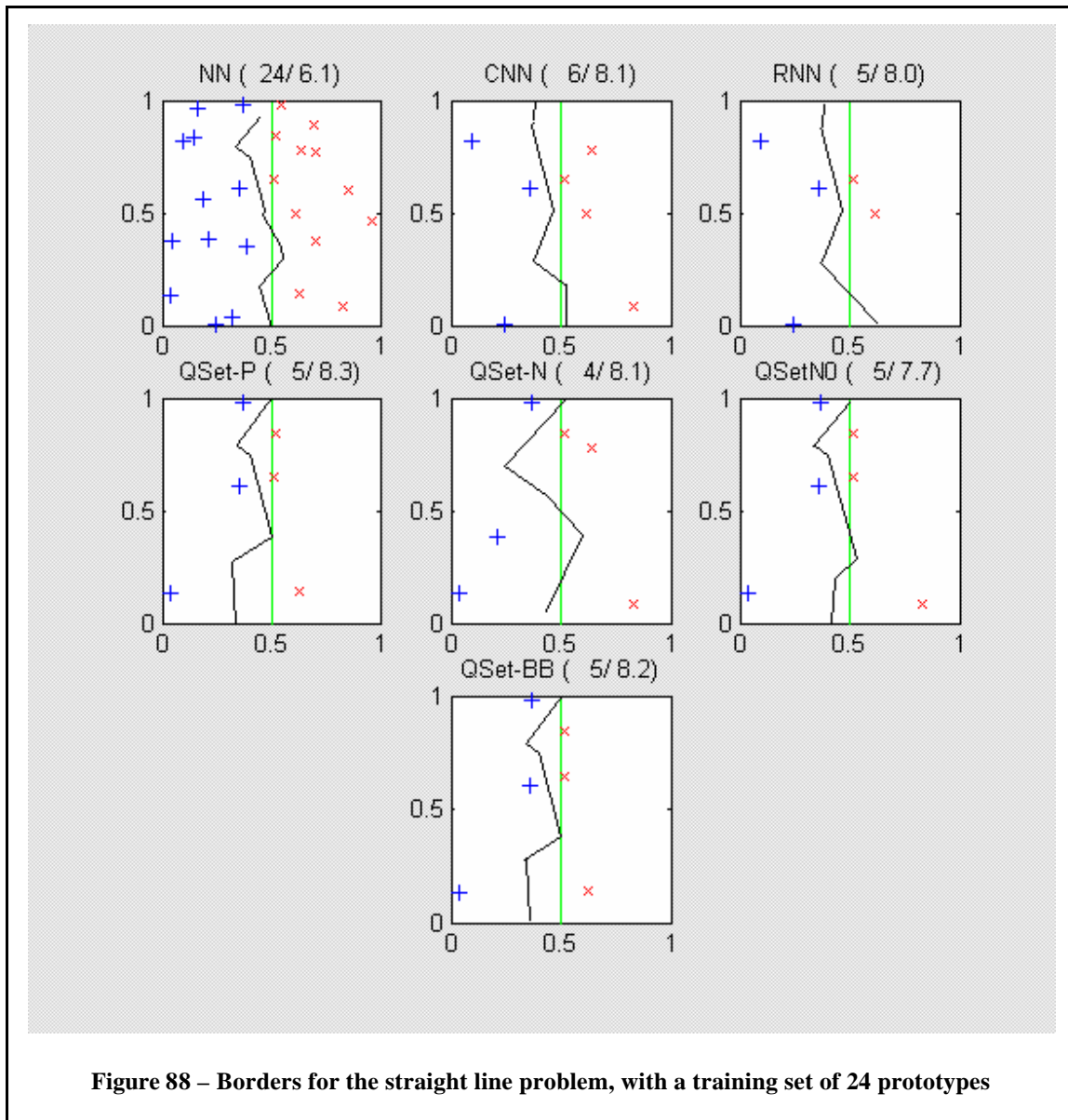
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	8.0 ± 0.0	10.9 ± 5.1	0.00 ± 0.00	0.98 ± 0.02
CNN	3.7 ± 1.4	11.6 ± 5.8	0.00 ± 0.01	0.51 ± 0.14
RNN	3.1 ± 1.0	12.9 ± 5.8	0.00 ± 0.01	0.45 ± 0.11
QSet-P	3.2 ± 1.0	12.4 ± 5.2	0.00 ± 0.01	0.45 ± 0.11
QSet-N	2.3 ± 0.7	12.6 ± 8.1	0.01 ± 0.02	0.35 ± 0.07
QSet-N ₀	2.9 ± 1.1	10.9 ± 6.2	0.02 ± 0.03	0.42 ± 0.12
QSet-BB	3.2 ± 1.0	12.4 ± 5.2	0.00 ± 0.01	0.46 ± 0.11

16 patterns



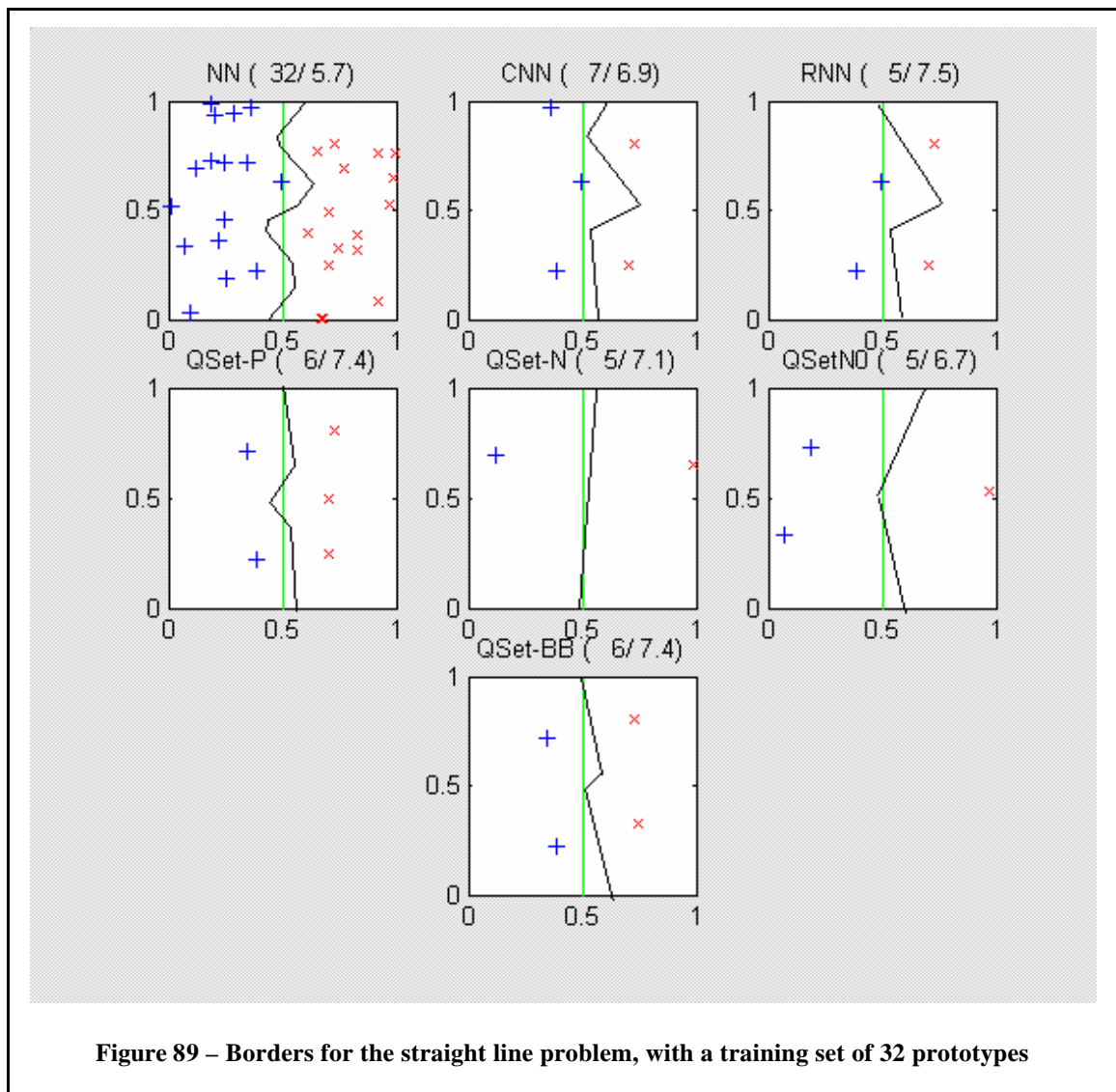
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	16.0 ± 0.0	8.0 ± 4.0	0.00 ± 0.00	1.84 ± 0.03
CNN	4.6 ± 1.3	10.0 ± 5.6	0.00 ± 0.01	0.60 ± 0.15
RNN	3.5 ± 1.1	11.0 ± 7.5	0.01 ± 0.02	0.48 ± 0.12
QSet-P	4.1 ± 0.9	11.8 ± 7.5	0.01 ± 0.02	0.54 ± 0.10
QSet-N	2.8 ± 0.7	10.4 ± 6.7	0.04 ± 0.03	0.39 ± 0.09
QSet-N ₀	3.8 ± 0.8	11.6 ± 7.9	0.02 ± 0.03	0.52 ± 0.10
QSet-BB	4.0 ± 0.9	11.8 ± 7.5	0.10 ± 0.14	0.54 ± 0.09

24 patterns



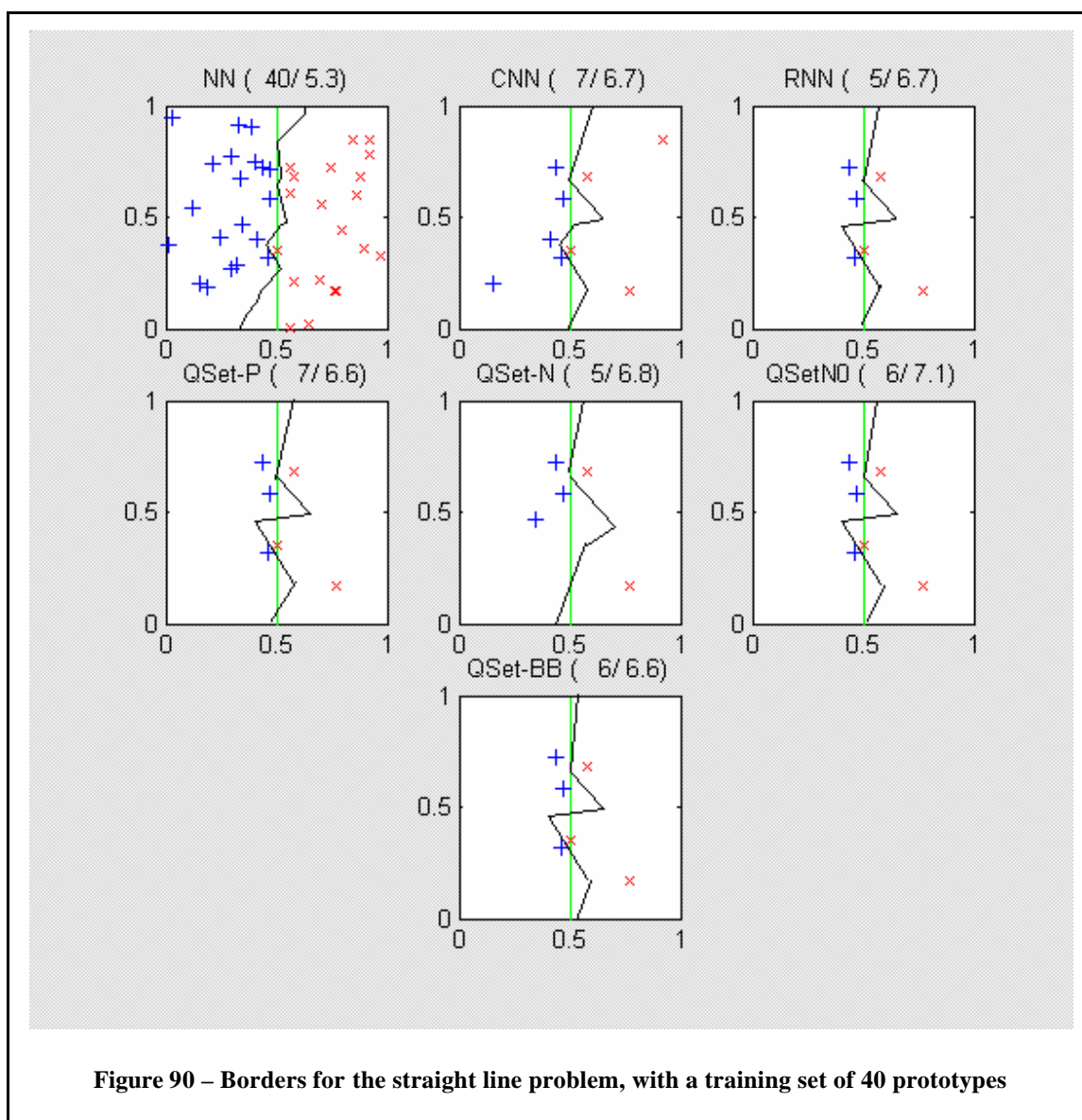
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	24.0 ± 0.0	6.1 ± 1.9	0.00 ± 0.00	2.72 ± 0.03
CNN	5.9 ± 1.9	8.1 ± 2.8	0.01 ± 0.02	0.75 ± 0.21
RNN	4.5 ± 1.3	8.0 ± 3.0	0.01 ± 0.02	0.59 ± 0.14
QSet-P	5.2 ± 1.0	8.3 ± 2.8	0.00 ± 0.01	0.66 ± 0.11
QSet-N	4.1 ± 1.4	8.1 ± 2.9	0.08 ± 0.06	0.55 ± 0.16
QSet-N ₀	4.8 ± 1.5	7.7 ± 2.7	0.05 ± 0.06	0.62 ± 0.16
QSet-BB	5.1 ± 1.0	8.2 ± 2.7	7.75 ± 34.79	0.65 ± 0.11

32 patterns



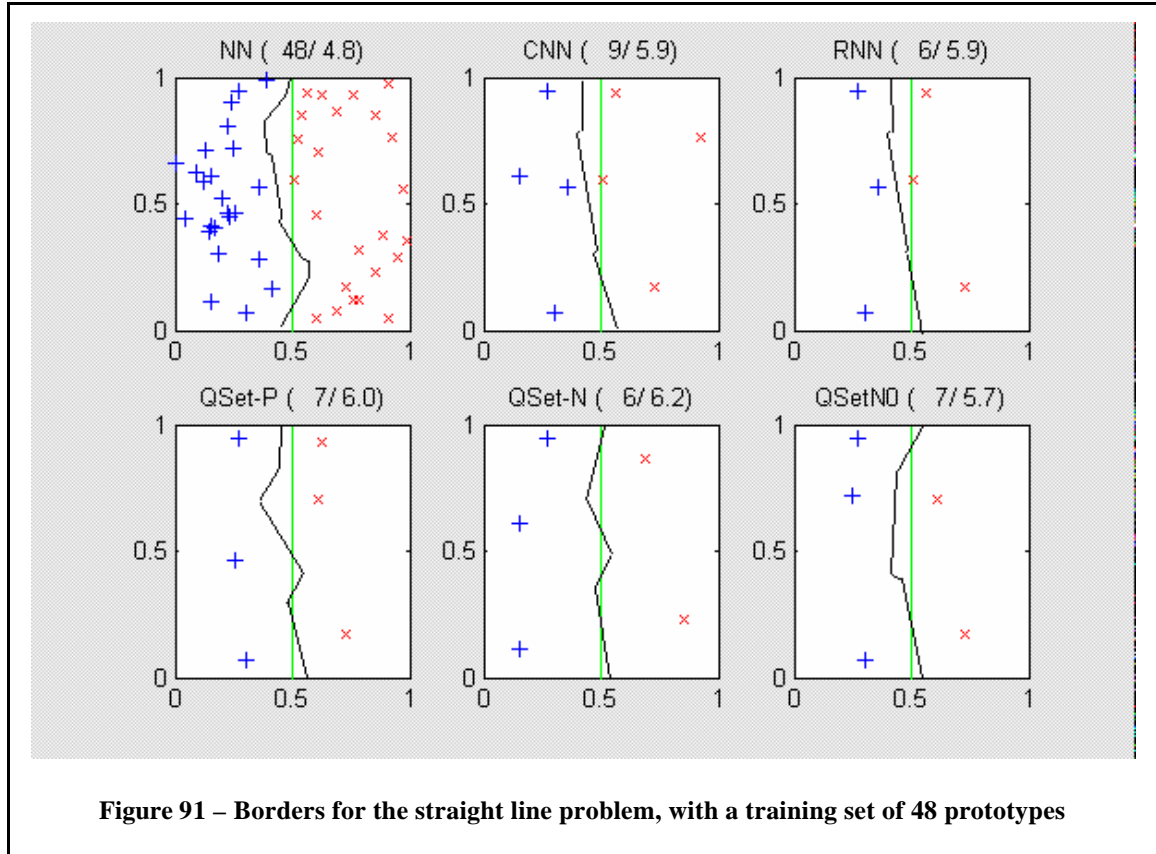
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	32.0 ± 0.0	5.7 ± 1.8	0.00 ± 0.00	3.58 ± 0.02
CNN	7.0 ± 2.0	6.9 ± 3.2	0.01 ± 0.02	0.86 ± 0.23
RNN	5.0 ± 1.3	7.5 ± 3.3	0.02 ± 0.03	0.64 ± 0.14
QSet-P	5.9 ± 1.2	7.4 ± 3.1	0.00 ± 0.01	0.76 ± 0.14
QSet-N	4.6 ± 1.4	7.1 ± 3.3	0.12 ± 0.08	0.60 ± 0.15
QSet-N ₀	5.4 ± 1.5	6.7 ± 2.7	0.06 ± 0.06	0.70 ± 0.16
QSet-BB	5.7 ± 1.3	7.4 ± 3.0	4.37 ± 10.97	0.73 ± 0.15

40 patterns



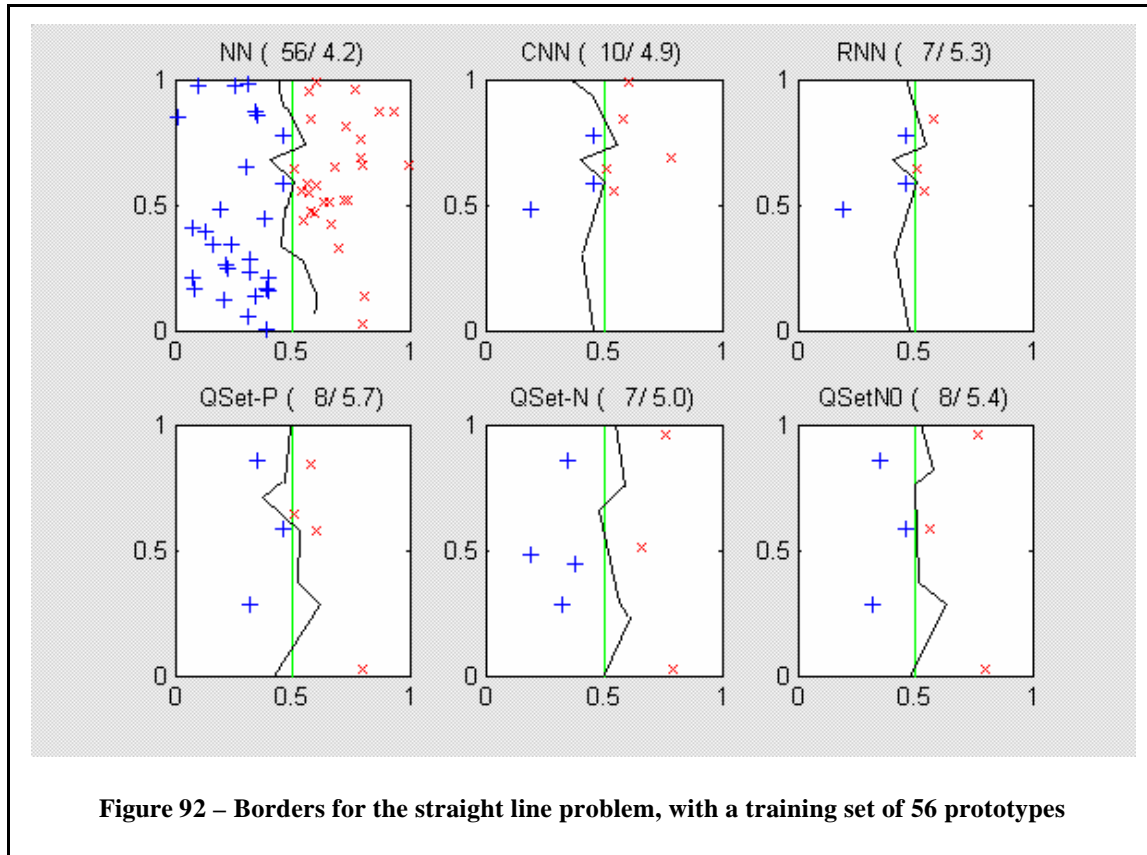
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	40.0 ± 0.0	5.3 ± 1.4	0.00 ± 0.00	4.44 ± 0.02
CNN	7.5 ± 2.6	6.7 ± 2.2	0.01 ± 0.02	0.92 ± 0.28
RNN	5.4 ± 1.6	6.7 ± 2.1	0.02 ± 0.03	0.68 ± 0.18
QSet-P	6.6 ± 1.0	6.6 ± 2.8	0.01 ± 0.02	0.82 ± 0.12
QSet-N	5.2 ± 1.2	6.8 ± 2.8	0.18 ± 0.13	0.67 ± 0.13
QSet-N ₀	6.1 ± 1.2	7.1 ± 2.5	0.09 ± 0.08	0.77 ± 0.13
QSet-BB	6.3 ± 1.0	6.6 ± 2.7	111.95 ± 422.53	0.79 ± 0.11

48 patterns



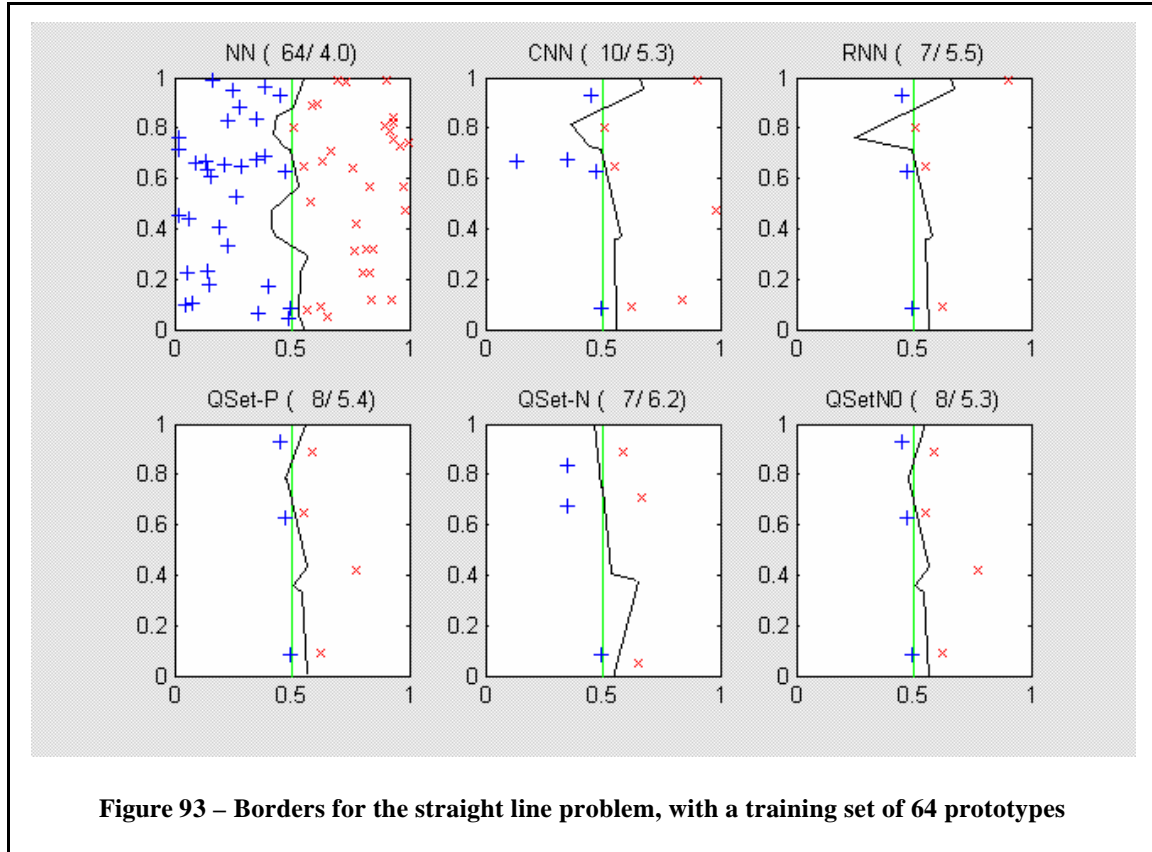
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	48.0 ± 0.0	4.8 ± 1.3	0.00 ± 0.00	5.34 ± 0.24
CNN	8.9 ± 1.9	5.9 ± 2.1	0.02 ± 0.03	1.04 ± 0.20
RNN	6.2 ± 1.3	5.9 ± 2.2	0.03 ± 0.03	0.75 ± 0.13
QSet-P	7.1 ± 1.3	6.0 ± 1.9	0.01 ± 0.02	0.85 ± 0.15
QSet-N	5.6 ± 1.2	6.2 ± 2.5	0.25 ± 0.17	0.69 ± 0.13
QSet-N ₀	6.8 ± 1.4	5.7 ± 2.0	0.11 ± 0.13	0.84 ± 0.17

56 patterns



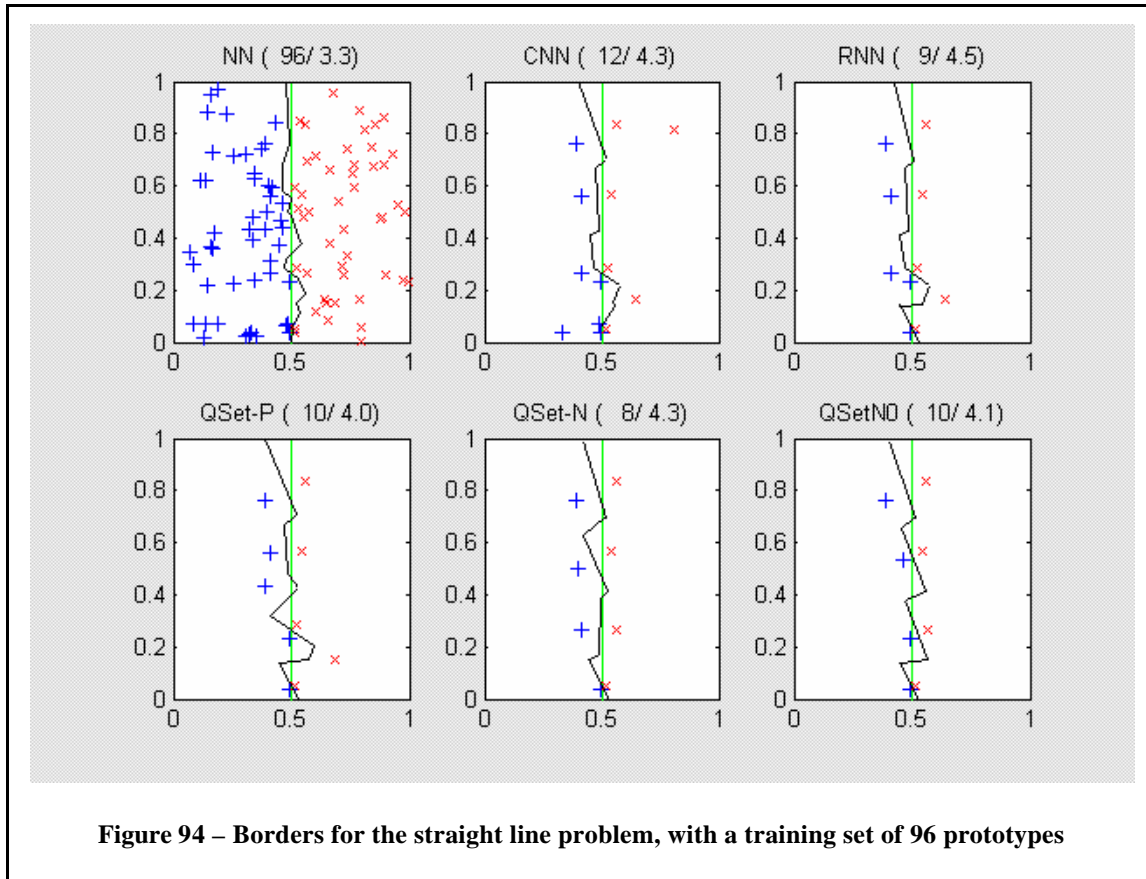
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	56.0 ± 0.0	4.2 ± 0.7	0.00 ± 0.00	6.19 ± 0.10
CNN	10.0 ± 2.9	4.9 ± 1.7	0.02 ± 0.03	1.16 ± 0.31
RNN	7.4 ± 2.0	5.3 ± 1.9	0.03 ± 0.03	0.89 ± 0.21
QSet-P	8.4 ± 1.5	5.7 ± 1.7	0.01 ± 0.02	1.00 ± 0.15
QSet-N	7.0 ± 1.7	5.0 ± 1.9	0.26 ± 0.19	0.85 ± 0.17
QSet-N ₀	7.9 ± 1.6	5.4 ± 1.5	0.14 ± 0.13	0.95 ± 0.20

64 patterns



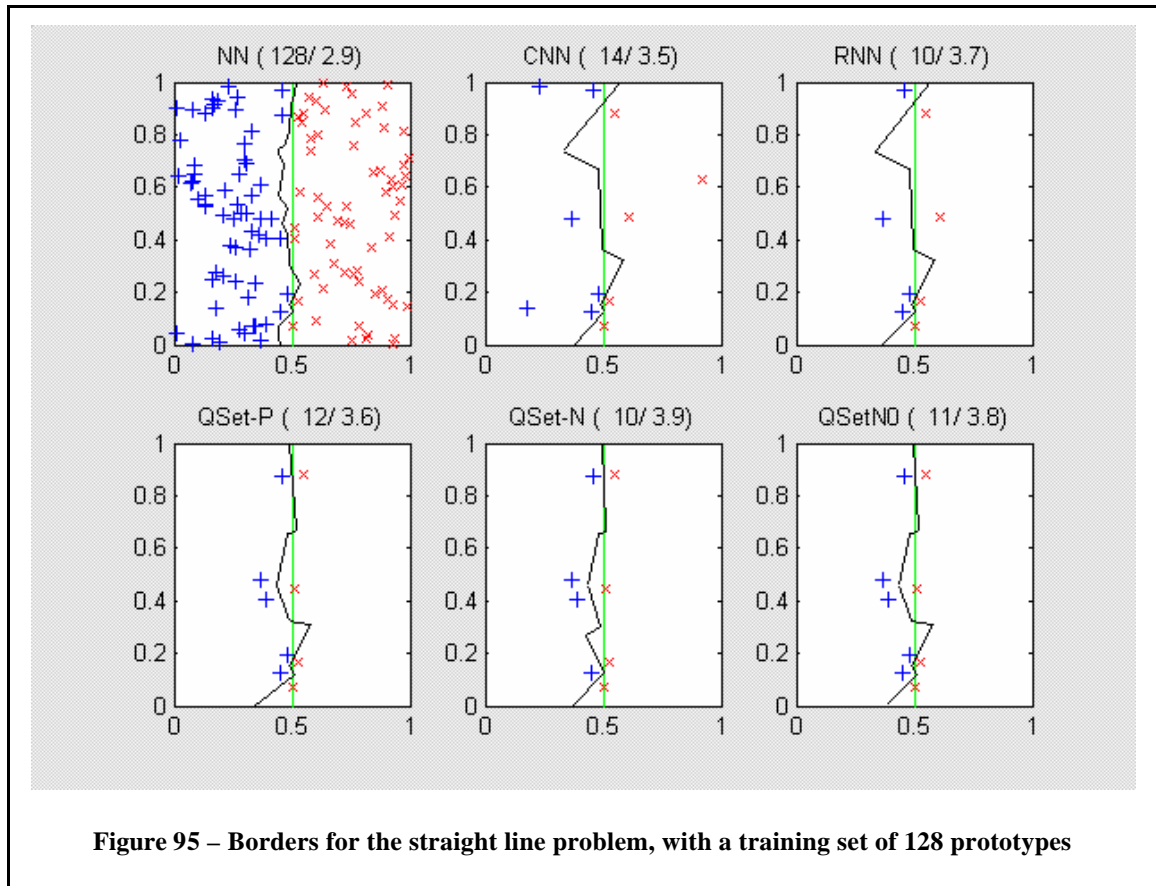
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	64.0 ± 0.0	4.0 ± 1.1	0.00 ± 0.00	7.17 ± 0.28
CNN	9.7 ± 2.9	5.3 ± 1.6	0.03 ± 0.03	1.13 ± 0.31
RNN	7.0 ± 1.7	5.5 ± 1.6	0.05 ± 0.04	0.84 ± 0.18
QSet-P	8.5 ± 1.5	5.4 ± 1.6	0.02 ± 0.03	1.00 ± 0.16
QSet-N	6.7 ± 1.6	6.2 ± 1.9	0.35 ± 0.25	0.80 ± 0.17
QSet-N ₀	8.2 ± 1.7	5.3 ± 1.5	0.15 ± 0.12	0.96 ± 0.17

96 patterns



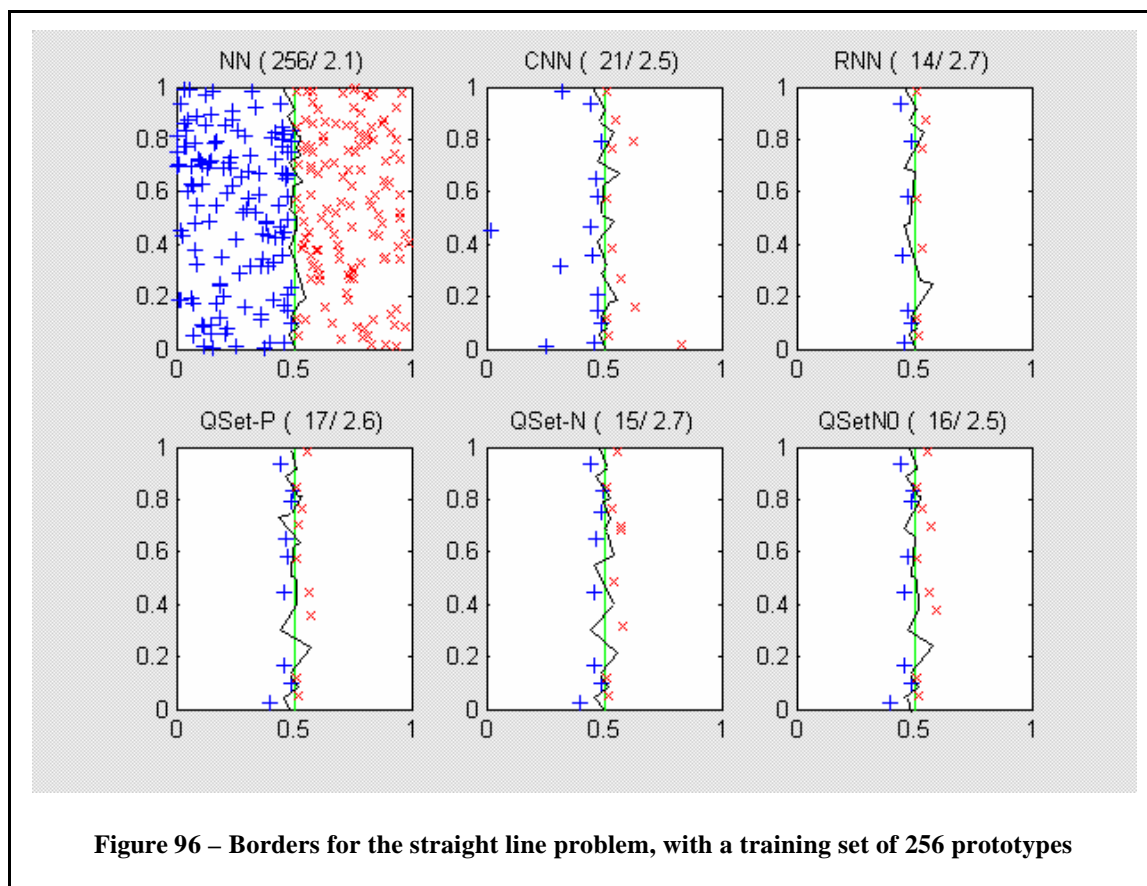
Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	96.0 ± 0.0	3.3 ± 0.8	0.00 ± 0.00	10.67 ± 0.05
CNN	12.0 ± 2.3	4.3 ± 1.1	0.04 ± 0.02	1.37 ± 0.24
RNN	8.5 ± 1.8	4.5 ± 1.4	0.07 ± 0.03	0.99 ± 0.18
QSet-P	9.9 ± 1.2	4.0 ± 1.2	0.03 ± 0.03	1.15 ± 0.13
QSet-N	8.3 ± 1.5	4.3 ± 1.4	0.63 ± 0.56	0.98 ± 0.16
QSet-N ₀	9.5 ± 1.1	4.1 ± 1.4	0.26 ± 0.13	1.11 ± 0.11

128 patterns



Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	128.0 ± 0.0	2.9 ± 0.7	0.00 ± 0.00	14.24 ± 0.05
CNN	14.3 ± 4.2	3.5 ± 0.9	0.06 ± 0.02	1.60 ± 0.44
RNN	9.7 ± 2.8	3.7 ± 0.9	0.11 ± 0.04	1.12 ± 0.29
QSet-P	12.2 ± 2.2	3.6 ± 0.8	0.05 ± 0.01	1.38 ± 0.23
QSet-N	10.3 ± 1.7	3.9 ± 0.9	1.81 ± 1.43	1.19 ± 0.18
QSet-N ₀	11.3 ± 2.2	3.8 ± 0.7	1.11 ± 1.14	1.29 ± 0.24

256 patterns



Method	N°Prototypes	Error rate	Training t /s	Test t /s
NN	256.0 ± 0.0	2.1 ± 0.4	0.00 ± 0.00	30.93 ± 12.25
CNN	21.4 ± 3.6	2.5 ± 0.5	0.15 ± 0.03	2.33 ± 0.38
RNN	14.3 ± 2.0	2.7 ± 0.6	0.31 ± 0.07	1.58 ± 0.21
QSet-P	17.0 ± 2.3	2.6 ± 0.4	0.19 ± 0.03	1.89 ± 0.24
QSet-N	14.9 ± 2.9	2.7 ± 0.6	10.35 ± 7.25	1.66 ± 0.30
QSet-N ₀	15.8 ± 2.3	2.5 ± 0.5	5.30 ± 2.88	1.75 ± 0.24

APPENDIX C

List of data recorded in the acoustical tank

Nº	File Name	Length /bytes	Duration /s	Target	Transient	Interference
1	A00000T00R01x1.wav	5292058	60.00066	None	-	Air (strong)
2	A00000T00R01x2.wav	5292058	60.00066	None	-	Air (strong)
3	A00000T00R02x1.wav	5292058	60.00066	None	-	Ar (weak)
4	A00000T00R02x2.wav	5292058	60.00066	None	-	Ar (weak)
5	A00000T00R08x1.wav	5292058	60.00066	None	-	Water (strong)
6	A00000T00R08x2.wav	5292058	60.00066	None	-	Water (strong)
7	A00000T00R04x1.wav	5292058	60.00066	None	-	Water (weak)
8	A00000T00R04x2.wav	5292058	60.00066	None	-	Water (weak)
9	A10000T00R00x1.wav	5292058	60.00066	Motor 1 (desengaged)	-	-
10	A10000T00R00x2.wav	5292058	60.00066	Motor 1 (desengaged)	-	-
11	A10000T00R00x3.wav	5292058	60.00066	Motor 1 (desengaged)	-	-
12	A10000T00R00x4.wav	5292058	60.00066	Motor 1 (desengaged)	-	-
13	A10000T00R00x5.wav	5292058	60.00066	Motor 1 (desengaged)	-	-
14	E20000T00R00x1.wav	5292058	60.00066	Motor 1(very slow)	-	-
15	E20000T00R00x2.wav	5292058	60.00066	Motor 1(very slow)	-	-
16	E20000T00R00x3.wav	5292058	60.00066	Motor 1(very slow)	-	-
17	E20000T00R00x4.wav	5292058	60.00066	Motor 1(very slow)	-	-
18	E20000T00R00x5.wav	5292058	60.00066	Motor 1(very slow)	-	-
19	E30000T00R00x1_R.wav	5292058	60.00066	Motor 1 (lento-ré)	-	-
20	A30000T00R00x1.wav	5292058	60.00066	Motor 1 (slow)	-	-
21	A30000T00R00x2.wav	5292058	60.00066	Motor 1 (slow)	-	-
22	A30000T00R00x3.wav	5292058	60.00066	Motor 1 (slow)	-	-
23	A30000T00R00x4.wav	5292058	60.00066	Motor 1 (slow)	-	-
24	A30000T00R00x5.wav	5292058	60.00066	Motor 1 (slow)	-	-
25	A40000T00R00x1.wav	5292058	60.00066	Motor 1 (half)	-	-
26	A40000T00R00x2.wav	5292058	60.00066	Motor 1 (half)	-	-
27	A40000T00R00x3.wav	5292058	60.00066	Motor 1 (half)	-	-
28	A40000T00R00x4.wav	5292058	60.00066	Motor 1 (half)	-	-
29	A40000T00R00x5.wav	5292058	60.00066	Motor 1 (half)	-	-
30	A20000T00R01x1.wav	5292058	60.00066	Motor 1(very slow)	-	Air (strong)
31	A20000T00R01x2.wav	5292058	60.00066	Motor 1(very slow)	-	Air (strong)
32	A20000T00R01x3.wav	5292058	60.00066	Motor 1(very slow)	-	Air (strong)
33	A20000T00R01x4.wav	5292058	60.00066	Motor 1(very slow)	-	Air (strong)
34	A20000T00R01x5.wav	5292058	60.00066	Motor 1(very slow)	-	Air (strong)
35	A20000T00R02x1.wav	5292058	60.00066	Motor 1(very slow)	-	Ar (weak)
36	A20000T00R02x2.wav	5292058	60.00066	Motor 1(very slow)	-	Ar (weak)

37	A20000T00R02x3.wav	5292058	60.00066	Motor 1(very slow)	-	Ar (weak)
38	A20000T00R02x4.wav	5292058	60.00066	Motor 1(very slow)	-	Ar (weak)
39	A20000T00R02x5.wav	5292058	60.00066	Motor 1(very slow)	-	Ar (weak)
40	A20000T00R04x1.wav	5292058	60.00066	Motor 1(very slow)	-	Water (weak)
41	A20000T00R04x2.wav	5292058	60.00066	Motor 1(very slow)	-	Water (weak)
42	A20000T00R04x3.wav	5292058	60.00066	Motor 1(very slow)	-	Water (weak)
43	A20000T00R04x4.wav	5292058	60.00066	Motor 1(very slow)	-	Water (weak)
44	A20000T00R04x5.wav	5292058	60.00066	Motor 1(very slow)	-	Water (weak)
45	A20000T00R08x1.wav	5292058	60.00066	Motor 1(very slow)	-	Water (strong)
46	A20000T00R08x2.wav	5292058	60.00066	Motor 1(very slow)	-	Water (strong)
47	A20000T00R08x3.wav	5292058	60.00066	Motor 1(very slow)	-	Water (strong)
48	A20000T00R08x4.wav	5292058	60.00066	Motor 1(very slow)	-	Water (strong)
49	A20000T00R08x5.wav	5292058	60.00066	Motor 1(very slow)	-	Water (strong)
50	A50000T00R00x1.wav	5292058	60.00066	Motor 5 (changing)	-	-
51	A50000T00R00x2.wav	5292058	60.00066	Motor 5 (changing)	-	-
52	A00000T16R00x01.wav	909538	10.31222	None	Gunshot	-
53	A00000T16R00x02.wav	915050	10.37472	None	Gunshot	-
54	A00000T16R00x03.wav	843394	9.56229	None	Gunshot	-
55	A00000T16R00x04.wav	915050	10.37472	None	Gunshot	-
56	A00000T16R00x05.wav	915050	10.37472	None	Gunshot	-
57	A00000T16R00x06.wav	893002	10.12474	None	Gunshot	-
58	A00000T16R00x07.wav	920562	10.43721	None	Gunshot	-
59	A00000T16R00x08.wav	915050	10.37472	None	Gunshot	-
60	A00000T16R00x09.wav	915050	10.37472	None	Gunshot	-
61	A00000T16R00x10.wav	837882	9.499796	None	Gunshot	-
62	A00000T16R00x11.wav	920562	10.43721	None	Gunshot	-
63	A00000T16R00x12.wav	920562	10.43721	None	Gunshot	-
64	A00000T16R00x13.wav	926074	10.49971	None	Gunshot	-
65	A00000T16R00x14.wav	926074	10.49971	None	Gunshot	-
66	A00000T16R00x15.wav	915050	10.37472	None	Gunshot	-
67	A00000T16R00x16.wav	926074	10.49971	None	Gunshot	-
68	A00000T16R00x17.wav	920562	10.43721	None	Gunshot	-
69	A00000T16R00x18.wav	920562	10.43721	None	Gunshot	-
70	A00000T16R00x19.wav	948122	10.74968	None	Gunshot	-
71	A00000T16R00x20.wav	920562	10.43721	None	Gunshot	-
72	A00000T16R00x21.wav	926074	10.49971	None	Gunshot	-
73	A00000T16R00x22.wav	843394	9.56229	None	Gunshot	-
74	A00000T16R00x23.wav	920562	10.43721	None	Gunshot	-
75	A00000T16R00x24.wav	909538	10.31222	None	Gunshot	-

76	A00000T16R00x25.wav	920562	10.43721	None	Gunshot	-
77	A00000T16R00x26.wav	920562	10.43721	None	Gunshot	-
78	A00000T16R00x27.wav	915050	10.37472	None	Gunshot	-
79	A00000T16R00x28.wav	920562	10.43721	None	Gunshot	-
80	A00000T16R00x29.wav	920562	10.43721	None	Gunshot	-
81	A00000T16R00x30.wav	920562	10.43721	None	Gunshot	-
82	A00000T16R00x31.wav	832370	9.437302	None	Gunshot	-
83	A00000T16R00x32.wav	898514	10.18723	None	Gunshot	-
84	A00100T00R00.wav	26507340	300.5367	Motor 3 (desengaged)	-	-
85	A00200T00R00.wav	26556948	301.0992	Motor 3(very slow)	-	-
86	A00300T00R00.wav	26512852	300.5992	Motor 3 (slow)	-	-
87	A00400T00R00.wav	26512852	300.5992	Motor 3 (half)	-	-
88	A00200T00R01.wav	26540412	300.9117	Motor 3(very slow)	-	Air (strong)
89	A00400T00R01.wav	26507340	300.5367	Motor 3 (half)	-	Air (strong)
90	A00200T00R08.wav	26512852	300.5992	Motor 3(very slow)	-	Water (strong)
91	A00400T00R08.wav	26667188	302.3491	Motor 3 (half)	-	Water (strong)
92	A00500T00R00x1.wav	5357796	60.74599	Motor 3 (changing)	-	-
93	A00500T00R00x2.wav	5335748	60.49601	Motor 3 (changing)	-	-
94	A01000T00R00.wav	26512852	300.5992	Motor 2 (desengaged)	-	-
95	A02000T00R00.wav	26540412	300.9117	Motor 2(very slow)	-	-
96	A03000T00R00.wav	26512852	300.5992	Motor 2 (slow)	-	-
97	A04000T00R00.wav	26507340	300.5367	Motor 2 (half)	-	-
98	A02000T00R01.wav	26545924	300.9742	Motor 2(very slow)	-	Air (strong)
99	A02000T00R08.wav	26518364	300.6617	Motor 2 (half)	-	Water (strong)
100	A04000T00R01.wav	26793964	303.7864	Motor 2(very slow)	-	Air (strong)
101	A04000T00R08.wav	26507340	300.5367	Motor 2 (half)	-	Water (strong)
102	A05000T00R00x1.wav	5352284	60.68349	Motor 2 (changing)	-	-
103	A05000T00R00x2.wav	5335748	60.49601	Motor 2 (changing)	-	-
104	A20000T00R00.wav	26518364	300.6617	Motor 1 (half)	-	-
105	A40000T00R01.wav	26540412	300.9117	Motor 1 (half)	-	Air (strong)
106	A40000T00R08.wav	26529388	300.7867	Motor 1 (half)	-	Water (strong)
107	A02000T16R00.wav	5292058	60.00066	Motor 2(very slow)	Gunshot	-
108	A02000T08R00.wav	5292058	60.00066	Motor 2(very slow)	Metal hammer	-
109	A02000T04R00.wav	5292058	60.00066	Motor 2(very slow)	Rubber hammer	-
110	A02000T01R00.wav	5292058	60.00066	Motor 2(very slow)	Air	-
111	A02000T02R00.wav	5292058	60.00066	Motor 2(very slow)	Bucket of water	-
112	A22000T00R00.wav	5292058	60.00066	Motor1+Motor2	-	-
113	A02200T00R00.wav	5292058	60.00066	Motor2+Motor3	-	-
114	E02040T00R00.wav	5292058	60.00066	Motor2+Motor4	-	-

115	A00200T16R00.wav	5292058	60.00066	Motor 3(very slow)	Gunshot	-
116	E00220T00R00.wav	5292058	60.00066	Motor3+Motor4	-	-
117	A00200T08R00.wav	5292058	60.00066	Motor 3(very slow)	Metal hammer	-
118	A00200T04R00.wav	5292058	60.00066	Motor 3(very slow)	Rubber hammer	-
119	A00200T01R00.wav	5292058	60.00066	Motor 3(very slow)	Air	-
120	A00200T02R00.wav	5292058	60.00066	Motor 3(very slow)	Bucket of water	-
121	A02002T00R00.wav	5292058	60.00066	Motor2+Model boat	-	-
122	A02020T00R00.wav	5292058	60.00066	Motor2+Motor4	-	-
123	A20200T00R00.wav	5292058	60.00066	Motor1+Motor3	-	-
124	A00220T00R00.wav	5292058	60.00066	Motor3+Motor4	-	-
125	E00200T00R00.wav	5292058	60.00066	Motor 3(very slow)	-	-
126	A20220T00R00.wav	5292058	60.00066	Motor1+3+4	-	-
127	A20020T00R00.wav	5292058	60.00066	Motor1+Motor3	-	-
128	E20000T16R00.wav	5292058	60.00066	Motor 1(very slow)	Gunshot	-
129	A20000T16R00.wav	5292058	60.00066	Motor 1(very slow)	Gunshot	-
130	A20000T08R00.wav	5292058	60.00066	Motor 1(very slow)	Metal hammer	-
131	A20000T04R00.wav	5292058	60.00066	Motor 1(very slow)	Rubber hammer	-
132	A20000T01R00.wav	5292058	60.00066	Motor 1(very slow)	Air	-
133	A20000T02R00.wav	5292058	60.00066	Motor 1(very slow)	Bucket of water	-
134	A00010T00R00.wav	31528772	357.4691	Motor 4 (desengaged)	-	-
135	A00020T00R00.wav	32702828	370.7804	Motor 4(very slow)	-	-
136	A00030T00R00.wav	39603852	449.0233	Motor 4 (slow)	-	-
137	A00040T00R00.wav	29097980	329.9091	Motor 4 (half)	-	-
138	A00020T00R01.wav	26667188	302.3491	Motor 4(very slow)	-	Air (strong)
139	E00040T00R01.wav	9144540	103.6796	Motor 4 (half)	-	Air (strong)
140	A00040T00R01.wav	27312092	309.6609	Motor 4 (half)	-	Air (strong)
141	A00020T00R08.wav	26523876	300.7242	Motor 4(very slow)	-	Water (strong)
142	A00040T00R08.wav	26738844	303.1615	Motor 4 (half)	-	Water (strong)
143	A00020T16R00.wav	5065586	57.43295	Motor 4(very slow)	Gunshot	-
144	A00020T08R00.wav	5292058	60.00066	Motor 4(very slow)	Metal hammer	-
145	A00020T04R00.wav	5275042	59.80773	Motor 4(very slow)	Rubber hammer	-
146	A00020T01R00.wav	5292058	60.00066	Motor 4(very slow)	Air	-
147	A00020T02R00.wav	5292058	60.00066	Motor 4(very slow)	Bucket of water	-
148	A00011T00R00.wav	5292058	60.00066	Motor4+Model boat	-	-
149	A00022T00R00.wav	5292058	60.00066	Motor3+Model boat	-	-
150	A20002T00R00.wav	5292058	60.00066	Motor 1 (half)	-	-
151	A00000T08R00.wav	28601900	324.2846	None	Metal hammer	-
152	A00000T04R00.wav	28353860	321.4723	None	Rubber hammer	-
153	A00000T01R00.wav	28342836	321.3473	None	Air	-

154	A00000T02R00.wav	29368068	332.9713	None	Bucket of water	-
155	A00002T16R00.wav	5292058	60.00066	Model boat	Gunshot	-
156	A00002T08R00.wav	5292058	60.00066	Model boat	Metal hammer	-
157	A00002T04R00.wav	5292058	60.00066	Model boat	Rubber hammer	-
158	A00002T01R00.wav	5292058	60.00066	Model boat	Air	-
159	A00002T02R00.wav	5292058	60.00066	Model boat	Bucket of water	-
160	A00001T00R00.wav	5335748	60.49601	Model boat (slow)	-	-
161	A00002T00R00.wav	5539692	62.8083	Model boat (rapido)	-	-
162	A00003T00R00.wav	5335748	60.49601	Model boat (re lento)	-	-
163	A00004T00R00.wav	5886948	66.74544	Model boat (re rápido)	-	-
164	A00005T00R00.wav	6945252	78.74435	None	-	-
165	E00006T00R00.wav	7292508	82.6815	None	-	-
166	A00006T00R00.wav	5341260	60.5585	None	-	-
167	A00000T00R09.wav	26601044	301.5991	None	-	Air + Water
168	A00000T00R01.wav	1.12E+08	1265.699	None	-	Air (strong)

APPENDIX D

Overview of the signals recorded in the acoustic tank

This appendix presents an overview of the signals recorded in the acoustic tank, as described in Chapter 4 of part III, and used in the experiments contained in that Chapter.

Each data pattern was extracted from approximately 3 s of raw signal. The total number of patterns available is presented in Table 12.

Effect	N° of Patterns	Time	Size /MB
motor 1	1263	1 h 03 min	333.606
motor 2	949	47 min	249.587
motor 3	968	48 min	254.763
motor 4	1045	52 min	275.536
motor (background)	5 1291	1 h 04 min	343.890
TOTAL	5516	5 h 01 min	1.457.382

Table 23 - General information about the Acoustic Tank data. The number of patterns correspond to 3 s segments of the original signal. These will later be subject to different feature extraction techniques, to produce the final patterns.

The plots presented in the following pages were obtained by computing the power spectra of each raw pattern with 8192 points (4096 positive frequency bins), corresponding to 5.3 Hz per bin from 0 to 22 kHz. A Hamming window and 50% overlap Welch periodograms were used, so each spectrum used is the average of 32 individual spectra.

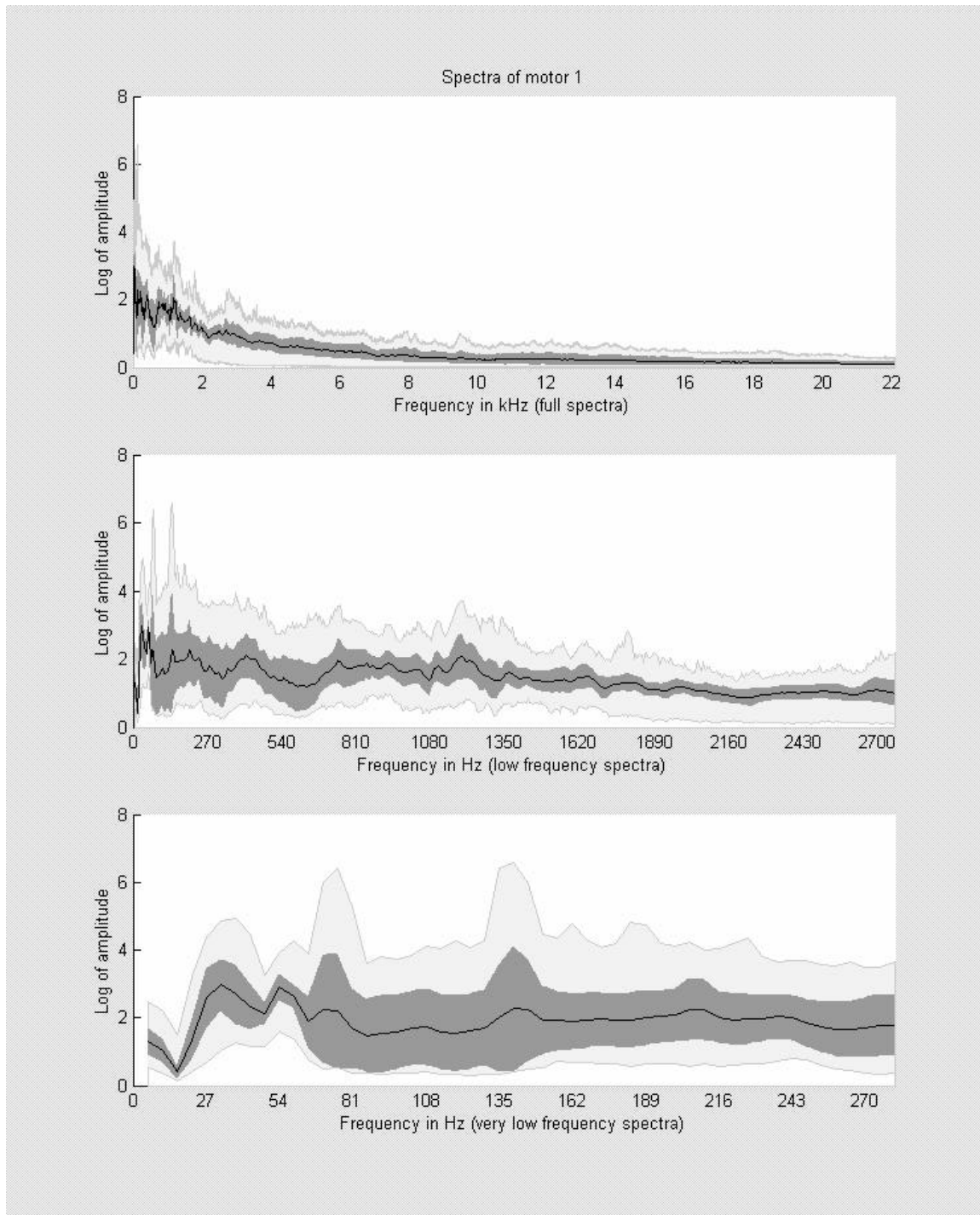


Figure 97 - Spectra of Motor 1. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values)

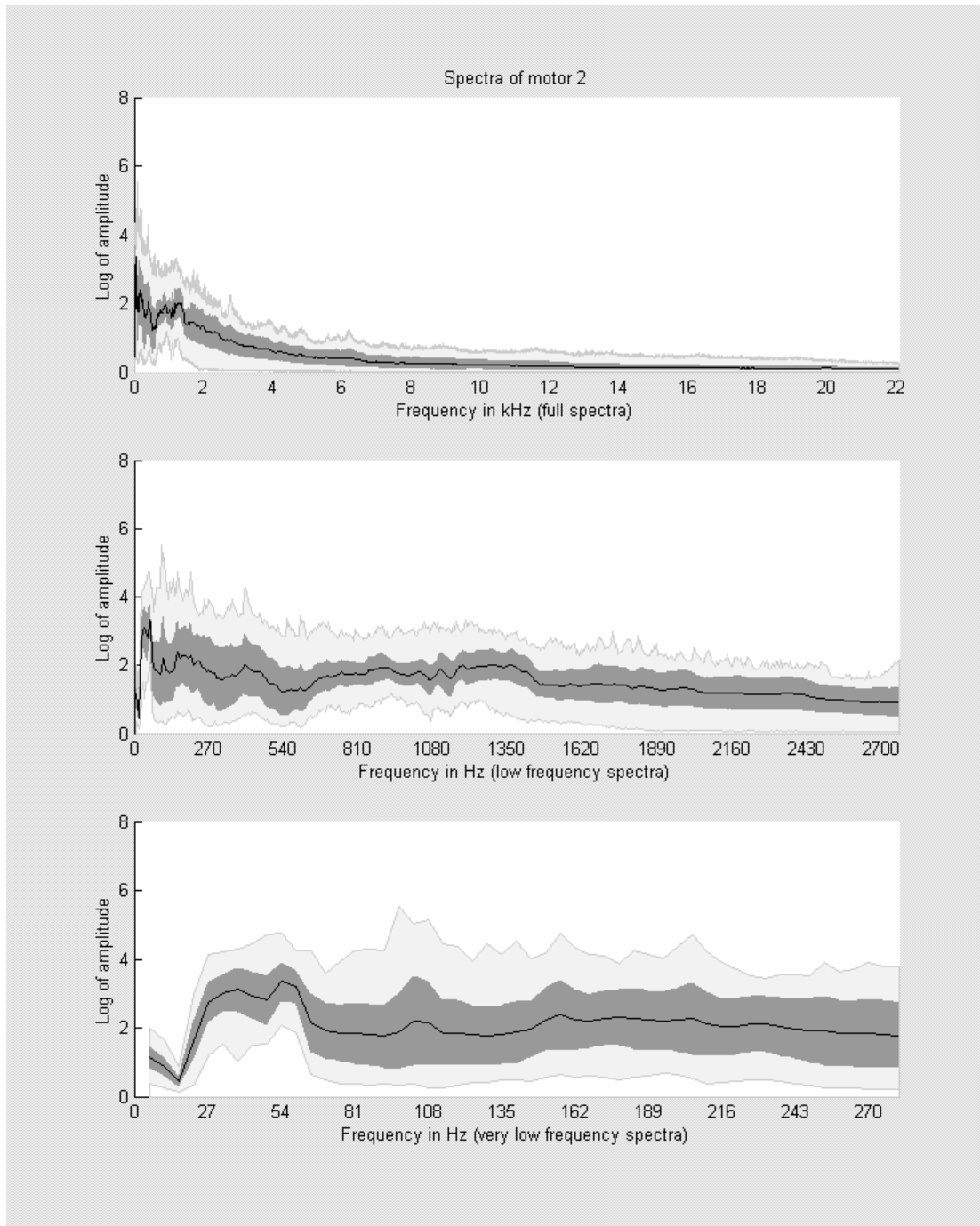


Figure 98 -Spectra of Motor 2. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).

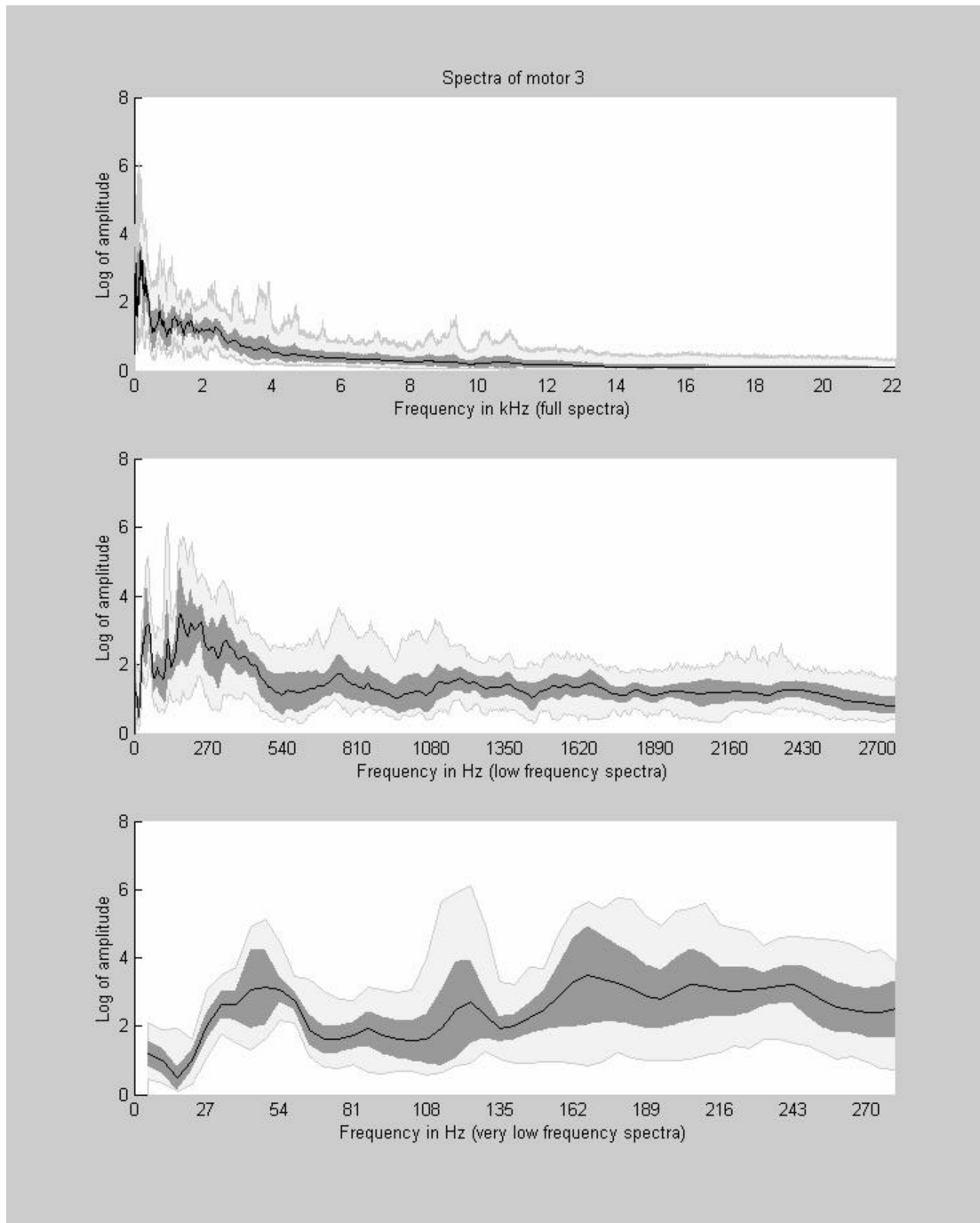


Figure 99 -Spectra of Motor 3. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).

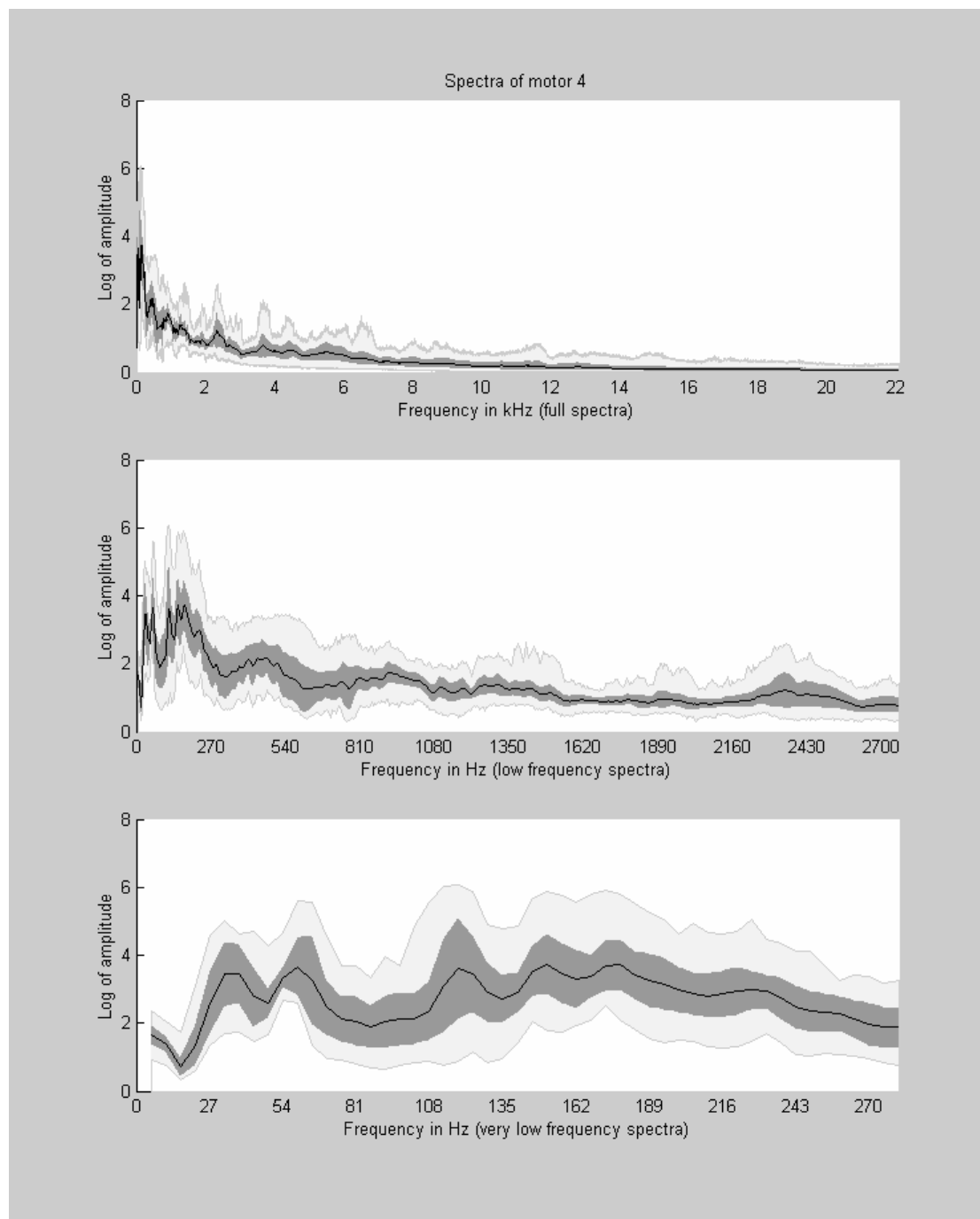


Figure 100 - Spectra of Motor 4. The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).

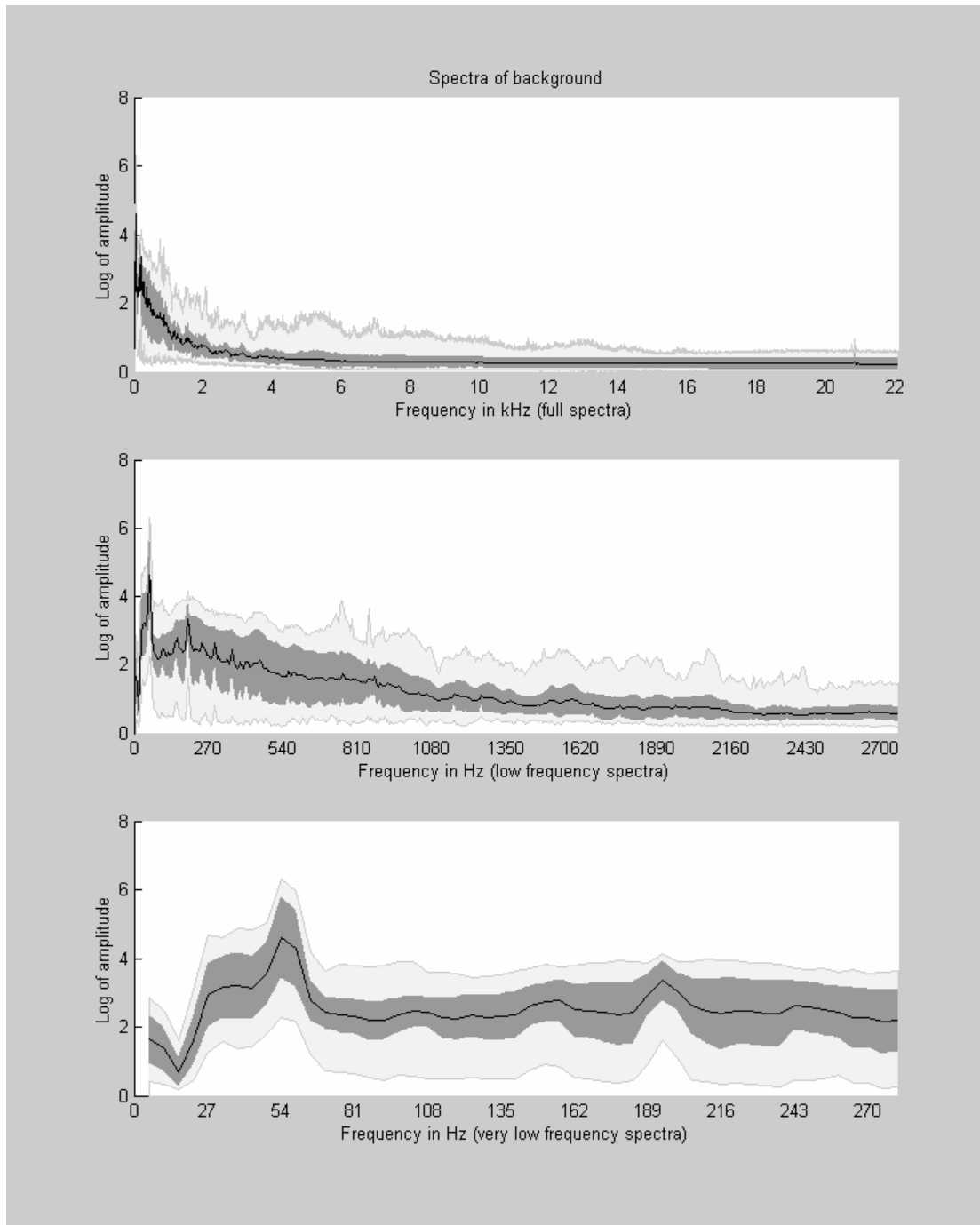


Figure 101 - Spectra of background noise (or motor 5). The black line represents the average, the dark gray area represents the region of average \pm standard deviation, and the light gray area encompasses all observed signals (from maximum to minimum values).

APPENDIX E

Matlab routines

```

function qs=qs_mat_build( prototype, label_prototype, pattern ,label_pattern )
% qs=qs_mat_build( prototypes, label_prototypes,patterns ,label_patterns )
%
% OBJECTIVE
%   Build the Q-set matrix for the patterns in PATTERNS. This routine
%   caclulates the Q-set of order 0, necessary for the positive-only
%   Q-set approach descibed in (Lobo).
%
% INPUT PARAMETERS
%   prototype       - Candidate prototypes (one per column)
%   label_prototype - Labels of the prototypes
%   pattern         - Data patterns (one per column) for which the
%                   Q matrix is calculated
%   label_pattern   - Labels of the patterns
%
% OUTPUT PARAMETERS
%   qs              - Boolean matrix where each row corresponds to
%                   a data pattern, and each column to a prototype
%                   A value of 1 indicates that the prototype
%                   corresponding to that columns belongs to the
%                   Q-set of the data pattern correspondng to the row
%
% V.1.0.0 - 00-JUN-2000 - V.Lobo, Home
% V.1.1.0 - 08-AUG-2002 - V.Lobo, SDSU

[numFeatures,numPatterns]=size(pattern);
[numFeatures2,numPrototypes]=size(prototype);
if numFeatures ~= numFeatures2
    disp('ERROR in qs_mat_build: number of features does not agree.');
```

return ;

```

end;

d = dist(pattern',prototype);
                                % sameClass is 1 if pattern and prototype
                                % have the same label

sameClass      =      (repmat(label_prototype,numPatterns,1)
(repmat(label_pattern,numPrototypes,1))');
                                % we shall now find the nearest prototype with
                                % WRONG class
```

```
warning off;  
wrongDistances = min((d ./ ~sameClass )')';  
warning on;  
qs= sameClass & (d<repmat(wrongDistances,1,numPrototypes));
```

```

function chosenPrototypes=qs_select_heuristic(qset)
% chosenPrototypes=qs_select_heuristic(qset)
%
% OBJECTIVE
%   This function selects the classifying prototypes, given
%   a matrix with their Q-sets, using the positive-only
%   heuristic descibed in (Lobo)
%
% INPUT PARAMETERS
%   qset      Binary matrix with qsets, produced by "qs_mat_build"
%
% OUTPUT PARAMETERS
%   chosenPrototypes  Indexes of the chosen prototypes
%
% COMMENTS
%
% V.1.0.0 - 00-000-2001 - V.Lobo
% V.1.1.0 - 08-AUG-2002 - V.Lobo

[numPatterns,numPrototypes]=size(qset);
indexPrototypes=1:numPrototypes;

% find how many prorotypes classify each pattern
sumPrototypes=sum(qset,2);

% Find patterns that are classified by a single prototype
singlyClassifiedPatterns = find(sumPrototypes==1);

% Store the candidate Prototypes that classify those patterns
[dummy,candidatePrototype] = find( qset(singlyClassifiedPatterns,:)==1 );

% Remove repeated prorotypes from the list
candidatePrototype = unique(candidatePrototype);

% make the choice permanent
chosenPrototypes = candidatePrototype;
[numChosenPrototypes,dummy] = size(chosenPrototypes);

% clear the qs_matrice of the already dealt with patterns
remainingPatterns = find( sumPrototypes > 1 );
qset = qset(remainingPatterns,:);

```

```
% find the indexes of patterns already covered by the chosen prototypes
tmpQset=qset(:,chosenPrototypes);
sumPrototypes=sum(tmpQset,2);
remainingPatterns = find( sumPrototypes < 1 );
qset = qset(remainingPatterns,:);
    % At this moment, all patterns in QSET are classified by at least 2 protot.

while( size(qset)>0 )
    frequency = sum(qset,1);
    [dummy,candidatePrototype]=max(frequency);
    remainingPatterns = find( qset(:,candidatePrototype) == 0 );
    qset = qset(remainingPatterns,:);
    chosenPrototypes = [chosenPrototypes ; candidatePrototype ];
end;
```



```
[lixo, qsgc ] = sort( d, 2 );

for i=1:numPatterns
    tmp=label_prototype(qsgc(i,:));
    qsgc_lv(i,:)=tmp==label_pattern(i);
end;
```

```

function [qs,removedP]=g2p( qsgc,qsgc_lv,amer )
% [qs,removedP]=g2p( qsgc,qsgc_lv,amer
%
% OBJECTIVE
%   Transform a general-case Q-set to a positive-only, usin amer
%
% INPUT PARAMETERS:
%   qsgc      - Q-Set-General-Case: Matrix with the indexes of the
%               nearest prototypes, sorted by distance. Each row
%               corresponds to a given pattern
%   qsgc_lv  -Q-Set-General-Case-Logical-Values: companion matrix
%               to qsgc, has 1 if the corresponding variable is
%               affirmative (same classes), and 0 it they are not
%               (different classes)
%   amer      - Maximum number of errors allowed
%
% OUTPUT PARAMETERS
%   qs        - Boolean matrix with the positive-only q-sets
%   removedP- Indexes of the patterns removed
%
% V.1.0.0 - 08-APR-2002 - V.Lobo, SDSU

% Variables:
%   gsgc_valid - boolean matrix that is a companion to qsgc and qsgc_lv
%               if it is 1, that combination of pattern/prototype is sill
%               possible, because the prototype hasen't been excluded

[numPatterns,numPrototypes]=size(qsgc);

qsgc_valid=ones(numPatterns,numPrototypes);
P=1:numPrototypes; % indexes of prototypes that can be included
removedP=[];       % indexes of prototypes that are excluded

%calculate the original error rate
errors=not(qsgc_lv(:,1)); % if the first entry in the Qset has the same
class
% then there is no error. Otherwise, there is. The
% variable "errors" is a vector that that has a
"1"
% in the positions corresponing to patterns that
are

```

```

                                % badly classified

validPatterns=find(errors==0); % "validPatterns" contains indexes of valid
Patt.
numValidPatterns=length(validPatterns);
numErrors=numPatterns-numValidPatterns;

oldRemovedP=removedP;           % we might have to backtrack 1 step
oldqsgc_valid=qsgc_valid;
olderrors=errors;
oldvalidPatterns=validPatterns;

while numErrors <= amer

% First search for obvious candidates for exclusion
candidateP=zeros(1,numValidPatterns);
for i=validPatterns'           % Search all still valid patterns
    bad=and(qsgc_valid(i,:),not(qsgc_lv(i,:)));
    ibad=find(bad==1);         % indexes of prototypes with wrong class
    thisCandidate=ibad(1);     % index of FIRST prototype with wrong class
    candidateP(i)= qsgc(i,thisCandidate); % add that prototype to the
candidate list
end;

candidateP=unique(candidateP); % Remove duplicate candidates
if candidateP(1)==0           % pathological case 1: one pattern has no bad prot.
    candidateP=candidateP(2:end);
end;
if candidateP==[]           % pathological case 2: there are no mode candidates
    break;
end;

% calculate min cost/benefit for each of the candidates
minCB=10000;                 % dummy initial value for minimum Cost/Benefit found
minCBP=0;                    % dummy initial value for the index of the best prototype
for j=candidateP             % for every candidate...
    cost=1;
    benefit=1;
    for i=validPatterns'     % check effect on all remaining Patterns
        position=find( qsgc(i,:)==j); % position of candidate prototype in
Qset

```

```

if qsgc_lv(i,position)==1
    %if they have the SAME class
    valid=qsgc_valid(i,1:position-1);
    right=qsgc_lv(i,1:position-1);
    previous_right=and(valid,right);
    if not(any(previous_right))
        %if it is the first right
        if qsgc_lv(i,position+1)==0
            % if the next has the wrong label
            % in this case, the cost would be increased
            cost=cost+1;
        end;
    end;
else % if they have different classes
    %find out if it's the first of the wrong class
    valid=qsgc_valid(i,1:position-1);
    wrong=not(qsgc_lv(i,1:position-1));
    previous_wrong=and(valid,wrong);
    if not(any(previous_wrong))
        % if this is the first wrong
        % we will count how may benefits we have
        % 1-go to the next valid
        position=position+1;
        while      ((position      <      numPrototypes)      &
(qsgc_valid(i,position)==0) )
            position=position+1;
        end;
        while (position < numPrototypes) & (qsgc_lv(i,position)==1)
            benefit=benefit+1;
            position=position+1;
            while      (position      <      numPrototypes)      &
(qsgc_valid(i,position)==0)
                position=position+1;
            end;
        end;
    end;
end;
end; % end of for validPatterns (the effect of this prototype "j" on all
% remaining patterns is accounted for.

% We shall now see if it is better than all previous candidates...

```

```

    CB=cost/benefit;
    if CB<minCB
        minCB=CB;
        minP=j;
    end;
end;
%We have now selected the best prototype

oldRemovedP=removedP;           % we might have to backtrack 1 step
oldqsgc_valid=qsgc_valid;
olderrors=errors;
oldvalidPatterns=validPatterns;

removedP=union(removedP,minP);   % we add it to the removed set;
P = setdiff(P,minP);           % ... and remove it from the prototype

%We will now remove it from the Qsets and which are not errors
errors=zeros(1,numPatterns);
for i=1:numPatterns'
    removedIndex=find( qsgc(i,:)== minP );
    qsgc_valid(i,removedIndex)=0;
    % now lets check if Ro={}
    good=and(qsgc_valid(i,:),qsgc_lv(i,:));
    igood=find(good==1);
    if igood==[]
        errors(i)=1;
    else
        igood=igood(1);
        bad=and(qsgc_valid(i,:),not(qsgc_lv(i,:)));
        ibad=find(bad==1);
        if ibad~=[]
            ibad=ibad(1);
            if ibad < igood
                % in this case there is a bad one before the good one
                errors(i)=1;
            end;
        end;
    end;
end;
end;

% calculate the error

```

```
validPatterns=find(errors==0)';
numErrors=numPatterns-length(validPatterns);

end; % do amer...

% now we must construct the boolean positive-only functions
errors=olderrors;
qsgc_valid=oldqsgc_valid;
validPatterns=oldvalidPatterns;

% initialize qs
qs=zeros(length(validPatterns),numPrototypes);

% fill qs eith the appropriate ones
for i=1:length(validPatterns)
    patternIndex=validPatterns(i);
    bad=and(qsgc_valid(patternIndex,:),not(qsgc_lv(patternIndex,:)));
    ibad=find(bad==1);
    ibad=ibad(1);
    good=and(qsgc_valid(patternIndex,1:ibad-1),qsgc_lv(patternIndex,1:ibad-
1));
    igood=find(good==1);
    iprototypes=qsgc(patternIndex,igood);
    qs(i,iprototypes)=1;
end;

return;
```

```
function [chosenPrototypes]=qs_select(qset)
% [chosenPrototypes]=qs_select(qset)
%
% OBJECTIVE
%   This function selects the classifying prototypes, given
%   a matrix with their Q-sets, using branch-and-bound described
%   in [Lobo 2002]
%
% INPUT PARAMETERS
%   qset      Binary matrix with qsets, produced by "qs_mat_build"
%
% OUTPUT PARAMETERS
%   chosenPrototypes   Indexes of the chosen prototypes
%
% COMMENTS
%   This functions contains "sub-functions", and uses global variables
%
% V.1.0.0 - 00-000-2001 - V.Lobo

global originalQs;
global indexPrototypes;
global stopCost;      % cost of bestsolution MINUS 1
global selectedPrototypes

originalQs = qset;
clear qset;

[numPatterns,numPrototypes]=size(originalQs);
indexPrototypes=1:numPrototypes;

% find how many prorotypes classify each pattern
sumPrototypes=sum(originalQs,2);

% Find patterns that are classified by a single prototype
singlyClassifiedPatterns = find(sumPrototypes==1);

% Store the candidate Prototypes that classify those patterns
[dummy,candidatePrototype] = find( originalQs(singlyClassifiedPatterns,:)==1
);

% Remove repeated prorotypes from the list
```

```

candidatePrototype = unique(candidatePrototype);

    % make the choice permanent
selectedPrototypes = candidatePrototype;
[numChosenPrototypes,dummy] = size(selectedPrototypes);

    % clear the qs_matrice of the already dealt with patterns
remainingPatterns = find( sumPrototypes > 1 );
originalQs = originalQs(remainingPatterns,:);

    % find the indexes of patterns already covered by the chosen prototypes
tmpQset=originalQs(:,selectedPrototypes);
sumPrototypes=sum(tmpQset,2);
remainingPatterns = find( sumPrototypes < 1 );
if ~isempty(remainingPatterns)
    % originalQs = originalQs(remainingPatterns,:);
    remainingPrototypes = setdiff(indexPrototypes,selectedPrototypes);
    stopCost = inf ;
    IterateSearch(remainingPatterns,remainingPrototypes,selectedPrototypes,1)
end;

chosenPrototypes = selectedPrototypes;

return
%-----
--
% now the iterative part
%-----
--
function
IterateSearch(remainingPatterns,remainingPrototypes,candidateSelectedPrototype
s,cost)
% ITERATIVE SEARCH - Iterative part of the QS selection with Branch&Bound
%
global originalQs;
global indexPrototypes;
global stopCost;          % cost of bestsolution MINUS 1
global selectedPrototypes

frequency = sum( originalQs(remainingPatterns,remainingPrototypes),1 );

```

```

    % frequency will contain the relative frequencies of remaining prot.
localPrototypes = indexPrototypes(remainingPrototypes);
    % localPrototypes will contain the numbers of the remaining prototypes,
    % in the same order as they appear in frequency

[maxFrequency,candidatePrototypePosition]= max(frequency);

while( maxFrequency > 0 )
    % select that candidate prototype
    candidatePrototype = localPrototypes( candidatePrototypePosition );

    removedPatterns = find( originalQs(:,candidatePrototype));
    localRemainingPatterns = setdiff(remainingPatterns,removedPatterns);
    if isempty(localRemainingPatterns)
        % in this case, the search has ended.
        stopCost = cost-1;
        selectedPrototypes = [ candidateSelectedPrototypes ; candidatePrototype
];
        %disp('found a new solution');
        %selectedPrototypes'
        return;
    end;
    % if the set isn't empty, is it worthwhile continuing this branch ?
    if cost < stopCost
        % if all is OK, let us iterate down one level
        localRemainingPrototypes
=
setdiff(remainingPrototypes,candidatePrototype);

        IterateSearch(localRemainingPatterns,localRemainingPrototypes,[candidate
SelectedPrototypes ; candidatePrototype],cost+1);
    end;
    if cost == stopCost
        return; %in this case, it's no use trying more at this level
    end;
    frequency(candidatePrototypePosition)=0;
    [maxFrequency,candidatePrototypePosition]= max(frequency);
end;

%disp('finished a branch');
return;

```

```

function [cnn,cnn_label]=cnn( train, train_label )
% [cnn,cnn_label]=Cnn(train, train_label )
%
% OBJECTIVE
% This function selects the Condensed Nearest Neighbor classifying
% patterns, according to the CNN rule given in [Hart 67]
%
% INPUT PARAMETERS
% train          Matrix with the initial training set
%                (one pattern per column)
% train_label    Row vector with the labels (1:Nclasses)
%                of the training set
%
% OUTPUT PARAMETERS
% cnn            Matrix with de CNN (one pattern per column)
% cnn_label      Row vector with the labels of cnn
%
% COMMENTS
% Uses the knn function by VSL
%
% V.1.0.0 - 00-FEB-2000 - V.Lobo

[num_features,num_patterns]=size(train);

% The first CNN is the first training pattern.
cnn = train(:,1);
cnn_label = train_label(1);
additions=1;
while additions~=0
    additions=0;
    for k=2:num_patterns
        class = knn( cnn, cnn_label, train(:,k), 1);
        if class ~= train_label(k)
            cnn=[cnn train(:,k)];
            cnn_label=[cnn_label train_label(k)];
            additions=1;
        end;
    end;
end;
end;

```

```

function [rnn,rnn_label]=rnn(train,train_label,cnn,cnn_label)
% [rnn,rnn_label]=rnn(train,train_label,cnn,cnn_label)
%
% OBJECTIVE
% This function selects the Reduced Nearest Neighbor classification
% set from a previously obtained Condensed Nearest Neighbor set,
% according to the RNN rule given in [Gates 72]
%
% INPUT PARAMETERS
% train          Matrix with the initial training set
%                (one pattern per column)
% train_label    Row vector with the labels (1:Nclasses)
%                of the training set
% cnn            Matrix with the CNN
%                (one pattern per column)
% cnn_label      Row vector with the labels (1:Nclasses)
%                of the CNN
%
% OUTPUT PARAMETERS
% rnn            Matrix with de RNN (one pattern per column)
% rnn_label      Row vector with the labels of rnn
%
% COMMENTS
%
% V.1.0.0 - 00-FEB-2000 - V.Lobo

[num_features,num_cnn]=size(cnn);
rnn = cnn;
rnn_label=cnn_label;
rnn_index=1;
for m=1:num_cnn
    try_rnn = Remove_col(rnn,rnn_index);          % remove pattern
    try_rnn_label = Remove_col(rnn_label,rnn_index); % remove label
    c=knn_mat(try_rnn,try_rnn_label,train);      % classify all patterns with
                                                % new set
    if c==train_label                            % if there are no errors...
        rnn=try_rnn;                             % Accept the new RNN
        rnn_label = try_rnn_label;
    else                                         % If not...
        rnn_index=rnn_index+1;                  % pass on to next candidate
    end;
end;

```

```
end;
```

```
return;
```

```

function [vxx,vy] = voronoi_boundary(x,y,class,arg3,arg4)
%VORONOI Voronoi boundary diagram.
%   VORONOI_BOUNDARY(X,Y,CLASS) plots the Voronoi diagram for the points X,Y.
%
%   VORONOI(X,Y,TRI) uses the triangulation TRI instead of
%   computing it via DELAUNAY.
%
%   H = VORONOI(...,'LineStyle') plots the diagram with color and linestyle
%   specified and returns handles to the line objects created in H.
%
%   [VX,VY] = VORONOI(...) returns the vertices of the Voronoi
%   edges in VX and VY so that plot(VX,VY,'-',X,Y, '.') creates the
%   Voronoi diagram.
%
%   See also DELAUNAY, TRIMESH, TRISURF, DSEARCH, CONVHULL.
%   Clay M. Thompson 7-15-95.
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 1.7 $   $Date: 1997/11/21 23:46:58 $
%   V.2.0.0 - 00-FEB-2000 - V.Lobo. Changes Voronoi do draw bondaries
%   V.2.2.0 - 00-MAY-2002 - V.Lobo. Re-orders trainges for compatibility
%
%                               with MATLAB 6

error(nargchk(3,5,nargin));   % 1 more parameter than Voronoi

if nargin==3,
%   1   more   parameter   than
Voronoi
    tri = delaunay(x,y);
    ls = '';
elseif nargin==4,
%   1 more parameter than Voronoi
    if isstr(arg3),
        tri = delaunay(x,y);
        ls = arg3;
    else
        tri = arg3;
        ls = '';
    end
else
    tri = arg3;
    ls = arg4;
end
end

```

```

% re-orient the triangles so that they are all clockwise
xt = x(tri); yt=y(tri);
ot = xt(:,1).*(yt(:,2)-yt(:,3)) + ...
      xt(:,2).*(yt(:,3)-yt(:,1)) + ...
      xt(:,3).*(yt(:,1)-yt(:,2));
bt = find(ot<0);
tri(bt,[1 2]) = tri(bt,[2 1]);
% ----- End reorientation

n = prod(size(x));
ntri = size(tri,1);
t = (1:ntri)';
T = sparse(tri,tri(:,[3 1 2]),t(:,ones(1,3))),n,n); % Triangle edge if T(i,j)
%NOTA: T is a nxn matrix, where every col/row intersection corresponds
%      to 0 if the points are adjacent in the triangulation

E = (T & T').*T; % Voronoi edge if E(i,j)

[i,j,v] = find(triu(E));
[i,j,vv] = find(triu(E'));

c1 = circle(tri(v,:),x,y);
c2 = circle(tri(vv,:),x,y);

vx = [c1(:,1) c2(:,1)].';
vy = [c1(:,2) c2(:,2)].';

% we shall now eliminate the v and vv that belong to isoclass triangles

[numlines dummy]=size(i);
indexes=1:numlines;
selection = ((class(i)~= class(j))');
selection = selection'.*indexes;
selection = find(selection);

vx = vx(:,selection);
vy = vy(:,selection);

if nargout<2
    if isempty(ls),
        co = get(gcf,'defaultaxescolororder');

```

```

    h = plot(vx,vy,'-',x,y,'.','color',co(1,:));
else
    [l,c,m,msg] = colstyle(ls); error(msg)
    if isempty(m), m = '.'; end
    h = plot(vx,vy,ls,x,y,[c m]);
end
if ~ishold,
    view(2), axis([min(x(:)) max(x(:)) min(y(:)) max(y(:))])
end
if nargin==1, vxx = h; end
else
    vxx = vx;
end

function c = circle(tri,x,y)
%CIRCLE Return center and radius for circumcircles
% C = CIRCLE(TRI,X,Y) returns a N-by-3 vector containing [xcenter(:)
% ycenter(:) radius(:)] for each triangle in TRI.

% Reference: Watson, p32.
x = x(:); y = y(:);

x1 = x(tri(:,1)); x2 = x(tri(:,2)); x3 = x(tri(:,3));
y1 = y(tri(:,1)); y2 = y(tri(:,2)); y3 = y(tri(:,3));

% Set equation for center of each circumcircle:
% [a11 a12;a21 a22]*[x;y] = [b1;b2] * 0.5;

a11 = x2-x1; a12 = y2-y1;
a21 = x3-x1; a22 = y3-y1;

b1 = a11 .* (x2+x1) + a12 .* (y2+y1);
b2 = a21 .* (x3+x1) + a22 .* (y3+y1);

% Solve the 2-by-2 equation explicitly
idet = a11.*a22 - a21.*a12;

% Add small random displacement to points that are either the same
% or on a line.
d = find(idet == 0);
if ~isempty(d), % Add small random displacement to points

```

```
delta = sqrt(eps);
x1(d) = x1(d) + delta*(rand(size(d))-0.5);
x2(d) = x2(d) + delta*(rand(size(d))-0.5);
x3(d) = x3(d) + delta*(rand(size(d))-0.5);
y1(d) = y1(d) + delta*(rand(size(d))-0.5);
y2(d) = y2(d) + delta*(rand(size(d))-0.5);
y3(d) = y3(d) + delta*(rand(size(d))-0.5);
a11 = x2-x1; a12 = y2-y1;
a21 = x3-x1; a22 = y3-y1;
b1 = a11 .* (x2+x1) + a12 .* (y2+y1);
b2 = a21 .* (x3+x1) + a22 .* (y3+y1);
idet = a11.*a22 - a21.*a12;
end

idet = 0.5 ./ idet;

xcenter = ( a22.*b1 - a12.*b2) .* idet;
ycenter = (-a21.*b1 + a11.*b2) .* idet;

radius = (x1-xcenter).^2 + (y1-ycenter).^2;

c = [xcenter ycenter radius];
```

```
function class_plot(x,y,class)
% class_plot(x,y,class)
%
% OBJECTIVE
%   Plot prototypes of different classes using different colors and
%   markers
%
% INPUT PARAMETERS
%   x          - x coordinates of the patterns
%   y          - y coordinates of the patterns
%   class      - classes of the patterns
%
% OUTPUT PARAMETERS
%
% COMMENTS
%
% V.1.0.0 - 00-MAR-2000 - V.Lobo

class_index=find(class==1);
classx = x(class_index);
classy = y(class_index);
plot(classx,classy,'+b');
hold on

class_index=find(class==2);
classx = x(class_index);
classy = y(class_index);
plot(classx,classy,'xr');

class_index=find(class==3);
classx = x(class_index);
classy = y(class_index);
plot(classx,classy,'oy');
```

```
function [ c ] = knn_mat( t_data, t_label, x )
% [ c ] = knn( t_data, t_label, x )
%
% OBJECTIVE
%   This function classifies a matrix of data patterns using a training
%   set and the 1-nn rule [Bishop 95].
%
% INPUT PARAMETERS
%   t_data      Training data matrix with one pattern per column
%   t_label     Row vector containg the labels of the t_data
%   x           Matrix with the patterns to classify (1 per column)
%
% OUTPUT PARAMETERS
%   c           Row vector with the classes of the patterns in x
%   cp         Row vector with the class probability for
%              each class
%
% COMMENTS
%   Classes are supposed to be labeled 1...N
%
% V.1.0.0 - 00-FEB-2000 - V.Lobo

% Euclidean distances from x to all t_data patterns
% 'distance' is a row vector with the discances
distance = dist( x',t_data);
[values,index]=min(distance,[],2);
c=t_label(index);
return;
```

```
function confusion=confusionMatrix( correct_class, given_class, num_classes )
% confusion=confusionMatrix( correct_class, given_class, num_classes )
%
% OBJECTIVE
%   To compute the confusion matrix (Fukunaga 1990), that shows which patterns
%   were correctly classified (in the diagonal), and between which classes the
%   errors occurred
%
% INPUT PARAMETERS
%   correct_class - vector with the correct class of each pattern
%   given_class   - vector with the class given by the classifier
%   num_classes   - number of classes present
%
% COMMENTS
%   nil
%
% V 1.0.0 - 00-000-1999 - V.Lobo
% V 1.1.0 - 22-MAY-2002 - V.Lobo

[tmp num_patterns]=size(correct_class);
confusion=zeros(num_classes);
for k=1:num_patterns
    m=correct_class(k);
    index=given_class(k)-correct_class(k);
    confusion(m,m+index)=confusion(m,m+index)+1;
    % k
end;
```

```

function outclass=selfClassify( dataset,inclass,k )
% outclass=SelfClassify( dataset,inclass,k )
%
% OBJECTIVE
%   This Function will classify every pattern in DATASET
%   using the knn rule (with K neighbors), using as training
%   set all remaining patterns of DATASET. The correct class
%   for each pattern must be passed in INCLASS, and the
%   function returns the vector OUTCLASS with the classifications
%
% INPUT PARAMETERS
%   dataset      Training data matrix with one pattern per column
%   inclass      Row vector containg the labels of "dataset"
%   k            Number of neighbours to consider
%
% OUTPUT PARAMETERS
%   outclass     Row vector with the given classes
%
% COMMENTS
%   Classes are supposed to be labeled 1...N
%
% V.1.0.0 - 00-JUN-2000 - V.Lobo

[nfeatures,npatterns]=size(dataset);

[ c,cp ] = knn( dataset(:,2:npatterns), inclass(2:npatterns), dataset(:,1),
k);
outclass(1)=c;

for a=2:npatterns-1
    xd=[dataset(:,1:a-1) dataset(:,a+1:npatterns)];
    xc=[inclass(1:a-1) inclass(a+1:npatterns)];
    [ c,cp ] = knn( xd, xc, dataset(:,a), k);
    outclass(a)=c;
    a
    disp(c);
end

[ c,cp ] = knn( dataset(:,1:npatterns-1), inclass(1:npatterns-1),
dataset(:,npatterns), k);
outclass(npatterns)=c;

```

```

function splitData=splitData(numParts,class)
% splitData=splitData(numParts,class)
%
% OBEJECTIVE:
%   Split a given dataset into small sets for cross-validation.
%
% INPUT PARAMETERS
%   numParts      Number of parts into which the data set is to be partitioned
%
%   class         A ROW vector, with the class of the pattern
%
% OUTPUT PARAMETERS
%   splitData     A matrix with "numParts" Columns, corresponding to each of the
%                 sets. The values in the matrix are the indexes into the
%                 original "data" and "class" sets.
%
%
% V. 1.0.0  V.Lobo, May 2002

numPatterns=length(class);      % Find the total number of patterns
indexes=1:numPatterns;         % Identify each pattern by it's index num.
sortNum=rand(1,numPatterns);   % Build a random sort index
classes=unique(class);         % Find which classes exist
numClasses=max(classes);       % Find how many classes exist
classNumbers=hist(class,numClasses); % find how many patterns per
class
classNumbers=floor(classNumbers/numParts); % find how many pat/class per part
patternsPerPart=sum(classNumbers); % find the total n.patt. per part
splitData=zeros(patternsPerPart,numParts); % initialize the splitData Matrix

for i=classes'
    thisClassIndexes=find(class==i); % Select the indexes that belong to C
    sortmatrix=[ sortNum(thisClassIndexes)' indexes(thisClassIndexes)'];
    sortmatrix=sortrows(sortmatrix);
    stopIndex=sum(classNumbers(1:i));
    startIndex=stopIndex-classNumbers(i)+1;
    for part=1:numParts
        fst=(part-1)*classNumbers(i)+1;
        lst=part*classNumbers(i);
        splitData(startIndex:stopIndex,part)=...

```

```
        sortmatrix(fst:lst,2);  
    end;  
end;  
  
return;
```

```
function [train,trainClass,test,testClass]=...
    buildTrainTestSet(data,class,splitData,k)
% [train,trainClass,test,testClass]=buildTrainTestSet(data,class,splitData,k)
%
% OBEJECTIVE:
%   Build a training and test set, based on the information contained in
%   the matrix splitData (produced by the routine SplitData)
%
% INPUT PARAMETERS
%   data      Matrix with one pattern per COLUMN
%   class     ROW vector with the class of each patterns
%   k         the number of the train/test set. Must be one of the rows
%             of the matrix splitData
%
%
% OUTPUT PARAMETERS
%   train
%   test
%
%
% V. 1.0.0  V.Lobo, May 2002

train=data(:,splitData(:,k));
trainClass=class(splitData(:,k));
tmpMat=removeCol(splitData,k);
[x,y]=size(tmpMat);
numElements=x*y;
tmpMat=reshape(tmpMat,[1 numElements]);
test=data(:,tmpMat);
testClass=class(tmpMat);

return
```

```
function [data,class] = read_koh(filename)
% [data,class] = read_koh(filename)
%
% OBJECTIVE
% Read an ASCII file in KOHONEN format
% This function will read an ASCII file using KOHONEN format [Kohonen 93]
%
% INPUT PARAMETERS
% filename String with the name of the file to read
%
% OUTPUT PARAMETERS
% y Matlab matrix with the data in the file
%
% COMMENTS
% Still is not universal, does not read classes, etc,etc
%
% V.1.0.0 - 00-FEB-2000 - V.Lobo

fin=fopen(filename,'r');

                                % read first line
x = fgetl(fin);
num_colunas = sscanf(x,'%g',1);
data = [];
class = [];

while ~feof(fin)
    x = fgetl(fin);
    if x(1)=='#'
        break;
    end;
    [ y, count, errmsg, nextindex ] = sscanf(x,'%g',num_colunas);
    if count==num_colunas
        data = [data ; y'];
        x=x(nextindex:end);
        [label,count,errmsg,nextindex ] = sscanf(x,'%g',1);
        if count==1
            class = [class ; label ];
        else
            class = [class ; 0 ];
        end;
    end;
end;
```

```
end;
```

```
end;
```

```
fclose(fin);
```

```
function write_koh(filename,data,class)
%   write_koh(filename,data,class)
%
% OBJECTIVE
%   Write an ASCII file in KOHONEN format
%   This function will write an ASCII file using KOHONEN format [Kohonen 93]
%
% INPUT PARAMETERS
%   filename  String with the name of the file to write
%   data      Matrix with one pattern per row
%   class     Class (label) given to each pattern
%
% OUTPUT PARAMETERS
%   nil
%
% COMMENTS
%
% V.1.0.0 - 00-FEB-2000 - V.Lobo

[num_patt,num_features]=size(data);

fout=fopen(filename,'w');
fprintf(fout,'%d\n',num_features);
for i=1:num_patt
    fprintf(fout,'%f ',data(i,:));
    fprintf(fout,'%d\n',class(i));
end;
fclose(fout);
```

```
function data=remove_col(data,index)
% data=remove_col(data,index)
%
% OBJECTIVE
%   This function will remove a single column from a matrix
%
% INPUT PARAMETERS
%   data          - Matrix from where the column is removed
%   index         - Index of the column to remove
%
% OUTPUT PARAMETERS
%   data          - Output matrix
%                  each class
% COMMENTS
%
% V.1.0.0 - 00-MAR-2000 - V.Lobo

[num_row,num_col]=size(data);
if index==1
    data=data(:,2:end);
else
    if index==num_col
        data=data(:,1:index-1);
    else
        data=[data(:,1:index-1) data(:,index+1:end)];
    end;
end;
return;
```

```
function data=generate_2D_uniform_data(x1,x2,y1,y2,n)
% data=generate_2D_uniform_data(x1,x2,y1,y2,n)
%
% OBJECTIVE
%   Generate 2-dimensional data with a uniform distribution
%   in a specified area
%
% INPUT PARAMETERS
%   x1,x2,y1,y2   - x and y coordinates of the rectangle where the
%                   data is to be generated
%   n             - number of data points
%
% OUTPUT PARAMETERS
%   data          - Output matrix, with one point per column
%
% COMMENTS
%
% V.1.0.0 - 00-MAR-2000 - V.Lobo

data=rand(2,n);
data(1,:)=data(1,:)*(x2-x1)+x1;
data(2,:)=data(2,:)*(y2-y1)+y1;
```

```

function [data,class]=generate_double_f_validation(totalpoints)
% [data,class]=generate_double_f_validation(totalpoints)
%
% OBJECTIVE
%   Generate 2-dimensional data for Hart's double F problem
%
% INPUT PARAMETERS
%   totalpoints - Total number of data patterns. Thus must be an
%                 even number, and preferably dividable by 12
%
% OUTPUT PARAMETERS
%   data        - Data matrix, with one pattern per column
%   class       - Class of the patterns contain in "data"
%
% COMMENTS
%
% V.1.0.0 - 00-MAR-2000 - V.Lobo%

error(nargchk(1,1,nargin));

nsmall = floor(totalpoints/12);
nlarge = floor(totalpoints/6);
nlast = (totalpoints/2) - 2*nsmall-nlarge;

a1=generate_2D_uniform_data(0,7.5,0,5,nsmall);
a2=generate_2D_uniform_data(0,15,5,10,nlarge);
a3=generate_2D_uniform_data(0,7.5,10,15,nsmall);
a4=generate_2D_uniform_data(0,15,15,20,nlast);

b1=generate_2D_uniform_data(7.5,22.5,0,5,nlarge);
b2=generate_2D_uniform_data(15,22.5,5,10,nsmall);
b3=generate_2D_uniform_data(7.5,22.5,10,15,nlast);
b4=generate_2D_uniform_data(15,22.5,15,20,nsmall);

data=[a1 a2 a3 a4 b1 b2 b3 b4];
class = [ones(1,totalpoints/2) 2*ones(1,totalpoints/2)];

return

```

```
function [data,class]=generate_straight(totalpoints)
% [data,class]=generate_straight(totalpoints)
%
% OBJECTIVE
%   Generate 2-dimensional data for the straight line problem
%
% INPUT PARAMETERS
%   totalpoints - Total number of data patterns. Thus must be an
%               even number.
%
% OUTPUT PARAMETERS
%   data        - Data matrix, with one pattern per column
%   class       - Class of the patterns contain in "data"
%
% COMMENTS
%
% V.1.1.0.0 - 00-AUG-2002 - V.Lobo

error(nargchk(1,1,nargin));

nsmall = floor(totalpoints/12);
nlarge = floor(totalpoints/6);
nlast  = (totalpoints/2) - 2*nsmall-nlarge;

a1=generate_2D_uniform_data(0,0.5,0,1,totalpoints/2);

b1=generate_2D_uniform_data(0.5,1,0,1,totalpoints/2);

data=[a1 b1 ];
class = [ones(1,totalpoints/2) 2*ones(1,totalpoints/2)];

return
```

```

function spec=spectra_wavfile(wavname,npontos,num_average,pontosUteis,useLog)
% spec=spectra_wavfile(wavname,npontos,num_average,pontosUteis,log)
%
% OBJECTIVE
% Calculate the spectra of a signal contained a WAV file
%
% INPUT PARAMETERS
%
% wavname      - Ascii text with the name of the wav file to open. The full
%                name must be given, including the .wav extension.
% npontos      - Number of points that will be used to calculate each spectra
% num_average  - Number of raw spectra that will be averaged to produce a
final
%                spectrum. There will be a 50% overlap between them.
% pontosUteis - Number of points of the spectra that will be output. Only
the
%                first "pontosUteis" will be used, and the higher
frequencies
%                will be ignored. This is used to compensate for bad analog
%                filtering.
% useLog       - LOGARITHM, set to "0" for linear amplitude, or "1" to
produce a spectra
%                where the values are equal to LOG(amplitude+1)
%
% OUTPUT PARAMETERS
%
% spec         - Spectra of the signal. The number of columns will be equal
to
%                "pontosUteis", and the number of rows will be as many as
can
%                be extracted from the file.
%
% COMMENTS
% The power is normalized to the number of bins (so that the average power
per bin is 1
% The spectra will always be positive, since 1 is added when calculating
LOGARITHMS
%
% USES
% mHamming
%
```

```
% V 1.0.0 - 00-000-2000 - V.Lobo
% V 1.0.0 - 00-MAY-2002 - V.Lobo

npontos2=npontos/2;
nHamming=mHamming(npontos);

wavsize=wavread(wavname,'size');
ntotal=wavsize(1); % Total number of samples in file

num_spectra=floor(ntotal/npontos2)-1;
num_patterns=floor(num_spectra/num_average);
spec=zeros(num_patterns,pontosUteis);
tmp_spec=zeros(num_average,pontosUteis);
iter=1;
for iter_pattern=1:num_patterns
    for i=1:num_average
        tmpt=wavread(wavname,[ (iter-1)*npontos2+1 (iter+1)*npontos2 ]');
        tmpf=abs(fft((tmpt-mean(tmpt)).*nHamming));
        tmpf=tmpf(1:pontosUteis); % Extract only the first points
        tmpf=tmpf./mean(tmpf); % Normalize power to n.of bins
        if useLog
            tmpf=log(tmpf+1); % Calculate the LOG (+1 to avoid neg)
        end;
        tmp_spec(i,:)=tmpf;
        iter=iter+1;
    end;
    spec(iter_pattern,:)=mean(tmp_spec);
end;
```

```
function window=mHamming(N)
% window=hamming(N)
%
% OBJECTIVE
%   Calculate Hamming's window
%
% INPUT PARAMETERS
%   N   - Number of points (must be even)
%
% OUTPUT PARAMETERS
%   window - Hamming's window
%
% COMMENTS
%   This version implements the correct Hamming window
%
% V.1.0.0 - 25-APR-1996 - P.Monica de Oliveira

window=0.54-0.46*cos(2*pi*(0:N-1)/N);
```

```
function hausdorff=hausdorff( A, B )
% hausdorff=hausdorff( A, B );
%
% OBJECTIVE
% Calculate the Hausdorff distance between two sets of points
%
% INPUT PARAMETERS
% A,B          - Vectors with data
%
% OUTPUT PARAMETERS
% hausdorff    - The Hausdorff distance
%
% COMMENTS
%
% V.1.0.0 - 00-AUG-1998 - V.Lobo

max_i = max( size(A));          % Find the dimension of the vectors
dist_vectorA = zeros(1,max_i);  % Initialize parital distance vector
dist_vectorB = zeros(1,max_i);  % Initialize parital distance vector

min_distA = zeros(1,max_i);     % Initialize the minimum dist.vector
min_distB = zeros(1,max_i);     % Initialize the minimum dist.vector

for i=1 : max_i
    for j=1 : max_i
        dist_vectorA(j) = sqrt( (A(i)-B(j)).^2 + (i-j).^2 ) ;
        dist_vectorB(j) = sqrt( (B(i)-A(j)).^2 + (i-j).^2 ) ;
    end;
    min_distA(i) = min (dist_vectorA );
    min_distB(i) = min (dist_vectorB );
end;

hausdorff = max( max(min_distA), max(min_distB) );
```

```

function primes=findPrimes(inmat)
% primes=findPrimes(inmat) - Finds Prime implicants
%
% OBJECTIVE:
%   Given a number of minterms of a Boolean function, this routine
%   calculates all prime implicants of that function
%
% INPUT PARAMETERS:
%   inmat      Matrix with one minterm per line. Each row corresponds
%              to a different variable, which must have the value 1 or 0
%
% OUTPUT PARAMETERS:
%   outmat     Matrix with one implicant per line. Variables that
%              are not contained in the implicant (that are Don't
%              cares) are represented by NAN (not-a-number).
%              Thus, this matrix has the same number of columns as inmat
%
% USES:
%   SolveImplicant, Dual3Single, DistHamming, & common Matlab routines.
%
% V.1.0.0 - 00-APR-2002 - V.Lobo

finish=0;
inmatc=ones(size(inmat));

primes=[];
primesc=[];

while not(finish)
    [o, oc, n, nc]=SolveImplicant(inmat,inmatc);
    primes = [ primes ; o ];
    primesc= [ primesc ; oc ];
    if n==[]
        finish=1;
    else
        inmat=n;
        inmatc=nc;
    end;
end;

primes=Dual2Single( primes, primesc );

```

```
return

function [outmat,outmatc,nextmat, nextmatc]=SolveImplicant(inmat,inmatc)
% [outmat,outmatc,nextmat, nextmatc]=SolveImplicant(inmat,inmatc)
%
% OBJECTIVE
%   Given a matrix with implicants, try to form larger implicants, and return
%   those implicants, together with another matrix with the implicants that
%   cannot be simplified. This implements the QUINE-McCLUSKY method of
%   simplification
%
% INPUT PARAMETERS
%   inmat   matrix where each row is an implicant. Each column corresponds
%           to a different boolean variable. If a variable is not specified,
%           (i.e. if it is a don't care), that column should contain a 0,
%           and the corresponding element of the companion matrix inmatc
%           should
%           also have a 0.
%   inmatc  companion matrix to inmat, that contains 1 for each column where
%           the value of inmat "counts" ( it is CARE as opposed do a don't
%           care)
%
% OUTPUT PARAMETERS
%   outmat  matrix with the implicants that cannot be solved further
%   outmatc companion matrix to outmat, indicating which variables are to
%           be considered (i.e., are NOT don't cares)
%   nextmat matrix with the simplified implicants, that may be passed again
%           to this routine to attempt further simplification
%
% by V.Lobo, April 2002, SDSU

[implicants,x]=size(inmat);

nextmat=[];
nextmatc=[];
prime=ones(1,implicants);

for i=1:implicants-1
    for j=i+1:implicants
        if inmatc(i,:)==inmatc(j,:);
```

```

    %if i and j have the same don't care
    d=DistHamming(inmat(i,:),inmat(j,:));
    if d==1      % now we'll build a new implicant
        newImplicant=inmat(i,:);
        newImplicantc=inmatc(i,:);
        toRemove=find(xor(inmat(i,:),inmat(j,:)));
        newImplicant(toRemove)=0;
        newImplicantc(toRemove)=0;
        nextmat= [nextmat ; newImplicant];
        nextmatc= [nextmatc ; newImplicantc];
        prime(j)=0;
        prime(i)=0;
    end;
end;
end;
end;
[x,i,j]=unique([nextmat nextmatc],'rows');
nextmat=nextmat(i,:);
nextmatc=nextmatc(i,:);
outmat=inmat(find(prime),:);
outmatc=inmatc(find(prime),:);

return;

function outmat=Dual2Single(inmat,inmatc)
% outmat=Dual2Single(inmat,inmatc)
%
% OBJECTIVE
% Convert between the two formats used to represent implicants
% of Boolean functions. In both formats, each implicant is
% represented by a row, and each column corresponds to a different
% variable. In the SINGLE format, each element is a 1, a 0 or a NAN
% (matlab's Not-A-Number) to represent don't care (an indetermined
% value). In the double format, the matrix representing the implicant
% has only 1 and 0, and when the value is don't care it uses also a
% 0. To distinguish between these two values, another, so called
% COMPANION MATRIX (whose name ends with a C) is used that has 1
% when the corresponding value in the implicant matrix is to be taken
% as 1 or 0, and 0 when it is a "Don't care".
%
% INPUT PARAMETERS

```

```
% inmat matrix where each row is an implicant. Each column corresponds
%       to a different boolean variable. If a variable is not specified,
%       (i.e. if it is a don't care), that column should contain a 0,
%       and the corresponding element of the companion matrix inmatc
should
%       also have a 0.
% inmatc companion matrix to inmat, that contains 1 for each column where
%       the value of inmat "counts" ( it is CARE as opposed do a don't
care)
%
% OUTPUT PARAMETERS
% outmat Matrix with one implicant per line.Variables that
%       are not contained in the implicant (that are Don't
%       cares) are represented by NAN (not-a-number).
%       Thus, this matrix has the same number of columns as inmat
%
% by V.Lobo, April 2002, SDSU

outmat=inmat;
outmat(find(inmatc==0))=nan;
```


References

- Aamodt, A. and E. Plaza (1994). "Case-based reasoning; Foundational issues, methodological variations, and system approaches." AI Communications **7**(1): 39-59.
- Aha, D. W. (1991). Case-Based Learning Algorithms. DARPA Case Based Reasoning WorkShop, Washington DC, Morgan Kaufman.
- Aha, D. W., D. Kibler and M. K. Albert (1991). "Instance-based learning algorithms." Machine Learning **6**: 37-66.
- Aha, D. W. (1995). An implementation and experiment with the nested generalized exemplars algorithm, Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence.
- Aha, D. W. (1997). Lazy Learning, Kluwer Academic Publishers.
- Alexandre, L. A., A. C. Campilho and M. Kamel (2001). "On combining classifiers using sum and product rules." Pattern Recognition Letters **22**(12): 1283-1289.
- Alhoniemi, E., J. Hollmén, O. Simula and J. Vesanto (1999). "Process Monitoring and Modeling using the Self-Organizing Map." Integrated Computer Aided Engineering **6**(1): 3-14.
- Almas (1998). Relatório do projecto de Estatística. Portugal, Escola Naval.
- Alves, A. (1997). WPVM Manual (version 2.0), Universidade de Coimbra.
- Andrew, R. K., B. M. Howe, J. A. Mercer and M. A. Dzieciuc (2001). "Ocean ambient sound: Comparing the 1960s with the 1990s for a receiver off the California coast." Acoustics Research Letters Online **3**(2): 65-70.
- Anthony, M. and P. L. Bartlett (1999). Learning in Neural Networks: Theoretical Foundations, Cambridge University Press.
- Anzai, Y. (1992). Pattern recognition and machine learning, Academic Press Inc.
- Apel, J. R. (1990). Principals of Ocean Physics, Academic Press.
- Atlas, K., L. Owsley, J. McLaughlin and G. Bernard (1996). Automatic feature-sindinf for time-frequency distributions. IEEE-SP : International Conference on Time-Frequency and Time-Scale Analysis, Newark, N.J., USA, Gordon & Breach.
- Baase, S. and A. V. Gelder (2000). Computer Algorithms, Addison-Wesley.
- Bandeira, N. (1996). Projecto Final de Curso. Departamento de Informatica. Lisbon, Universidade Nova de Lisboa.
- Baram, Y. (2000). "A geometric approach to consistent classification." Pattern Recognition **33**: 177-184.
- Barreto, G. and A. Araújo (1999). Unsupervised Context-based Learning of Multiple Temporal Sequences. IEEE-INNS Intl. Joint Conf. on Neural Networks (IJCNN'99), Washington, DC, USA.
- Barth, P. (1995). A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization, Max Planck Institute for Computer Science: 13.
- Bax, E. (2000). "Validation of Nearest Neighbor Classifiers." IEEE Transactions on Information Theory **46**(7): 2746-2752.
- Behme, H., W. D. Brandt and H. W. Strube (1993). Speech Recognition by Hierarchical Segment Classification. ICANN 93, Springer.
- Bendat, J. S. and A. G. Piersol (1993). Engineering Applications of Correlations and Spectral Analysis, John Wiley & Sons.

- Bergsten, U., J. Schubert and P. Svensson (1997). Applying Data Mining and Machine Learning Techniques to Submarine Intelligence Analysis. KDD'97 - Third International Conference on Knowledge Discovery and Data Mining, Newport Beach, USA, AAAI Press.
- Bezdek, J. C., T. R. Reichherzer, G. S. Lim and Y. Attikiouzel (1998). "Multiple-prototype classifier design." IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **28**(1): 67 - 79.
- Bezdek, J. C., J. Keller, R. Krishnapuram and M. Pal (1999). Fuzzy models and algorithms for pattern recognition and image processing, Kluwer Academic Publishers.
- Bezdek, J. C., J. M. Keller, R. Krishnapuram, L. I. Kuncheva and N. R. Pal (1999). "Will the real iris data please stand up?" IEEE Transactions on Fuzzy Systems **7**(3): 368 - 369.
- Bezdek, J. C. and L. I. Kuncheva (2001). "Nearest Prototype Classifier Designs: An experimental study." International Journal of Intelligent Systems **16**: 1445-1473.
- Bishop, C. M. (1995). Neural Networks for Pattern Recognition, Oxford University Press.
- Bishop, C. M., M. Svensén and C. K. I. Williams (1996). GTM: A Principled Alternative to the Self-organizing Map.
- Bishop, C. M., M. Svensen and C. K. I. Williams (1998). "GTM: The Generative Topographic Mapping." Neural Computation **10**(1): 215-234.
- Blanc-Benon, P. and G. Bienvenu (1995). Passive target motion analysis using multipath differential time-delay and differential Doppler shifts. ICASSP-95.
- Bogert, B. P., M. J. R. Healy and J. W. Tukey (1963). The Quefrequency Analysis of Time Series for Echoes: Cepstrum, Pseudo-Auto Covariances, Cross-Cepstrum, and Saphe Cracking. Symposium on Time Series Analysis.
- Boser, B. E., I. M. Guyon and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. Workshop on Computational Learning Theory, 5th Annual, Pittsburgh, Pennsylvania, United States, ACM Press.
- Breiman, L., J. H. Friedman, R. A. Olsen and C. J. Stone (1984). Classification and Regression Trees, Chapman & Hall.
- Brighton, H. and C. Mellish (2002). "Advances in Instance Selection for Instance-Based Learning Algorithms." Data Mining and Knowledge Discovery **6**: 153-172.
- Broadhead, M. K., L. A. Pflug and R. L. Field (1996). Minimum entropy filtering for improving nonstationary sonar signal classification. IEEE Signal Processing Workshop on Statistical Signal and Array Processing.
- Broomhead, D. S. and D. Lowe (1988). "Multivariable functional interpolation and adaptive networks." Complex Systems **2**: 321-355.
- Bruckner, B., M. Franz and A. Richter (1992). A modified Hypermap Architecture for Classification of Biological Signals. Artificial Neural Networks 2. I. Aleksander and J. Taylor. Amsterdam, Netherlands: 1167-1170.
- Bryant, R. (1986). "Graph-Based Algorithms for Boolean Function Manipulation." IEEE Transactions on Computers **C-35**(8): 677-691.
- Burdic, W. S. (1991). Underwater acoustic system analysis. Prentice Hall signal processing series. Englewood Cliffs, N.J. :, Prentice Hall; xiii, 466 p. : ill. ; 25 cm.
- Burges, C. J. C. (1998). "A Tutorial on Support Vector Machines for Pattern Recognition." Data Mining and Knowledge Discovery **2**(2): 121-167.
- Burton, D. (1991). Acoustic transient classification of passive sonar signals by using vector quantization. ICASSP-91, Toronto, Canada.
- Carpintei, O. A. (1998). A Hierarchical Self-Organizing Map Model for Sequence Recognition. ICANN98, the 8th International Conference on Artificial Neural Networks, Springer.
- Carpintei, O. A. S. (1998). A Self-Organizing Map Model for Analysis of Musical Time Series. Vth Brazilian Symposium on Neural Networks, Belo Horizonte, Brasil.
- Carpintei, O. A. S., A. P. A. Silva and C. H. L. Feichas (2000). A hierarchical neural model in short-term load forecasting. IJCNN00, Como, Italy, IEEE Press.
- Casselmann, F. L., D. F. Freeman, D. A. Kerrigan, S. E. Lane, N. H. Millstrom and W. G. Nichols, Jr. (1991). A neural network-based passive sonar detection and classification design with a low false alarm rate. IEEE Conference on Neural Networks for Ocean Engineering, 1991.
- Cerverón, V. and A. Fuertes (1998). Parallel Random Search and Tabu Search for the Minimal Consistent Subset Selection Problem. Random 98.
- Cerverón, V. and F. J. Ferri (2001). "Another Move Toward the Minimum Consistent Subset: A Tabu Search Approach to the Condensed Nearest Neighbor Rule." IEEE Transactions on Systems, Man, and Cybernetics -Part B: Cybernetics **31**(3): 408-412.
- Chandrasekaran, V. and M. Palaniswami (1995). "Spatio-temporal Feature Maps using Gated Neuronal Architecture." IEEE Transactions on Neural Networks **6**(5): 1119-1131.
- Chandrasekaran, V. and Z.-Q. Liu (1998). "Topology Constraint Free Fuzzy Gated Neural Networks for Pattern Recognition." IEEE TRANSACTIONS ON NEURAL NETWORKS **9**(3): 483-502.

- Chang, C.-L. (1974). "Finding Prototypes for nearest neighbor Classifiers." IEEE Transactions on Computers **23**(11): 1179-184.
- Chang, E. I. and R. P. Lippmann (1991). Using genetic algorithms to improve pattern classification performance. Advances in Neural Information Processing Systems. R. P. Lippman, J. E. Moody and S. Touretzky. San Mateo, CA, Morgan Kaufman. **3**: 797-803.
- Chang, K. and J. Ghosh (2001). "A unified model for probabilistic principal surfaces." IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(1): 22-41.
- Chappelier, J. and A. Grumbach (1995). A Kohonen Map for Temporal Sequences. NEURAP'95, Marseilles, France.
- Chappell, G. J. and J. G. Taylor (1993). "The Temporal Kohonen Map." Neural Networks **6**: 441-445.
- Child, D. (1990). The Essentials of Factor Analysis. Cassel.
- Choi, S.-H. and P. Rockett (2002). "The training of Neural Classifiers with Condensed Datasets." IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **32**(2): 202-206.
- CIE (1986). Colorimetry, CIE Publication 15.2 (1986), International Commission on Illumination, CIE.
- Cios, K. J., W. Pedrycz and R. Swiniarski (1998). Data Mining Methods for Knowledge Discovery, Kluwer.
- Coates, R. F. W. (1990). Underwater acoustic systems. Basingstoke, Hampshire :, Macmillan,.
- Collier, R. D. (1998). Ship and platform noise, propeller noise. Handbook of acoustics. M. J. Crocker, John Wiley & Sons.
- Cook, S. A. (1971). The complexity of theorem proving procedures. Annual ACM Symposium on Theory of Computing.
- Cook, S. A. (1983). "An overview of computational complexity." Communications of the ACM **26**(6): 400-408.
- Cooley, J. W. and J. W. Tukey (1965). "An algorithm for the machine calculation of the complex Fourier series." Mathematics of Computation **19**: 297-301.
- Cottrell, M., J. C. Fort and G. Pages (1998). "Theoretical Aspects of the SOM algorithm." Neurocomputing, Elsevier **21**: 119-138.
- Coultrip, R. (1998). "A CMOS binary pattern classifier based on Parzen's Method." IEEE Transactions on Neural Networks **9**(1): 2-10.
- Council, U. S. N. R. (1994). Low-frequency sound and marine mammals : current knowledge and research needs. Washington, D.C. :, National Academy Press,.
- Cover, T. M. (1965). "Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition." IEEE Transactions on Electronic Computers **14**: 326-334.
- Cover, T. M. and P. E. Hart (1967). "Nearest Neighbor Pattern Classification." IEEE Transactions on Information Theory **13**(1): 21-27.
- Cristianini, N. and J. Shawe-Taylor (2001). An introduction to Support Vector Machines, Cambridge University Press.
- Critchley, D. A. (1994). Extending the Kohonen Self-Organizing Maps by use of adaptive parameters and temporal neurons. London, University College London.
- Crocker, M. J. (1998). Handbook of acoustics, John Wiley.
- Damon, R. A. (1987). Experimental design, ANOVA, and regression, Harper & Row.
- DARPA (1988). DARPA Neural Network Study, AFCEA International Press.
- Dasarathy, B. V. and L. J. White (1978). "A characterization of nearest neighbor rule decision surfaces and a new approach to generate them." Pattern Recognition **10**: 41-46.
- Dasarathy, B. V. (1991). Nearest Neighbor Pattern Classification techniques, IEEE Computer Society Press.
- Dasarathy, B. V. (1994). "Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design." IEEE Transactions on Systems, Man, and Cybernetics **24**(3): 511-517.
- Dasarathy, B. V. and J. S. Sanchez (2000). Tandem fusion of nearest neighbor editing and condensing algorithms - data dimensionality effects. 15th International Conference on Pattern Recognition, 2000, Barcelona, Spain.
- Daubechies, I. (1990). "The Wavelet Transform, Time-Frequency Localization and Signal Analysis." IEEE Transactions on Information Theory **36**(5): 961-1005.
- Davey, N., S. P. Hunt and R. J. Frank** (1999). "Time Series Prediction and Neural Networks." Proc. 5th International Conference on Engineering Applications of Neural Networks (EANN'99): 93-98.
- Davis, M. and H. Putnam (1960). "A computing procedure for Quantification Theory." Journal of the Association for Computing Machinery **7**: 201-215.
- De, R. K. and S. K. Pal (2001). "A connectionist model for selection of cases." Information Sciences (132).
- Decaestecker, C. (1997). "Finding prototypes for nearest neighbor classification by means of gradient descent and deterministic annealing." Pattern Recognition **30**(2): 281-288.
- Demirekler, M. and H. Altincay (2002). "Plurality voting-based multiple classifier systems: statistically independent with respect to dependent classifier sets." Pattern Recognition **35**(11): 2365-2379.
- Dempster, A. P., N. M. Laird and D. B. Rubin (1977). "Maximum Likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society **39**(1): 1-38.

- Devi, V. S. and M. N. Murty (2002). "An incremental prototype set building technique." Pattern Recognition **53**: 505-513.
- Devijver, P. A. and J. Kittler (1982). Pattern Recognition - A Statistical Approach, Prentice-Hall.
- Devroye, L. and T. J. Wagner (1979). "Distribution-free inequalities for the deleted and holdout error estimates." IEEE Transactions on Information Theory **25**: 202-207.
- Devroye, L., L. Györfi and G. Lugosi (1996). A probabilistic theory of pattern recognition. New York :, Springer,.
- Dietterich, T. G., D. Wettschereck, C. G. Atkeson and A. W. Moore (1994). Memory-Based Methods for Regression and Classification. Advances in Neural Information Processing Systems.
- Dietterich, T. G. (1998). "Approximate Statistical Test For Comparing Supervised Classification Learning Algorithms." Neural Computation **10**(7): 1895-1923.
- Domingos, P. (1995). The RISE 2.0 System: A Case Study in Multistrategy Learning, University of California at Irvine.
- Domingos, P. (1996). "Unifying Instance-Based and Rule-Based Induction." Machine Learning **24**: 141-168.
- Domingos, P. (1997). A Unified Approach to Concept Learning. Department of Information and Computer Science, University of California at Irvine.
- Domingos, P. (1999). "The Role of Occam's Razor in Knowledge Discovery." Data Mining and Knowledge Discovery **3**(4).
- Drakopoulos, J. A. (1995). Bounds on the Classification Error of the Nearest Neighbor Rule. ICML 95.
- Drobnics, M., U. Bodenhofer and W. Winiwarter (2000). Interpreting Self-Organizing Maps with Fuzzy Rules. IEEE Conference on Tools with Artificial Intelligence ICTAI'00, Vancouver, IEEE Press.
- Duda, R. O., P. E. Hart and D. G. Stork (2001). Pattern Classification, Wiley-Interscience.
- Dwyer, R. F. (1996). Advances in sonar signal processing in the 90's. OCEANS '96. MTS/IEEE.
- Dyer, I. (1998). Ocean Ambient Noise. Handbook of Acoustics. M. J. Crocker, John Wiley & Sons.
- Ebbeson, G. R., J. M. Ozard, P. Wort, G. Litchfield and C. Vigneron (1997). An environmental database for matched-field processing. OCEANS '97. MTS/IEEE.
- Efron, B. (1979). "Bootstrap methods: Another look at the jackknife." Annals of statistics **7**(1): 1-26.
- Efron, B. (1986). "Discussion: Jackknife, Bootstrap and other resampling methods in regression analysis."
- Efron, B. (1990). "More efficient bootstrap computations." Journal of the American Statistical Association **85**: 79-89.
- Eiter, T. and G. Gottlob (1995). "Identifying the minimal transversals of a hypergraph and related problems." SIAM Journal of Computing **24**(6): 1278-1304.
- Emam, K. E., S. Benlarbi, N. Goel and S. N. Rai (2001). "Comparing case-based reasoning classifiers for predicting high risk software components." Journal of Systems and Software **55**(3): 301-320.
- Erwin, E., K. Obermeyer and K. Schulten (1991). Convergence properties of self-organizing maps. Artificial Neural Networks. T. Kohonen, K. Mäkiäara, O. Simula and J. Kangas, Elsevier: 409-414.
- Essex, C. and N. M.A.H. (1990). "Fractal dimension: Limit capacity of Hausdorff dimension." American Journal of Physics **58**(10): 986-988.
- Euliano, N. and J. Principe (1996). Spatio-temporal self-organizing feature maps. ICNN '96, Washington.
- Euliano, N., J. Principe and P. Kulzer (1996). A Self-Organizing Temporal Pattern Recognizer with Application to Robot Landmark Recognition. SpatioTemporal Models in Biological and Artificial Systems.
- Euliano, N. and J. Principe (1998). Temporal plasticity in self-organizing networks. IJCNN '98, Alaska, USA.
- Euliano, N. and J. Principe (1999). A Spatio-Temporal Memory Based on SOMs with Activity Diffusion. Kohonen Maps. E. Oja and S. Kaski. Amsterdam, Elsevier: 253-266.
- Everitt, B., S. Landau and M. Leese (2001). Cluster analysis. London : New York :, Arnold ; Oxford University Press,.
- Fasulo, D. (1999). An analysis of recent work on clustering algorithms, University of Washington, Seattle, Department of Computer Science & Engineering.
- Ferri, F. J., J. V. Albert and E. Vidal (1999). "Considerations About Sample-Size Sensitivity of a Family of Edited Nearest-Neighbor Rules." IEEE Transactions on Systems, Man, and Cybernetics **29**(4): 667-672.
- Fisher, R. A. (1936). "The use of Multiple Measurements in Taxonomic Problems." Annals of Eugenics **VII**(II): 179-188.
- Fix, E. and J. L. Hodges (1951). Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties. Randolph Field, USAF School of Aviation Medicine - Project 21-49-004: 261-272.
- Flury, B. (1988). Common principal components and related multivariate models. New York :, Wiley,.
- Fogel, D. B. (1999). Evolutionary Computation : Towards a New Philosophy of Machine Intelligence, IEEE Press.
- Fritzke, B. (1991). Let it Grow - Self-organizing Feature Maps With Problem Dependent Cell Structure. ICANN-91, Helsinki, Elsevier Science Publ.
- Fritzke, B. (1995). "Growing grid { a self-organizing network with constant neighborhood range and adaptation strength." Neural Processing Letters **2**(5): 9-13.

- Fritzke, B. (1995). A growing neural gas network learns topologies. Advances in Neural Information Processing Systems. G. Tesauro, D. S. Touretzky and T. K. Leen. Cambridge MA, MIT Press. 7: 625-632.
- Fu, L. (1994). Neural Networks in Computer Intelligence, McGraw Hill.
- Fujii, T. (1995). Neural networks for ocean engineering. IEEE International Conference on Neural Networks, 1995.
- Fukunaga, K. and D. R. Olsen (1971). "An Algorithm for Finding Intrinsic Dimensional of Data." IEEE Transactions on Computers **c-20**(2): 176-183.
- Fukunaga, K. and R. R. Hayes (1989). "Estimation of classifier performance." IEEE Transactions on Pattern Analysis and Machine Intelligence **11**(10): 1087 -1101.
- Fukunaga, K. and R. R. Hayes (1989). "The reduced Parzen classifier." IEEE Transactions on Pattern Analysis and Machine Intelligence **11**(4): 423 -425.
- Fukunaga, K. and R. R. Hayes (1989). "Effects of sample size in classifier design." IEEE Transactions on Pattern Analysis and Machine Intelligence **11**(8): 873 -885.
- Fukunaga, K. (1990). Introduction to statistical patterns recognition, Academic Press Inc.
- Gabor, D. (1946). "Theory of communication."
- Gama, J. (2000). "Combining Classification Algorithms." AI Communications **1**(2).
- Gates, G. W. (1972). "The Reduced Nearest Neighbor Rule." IEEE Transactions on Information Theory: 431-433.
- Gawrys, M. and J. Sienkiewicz (1993). Roush Sets Library User's Manual ver 2.0. Warsaw, ICS - Warsaw University of Technology.
- Gawrys, M. and J. Sienkiewicz (1994). RSL - The Rough Set Library, version 2.0, ICS - Warsaw University of Technology.
- Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam (1994). PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press.
- Gentile, C. and M. Sznaiar (2001). "An Improved Voronoi-Diagram-Based Neural Net for Pattern Classification." IEEE Transactions on Neural Networks **12**(5): 1227-1234.
- Gerstoft, P. (1994). "Inversion of seismoacoustic data using genetic algorithms and a posteriori probability distributions." Journal of the Acoustical Society of America **95**(2): 770-782.
- Geva, S. and J. Sitte (1991). "Adaptive nearest neighbor pattern classifier." IEEE Transactions on Neural Networks **2**(2): 318-322.
- Ghosh, J., S. Chakravarthy, Y. Shin, C.-C. Chu, L. Deuser, S. Beck, R. Still and J. Whiteley (1991). Adaptive kernel classifiers for short-duration oceanic signals. IEEE Conference on Neural Networks for Ocean Engineering, 1991.
- Giellis, G. R. (1983). A technique of comparative analysis of underwater sound transmission loss curves. NRL report : Washington, D.C. :, Naval Research Laboratory,. **8711**: iii, 24 p. : ill. ; 28 cm.
- Gioiello, M., G. Vassallo and F. Sorbello (1992). A New Fully Digital Neural Network Hardware Architecture for Binary Valued Pattern Recognition. International Conference on Signal Processing Applications and Technology, Boston.
- Girolami, M. (2001). "The topographic organization and visualization of binary data using multivariate-Bernoulli latent variable models." IEEE Transactions on Neural Networks **16**(6): 1367 -1374.
- Glover, F. and M. Laguna (1997). Tabu search, Kluwer Academic Publishers.
- Goodman, D. and T. Yamamoto (1988). "Directional spectral and particle motion measurements of underwater seismic noise." Journal of the Acoustical Society of America **84**(S1): S175.
- Gowda, K. C. and G. Krishna (1979). "The Condensed Nearest Neighbor Rule Using the Concept of Mutual Nearest Neighborhood." IEEE Transactions on Information Theory **25**(4): 488-490.
- Gowda, K. C. and T. V. Ravi (1994). A Modified Condensed Nearest Neighbor Rule using the Symbolic Approach. 2nd Australian and New Zealand Conference on Intelligent Information Systems.
- Green, M. D. (1997). Shallow water acoustic communications system (SWACS).
- Group, S. A. (1988). Physics of sound in the sea. Los Altos, Calif. :, Peninsula Pub.,.
- Guimarães, G. and W. Urfer (2000). Self-Organizing Maps and its Applications in Sleep Apnea Research and Molecular Genetics, University of Dortmund - Statistics Department.
- Guimarães, G., J. H. Peter and A. Ultsch (2001). "A method for automated temporal knowledge acquisition applied to sleep-related breathing disorders." Artificial Intelligence in Medicine **23**: 211-237.
- Gyorfi, L. (1978). "On the rate of convergence of nearest neighbor classification." IEEE Transactions on Information Theory **21**: 552-557.
- Halls, P. J., M. Bulling, P. C. L. White, L. Garland and S. Harris (2001). "Dirichlet neighbours: revisiting Dirichlet tessellation for neighbourhood analysis." Computers, Environment and Urban Systems **25**(1): 105-117.
- Harmelen, V. (1993). Time-dependent self-organizing feature map for speech recognition, University of Twente, Netherlands.
- Harris, F. J. (1978). "On the use of windows for harmonic analysis with the discrete Fourier transform." Proceedings of the IEEE **66**: 51-83.
- Hart, P. E. (1968). "The Condensed Nearest Neighbor Rule." IEEE Transactions on Information Theory: 515-516.

- Hastie, T. and W. Stuetzle (1989). "Principal curves." Journal of the American Statistical Association **84**(406): 502-516.
- Hastie, T. and R. Tibshirani (1996). "Discriminant Adaptive Nearest Neighbor Classification." IEEE Transactions on Pattern Analysis and Machine Intelligence **18**(6): 607-615.
- Haykin, S. (1999). Neural Networks: A Comprehensive Foundation.
- Hemminger, T. L. and Y.-H. Pao (1994). "Detection and classification of underwater acoustic transients using neural networks." IEEE Transactions on Neural Networks **5**(5): 712-718.
- Herbrich, R. (2001). Learning Kernel Classifiers: Theory and algorithms, MIT Press.
- Hertz, J. A., A. Krogh and R. G. Palmer (1991). Introduction to the Theory of Neural Computation, Perseus Publishing.
- Himberg, J. (2000). A SOM based cluster visualization and it's application for false coloring. IJCNN'2000 - International Joint Conference on Neural Networks, Como, Italy, IEEE-Press.
- Hippenstiel, R. D. and P. M. d. Oliveira (1988). Instantaneous Power Spectrum Twenty-Second Asilomar Conference on Signals, Systems and Computers.
- Hippenstiel, R. D. and P. M. d. Oliveira (1990). "Time-varying spectral estimation using the instantaneous power spectrum (IPS)." IEEE Transactions on Acoustics, Speech and Signal Processing **38**(10): 1752-1759.
- Ho, K. C., Y. T. Chan and M. O. Johnson (1996). Estimation of delay and Doppler by wavelet transform. ICASSP-96, Atlanta, GA, USA.
- Ho, S.-Y., C.-C. Liu and S. Liu (2002). "Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm." Pattern Recognition Letters **23**: 1495-1503.
- Holmstrom, L., P. Koistinen, J. Laaksonen and E. Oja (1997). "Neural and Statistical Classifiers—Taxonomy and Two Case Studies." IEEE TRANSACTIONS ON NEURAL NETWORKS **8**(1): 5-17.
- Huang, J., J. Zhao and Y. Xie (1997). Source classification using pole method of AR model. ICASSP-97.
- Huang, Y. S., K. Liu, C. Y. Suen, A. J. Shie, I. I. Shyu, M. C. Liang, R. Y. Tsay and P. K. Huang (1996). A Simulated Annealing Approach to Construct Optimized Prototypes for Nearest-Neighbor Classification. 13th International Conference on Pattern Recognition, Vienna, Austria.
- Huang, Y. S., C. C. Chiang, J. W. Shieh and E. Grimson (2002). "Prototype optimization for nearest-neighbor classification." Pattern Recognition **35**: 1237-1245.
- Hyvarinen, A. and E. Oja (2000). "Independent component analysis: algorithms and applications." Neural Networks **13**: 411-430.
- Jain, A. K., R. C. Dubes and C. Chen (1987). "Bootstrap techniques for error estimation." IEEE Transactions on Pattern Analysis and Machine Intelligence **9**(5): 628-633.
- Jain, A. K. and R. C. Dubes (1988). Algorithms for clustering data, Prentice Hall.
- James, D. and R. Miikkulainen (1995). SARDNET: A self-organizing Feature Map for Sequences. Advances in Neural Information Processing Systems. G. Tesauro, D. Touretzky and T. Leen, MIT Press. **7**.
- Jauffret, C. and D. Bouchet (1996). Frequency line tracking on a lofargram: An efficient wedding between probabilistic data association modeling and dynamic programming technique. Asilomar Conference.
- Jesus, S. M., P. Felisberto and F. Coelho (1994). Towed array geometry estimation during ship's maneuvering. Journal of the Acoustical Society of America.
- Jesus, S. M., P. Felisberto and F. Coelho (1996). "Towed array beamforming during ship's maneuvering." IEE Proceedings: Radar, Sonar and Navigation **141**(3): 210-215.
- Jiang, X., Z. Gong, F. Sun and H. Chi (1994). A speaker recognition system based on auditory model. WCNN'93 - World Conference on Neural Networks, Lawrence Erlbaum, Hillsdale.
- Jolliffe, I. T. (1986). Principal component analysis. New York :, Springer-Verlag,.
- Jossa, I., U. Marschner and W. J. Fischer (2001). Signal based feature extraction and SOM based dimension reduction in a vibration monitoring microsystem. Advances in Self-Organizing Maps. N. Allison, H. Yin, L. Allison and J. Slack, Springer: 283-288.
- Joutsiniemi, S. L., S. Kaski and T. A. Larsen (1995). "Self-Organizing Map in Recognition of Topographic Patterns of EEG Spectra." IEEE Transactions on Biomedical Engineering **42**(11): 1062-1068.
- Kangas, J. (1992). Temporal Knowledge in Locations of Activations in a Self-Organizing Map. Artificial Neural Networks. J. T. : I. Aleksander, Elsevier Science Publisher. **2**: 117-120.
- Kangas, J., K. Tarkkola and M. Kokkonen (1992). Using SOMS as feature extractors for speech recognition. ICASSP 92, San Francisco, USA.
- Kangas, J. (1999). Prototype search for a nearest neighbor classifier by a genetic algorithm Third International Conference on Computational Intelligence and Multimedia Applications, 1999 - ICCIMA '99, New Delhi, India.
- Kangas, J. A., T. K. Kohonen and J. T. Laaksonen (1990). "Variants of Self-Organizing Maps." IEEE Transactions on Neural Networks **1**(1): 93-99.
- Kankas, J. (1994). On the Analysis of Pattern Sequences by Self-Organizing Maps. Laboratory of Computer and Information Science. Helsinki, Helsinki University of Technology: 86.

- Kanungo, T., D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A. Y. Wu (2002). "An efficient k-means clustering algorithm: analysis and implementation." IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(7): 881-892.
- Karayiannis, N. B., J. C. Bezdek, N. R. Pal, R. J. Hathaway and P.-I. Pai (1996). "Repairs to GLVQ: a new family of competitive learning schemes." IEEE Transactions on Neural Networks **7**(5): 1062-1071.
- Kaski, S. and S. L. Joutsiniemi (1993). Monitoring EEG Signal with the Self-Organizing Map. Intl. Conf. on Artificial Neural Networks (ICANN 93), Springer Verlag.
- Kaski, S. (1997). Data exploration using self-organizing maps. Helsinki, Finland, Helsinki University of Technology.
- Kaski, S., J. Venna and T. Kohonen (1999). Coloring that Reveals High-Dimensional Structures in Data. Proceedings of ICONIP'99, 6th International Conference on Neural Information Processing, London, IEEE Service Center.
- Kasslin, M., J. Kangas and O. Simula (1992). Process State Monitoring using Self-Organizing Maps. Artificial Neural Networks. I. Aleksander and J. Taylor, Elsevier Science Publisher. **2**: 1531-1534.
- Kay, S. M. (1988). Modern Spectral Estimation, Prentice-Hall.
- Kay, S. M. (1998). Fundamentals of statistical signal processing. Englewood Cliffs, N.J. :, Prentice-Hall PTR.,
- Keenan, R. E. and I. Dyer (1984). "Noise from Artic Ocean earthquakes." Journal of the Acoustical Society of America **75**: 819-825.
- Kégl, B., A. Krzyzak, T. Linder and K. Zeger (2000). "Learning and Design of Principal Curves." IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(3): 281-297.
- Kempke, C. and A. Wichert (1993). Hierarchical Self-Organizing Feature Maps for Speech Recognition. WCNN'93 - World Conference on Neural Networks, Lawrence Erlbaum, Hillsdale.
- Kilber, D. and D. W. Aha (1987). Learning representative examples of concepts: an initial study. Machine Learning, 4th International Workshop on.
- Kim, B.-s., S. H. Lee and D. K. Kim (1993). Determination of initial configuration for LVO by using CNN. 1993 International Joint Conference on Neural Networks, 1993. IJCNN '93, Nagoya.
- Kinsler, L. E. (1982). Fundamentals of acoustics. New York :, Wiley.,
- Kirkpatrick, S., C. D. Gelatt Jr. and M. P. Vecchi (1983). "Optimization by Simulated Annealing." Science **220**: 671-680.
- Kleppe, J. A. (1989). Engineering applications of acoustics. Norwood, MA :, Artech House.,
- Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. IJCAI 95.
- Kohonen, T. (1982). Clustering, Taxonomy, and Topological Maps of Patterns. Proceedings of the 6th International Conference on Pattern Recognition.
- Kohonen, T., K. Makisara and T. Saramaki (1984). Phonotopic Maps - Insightful Representation of Phonological Features For Speech Recognition. 7th International Conference on PATTERN RECOGNITION, Montreal Canada.
- Kohonen, T. (1988). "The 'neural' phonetic typewriter." IEEE Computer **21**(3): 11-22.
- Kohonen, T. (1991). The Hypermap Architecture. Artificial Neural Networks. T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, Elsevier Science Publishers. **1**: 1357-1360.
- Kohonen, T. (1995). Self-Organizing Maps, Springer.
- Kohonen, T., J. Hynninen, J. Kangas and J. Laaksonen (1995). The Self-Organizing Map Program Package, Helsinki University of Technology.
- Kohonen, T. (2001). Self-Organizing Maps, Springer.
- Kolodner, J. L. (1993). Case-based reasoning. San Mateo, CA :, Morgan Kaufmann Publishers.,
- Kong, X., R. Wang and G. Li (2002). "Fuzzy clustering algorithms based on resolution and their application in image compression." Pattern Recognition **35**(11): 2439-2444.
- Kopecz, K. (1995). Unsupervised learning of sequences on maps with lateral connectivity. ICANN'95 - International Conference on Neural Networks.
- Koskela, T., M. Varsta, J. Heikkonen and K. Kaski (1997). Time Series Prediction using Recurrent SOM with Local Linear Models. Helsinki, Helsinki University of Technology Laboratory of Computational Engineering: 17.
- Koskela, T., Varsta, M., H. J. and K. K. (1998). Temporal Sequence Processing using Recurrent SOM. KES'98, 2nd Int. Conf. on Knowledge-Based Intelligent Engineering Systems, Adelaide, Australia.
- Koskela, T., M. Varsta, J. Heikkonen and K. Kaski (1998). Recurrent SOM with Local Linear Models in Time Series Prediction. ESANN'98, 6th European Symposium on Artificial Neural Networks, Brussels, Belgium.
- Krishna, K., M. A. L. Thathachar and K. R. Ramakrishnan (2000). "Voronoi Networks and Their Probability of Misclassification." IEEE Transactions on Neural Networks **11**(6): 1361-1372.
- Kulkarni, S. R., S. E. Posner and S. Sandilya (1998). Data-dependant k-nn estimors consistent for arbitrary processes. IEEE International Symposium on Information Theory, Cambridge, MA, USA.

- Kuncheva, L. I. (1995). "Editing for the k-nearest neighbors rule by a genetic algorithm." Pattern Recognition Letters **16**: 809-814.
- Kuncheva, L. I. and J. C. Bezdek (1998). "Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search ?" IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **28**(1): 160 -164.
- Kuncheva, L. I. (2001). Reducing the Computational Demand of the Nearest Neighbor Classifier. School of Informatics Symposium on Computing 2001, Aberystwyth, UK.
- Laha, A. and N. R. Pal (2001). "Some novel classifiers designed using prototypes extracted by a new scheme based on self-organizing feature map." IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics **31**(6): 881-890.
- Lakany, H. M. (2001). Humain Gait Analysis using SOM. Advances in Self-Organizing Maps. N. Allison, H. Yin, L. Allison and J. Slack, Springer: 29-38.
- Lee, E.-W. and S.-I. Chae (1998). "Fast Design of Reduced-Complexity Nearest-Neighbor Classifiers Using Triangular Inequality." IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(5): 562-566.
- Leinonen, L., K. J. and A. Juvas (1992). "Dysphonia Detected by Pattern Recognition of Spectral Composition." Journal of Speech and Hearing Research **35**: 287-295.
- Leinonen, L., T. Hiltunen, K. Torkkola and J. Kangas (1993). "Self-organized Acoustic Feature Map in Detection of Forcative-Vowel Coarticulation." J. Acoust. Soc. Am. **6**: 3468-3474.
- Lim, T.-S., W.-Y. Loh and T.-S. Shih (2000). "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms." Machine Learning **40**(3): 203-228.
- Lin, S., J. Si and A. B. Schwartz (1998). "Self-organization of Firing Activities in Monkey's Motor Cortex: Trajectory Computation from Spike Signals." Neural Computation **9**(3): 607-621.
- Linde, Y., A. Buzo and M. Gray (1980). "An algorithm for vector quantization design." IEEE Transactions on Communications **28**: 84-95.
- Liu, C.-L. and M. Nakagawa (2001). "Evaluation of prototype learning algorithms for nearest-neighbor classifier in application to handwritten character recognition." Pattern Recognition **34**(3): 601-615.
- Lobo, V. (1995). Classificacao de efeitos hidrofonicos. Departamento de Informatica. Lisboa, Universidade Nova de Lisboa.
- Lobo, V. and F. Moura-Pires (1995). Ship noise classification using Kohonen Networks. EANN 95, Helsinki, Finland.
- Lobo, V. (1998). Tutorial on SOM for EAIA'98. Lisbon, Escola Naval.
- Lobo, V., R. Swinarski and F. Moura-Pires (1998). Pruning a classifier based on a Self-Organizing Map using Boolean function formalization. WCCI - World Conference on Computational Intelligence, Anchorage, Alaska, USA, IEEE Press.
- Lobo, V. and P. M. d. Oliveira (1999). Relatorio das gravacoes de efeitos hidrofonicos no tanque acustico do GOAME, em Julho de 1999. Lisbon, Escola Naval.
- Lobo, V. (2002). Ship noise classification: a contribution to prototype based classifier design. Departamento de Informatica. Lisbon, Universidade Nova de Lisboa.
- Lohse, D., B. Schmitz and M. Versluis (2001). "Snapping shrimp make flashing bubbles." Nature **413**(6855): 477-478.
- Lourens, J. G. (1988). Classification of Ships using Underwater Radiated Noise. COMSIG 88 - South African Symposium on Communications and Signal Processing, Pretoria.
- Lourens, J. G. (1997). Passive Sonar ML Estimator for Ship Propeller Speed. COMSIG 97 - South African Symposium on Communication and Signal Processing, 1997.
- Lloyd, S. P. (1982). "Least Squares quantization in PCM." IEEE Transactions on Information Theory **28**(2): 129-137.
- Lyons, A. R., T. J. Newton, N. J. Goddard and A. T. Parsons (1995). Can passive sonar signals be classified on the basis of their higher order statistics? IEE Colloquium on Higher Order Statistics in Signal Processing.
- MacQueen, J. (1967). Some methods for classification and analysis of multivariate observation. 5th Berkeley Symposium on Mathematical Statistics and Probability, University of California Press.
- Maher, M. L., M. Balachandran and D. M. Zhang (1995). Case-based reasoning in design. Mahwah, N.J. :, Lawrence Erlbaum Associates.
- Marques, J. S. (1999). Reconhecimento de Padrões, IST Press.
- Martinetz, T. M., S. G. Berkovich and K. J. Schulten (1993). "Neural-Gas network for vector quantization and its application to time-series prediction." IEEE Transactions on Neural Networks **4**(4): 558-569.
- Mathworks (2001). Matlab Users Manual, Mathworks.
- Medwin, H. and C. S. Clay (1998). Fundamentals of acoustical oceanography. Applications of modern acoustics. Boston :, Academic Press,: xx, 712 p. : ill. (some col.) ; 24 cm.
- Mehta, S. K., J. Fay and P. Maciejewski (1996). A modified Eckart post-beamformer filter for improved detection using broadband features. ICASSP-96.
- Meister, J. (1993). "A neural network harmonic family classifier." J. Acoust. Soc. Am. **93**(3): 1488-1495.

- Messer, H. (1994). "The use of spectral information in optimal detection of a source in the presence of a directional interference." IEEE Journal of Oceanic Engineering **19**(3): 422-430.
- Michie, D., D. J. Spiegelhalter and C. C. Taylor (1994). Machine Learning, Neural and Statistical Classification, Ellis Horwood.
- Midenet, S. and A. Grumbach (1994). "Learning Associations by Self-Organization: The LASSO Model." Neurocomputing **6**: 343-361.
- Miller, R. G. (1974). "The Jackknife - A review." Biometrika **61**(1): 1-15.
- Mitchell, T. M. (1997). Machine Learning, McGraw-Hill.
- Mitiche, A. and M. Lebidoff (2001). "Pattern classification by a condensed neural network." Neural Networks **14**(4-5): 575-580.
- Mitra, P., C. A. Murthy and S. K. Pal (2000). Data condensation in large databases by incremental learning with support vector machines. 15th International Conference on Pattern Recognition, Barcelona, Spain.
- Moizao, R. (1997). Projecto Final de Curso. Departamento de Informatica, Lisbon, Universidade Nova de Lisboa.
- Moshou, D. and H. Ramon (2000). "Wavelets and Self-Organizing Maps in financial time-series analysis." Neural Network World (10): 231-238.
- Mueller, G. (1993). Monitoring impacts on inland fisheries using hydroacoustics. Denver, Colo. :, U.S. Dept. of the Interior, Bureau of Reclamation, Denver Office.,
- Mujunen, R., L. Leinonen, J. Kangas and K. Torkkola (1993). "Acoustic Pattern Recognition of /s/ Misarticulation by the Self-Organizing Map." Folia Phoniatica **45**: 135-144.
- Muller, F. J. (1993). A Software Environment for Rough Sets. Mathematics and Computer Science. San Diego, CA, Sand Diego State University: 272.
- Mullin, M. and R. Sukthankar (1999). An efficient technique for calculating exact nearest-neighbor classification accuracy. NIPS 99.
- Murtagh, F. (1995). "Interpreting the Kohonen Self-Organizing Maps using contiguity-constraint clustering." Pattern Recognition Letters (16): 399-408.
- Musman, S. A., L. W. Chang and L. B. Booker (1990). A real time control strategy for Bayesian belief networks with application to ship classification problem solving. IEEE Conference on Tools for Artificial Intelligence, 1990.
- Nabney, I. T. (2001). Netlab, Springer.
- Natarajan, B. K. (1991). Machine Learning, Morgan Kaufmann.
- NATO (1993). Linked Seas 93 - Environmental Briefing Docket. Lisbon, Nato-Cinciberlant: 28.
- Nielsen, R. O. (1991). Sonar Signal Processing, Artec House.
- Nikias, C. L. and A. P. Petropulu (1993). Higher-Order spectra analysis: a nonlinear signal processing framework, Prentice-Hall PTR.
- Nishimura, C. E. (1994). Monitoring whales and earthquakes by using SOSUS. Washington, DC :, Naval Research Laboratory.,
- Nock, R. and M. Sebban (2001). "An improved bound on the finite-sample risk of the nearest neighbor rule." Pattern Recognition Letters **22**(3-4): 407-412.
- Oliveira, P. M. d. and V. Barroso (1998). On the concept of instantaneous frequency. International Conference on Acoustics, Speech and Signal Processing ICASSP98, Washington.
- Oliveira, P. M. d. and V. Barroso (1999). "Instantaneous frequency of multicomponent signals." IEEE Signal Processing Letters **6**(4): 81-83.
- Oppenheim, A. V. and R. W. Shafer (1989). Discrete-Time Signal Analysis, Prentice Hall.
- Pao, Y.-H. (1989). "Functional Link Nets: removing hidden layers." AI Expert: 60-68.
- Pawlak, Z. (1982). "Rough Sets." International Journal of Computer and Information Sciences **11**(5): 341-356.
- Pawlak, Z. (1988). "Hard Sets and Soft Sets." Bulletin of the Polish Academy of Sciences **36**: 119-123.
- Pawlak, Z. and R. Slowinski (1994). "Decision Analysis Using Rough Sets." Pergamon **1**(1): 107-114.
- Pawlak, Z. and R. Slowinski (1994). "Rough set approach to multi-attribute decision analysis." European Journal of Operational Research, Elsevier **72**: 443-459.
- Pesu, L., E. Ademovic, J. C. Pesquet and P. Helisto (1996). Wavelet packet based respiratory sound classification. IEEE-SP: International Symposium on Time-Frequency and Time-Scale Analysis, New York, USA, IEEE Press.
- Peterson, D. W. (1970). "Some Convergence Properties of a Nearest Neighbor Decision Rule." IEEE Transactions on Information Theory **16**(1): 26-31.
- Pizzuti, C. (1996). Computing Prime Implicants by Integer Programming. Int. Conf. on Tools with Artificial Intelligence.
- Plutowski, M., G. Cottrell and H. White (1996). "Experience with selecting exemplars from clean data." Neural Networks **9**(2): 273-294.
- Poggio, T. and F. Girosi (1990). "Networks for approximation and learning." Proceedings of the IEEE **78**(9): 1481-1497.

- Poularikas, A. D. (1998). The handbook of formulas and tables for signal processing, IEEE Press.
- Powell, M. D. J. (1987). Radial Basis Functions for multivariable interpolation: a review. Algorithms for approximation. J. C. Mason and M. G. Cox, Oxford: Clarendon Press: 143-167.
- Pridham, R. G. and D. J. Hamilton (1991). Evaluation of neural network and conventional techniques for sonar signal discrimination. IEEE Conference on Neural Networks for Ocean Engineering, 1991.
- Príncipe, J. and L. Wang (1995). Non-Linear Time Series Modeling with Self-Organization Feature Maps. IEEE Workshop on Neural Networks for Signal Processing, IEEE.
- Principe, J. C., N. R. Euliano and W. C. Lefebvre (2000). Neural and Adaptive Systems: Fundamentals through Simulation, John Wiley & Sons.
- Przytula, K. W., V. K. Prasanna, V. K. P. Kumar and K. W. Praytula (1993). Parallel Digital Implementations of Neural Networks, Prentice-Hall.
- Psaltis, D., R. R. Snapp and S. S. Venkatesh (1994). "On the finite sample Performance of the Nearest Neighbor Classifier." IEEE Transactions on Information Theory **40**(3): 820-837.
- Pujol, A. W., H.; Villanueva, J.J. (2001). Learning and caricaturing the face space using self-organization and Hebbian learning for face processing. Image Analysis and Processing, 11th International Conference on, Palermo, Italy.
- Pumphrey, H. C., L. A. Crum and L. Bjorno (1989). "Underwater sound produced by individual drop impacts and rainfall." Journal of the Acoustical Society of America **85**: 1518-1526.
- Qian, S. (2002). Time-Frequency and Wavelet Transforms, Prentice-Hall PTR.
- Quazi, A. H. (1996). Nonlinear sonar signal processing. Signal Processing, 1996., 3rd International Conference on.
- Ramasubramanian, V. and K. K. Paliwal (1992). "Fast K-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding." IEEE Transactions on Signal Processing **40**(3): 518-531.
- Ren, Q. S. and A. J. Willis (1995). A maximum entropy approach to filtering and reconstructive imaging of the underwater environment. OCEANS '95. MTS/IEEE.
- Ricci, F. and P. Avesani (1999). "Data Compression and Local Metrics for Nearest Neighbor Classification." IEEE Transactions on Pattern Analysis and Machine Intelligence **21**(4): 380-384.
- Ripley, B. D. (1996). Pattern recognition and neural networks, Cambridge University Press.
- Ritter, G. L., H. B. Woodruff, S. R. Lowry and T. L. Isenhour (1975). "An Algorithm for a Selective Nearest Neighbor Decision Rule." IEEE Transactions on Information Theory: 665-669.
- Ritter, H. and T. Kohonen (1989). "Self-Organizing Maps." Biological Cybernetics **61**: 241-254.
- Ritter, H., T. M. Martinez and K. Schulten (1992). Neural Computation and Self-Organizing Maps: an introduction, Addison-Wesley.
- Ritter, H. (1994). Parametrized Self-Organizing Maps for Visual Learning Tasks. ICANN'94 - International Conference on Neural Networks, Springer.
- Rogers, W. and T. J. Wagner (1978). "A finite-sample distribution-free performance bound for local discrimination rules." Annals of statistics **6**: 506-514.
- Roitblat, H. L., Moore, Nachtigall and Penner (1989). Dolphin echolocation: Identification of returning echoes. IJCNN 89.
- Ross, D. (1987). Mechanics of Underwater Noise. Los Altos, CA, Peninsula Publishing.
- Rueping, S. G., K.; Rueckert, U. (1994). A chip for self-organizing feature maps. Microelectronics for Neural Networks and Fuzzy Systems, Fourth International Conference on, Turin, Italy.
- Ruf, B. and M. Schmitt (1998). "Self-organization of spiking neurons using action potential timing." IEEE TRANSACTIONS ON NEURAL NETWORKS **9**(3).
- Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986). Learning internal representations by error propagation. Parallel distributed processing: Exploring times in the microstructure of cognition. D. E. Rumelhart and J. L. McClelland. Cambridge, MA, USA, Bradford Books. **1**.
- Russo, A. P. (1991). Constrained neural networks for recognition of passive sonar signals using shape. IEEE Conference on Neural Networks for Ocean Engineering.
- Salzberg, S. (1991). "A Nearest Hyperrectangle Learning Method." Machine Learning **6**: 277-309.
- Sammon, J. W. J. (1969). "A Nonlinear Mapping for Data Structure Analysis." IEEE Transactions on Computers **C-18**(5): 401-409.
- Sarker, R. A., H. A. Abbass and C. S. Newton (2002). Heuristics & Optimization for Knowledge Discovery, IGP - Idea Group Publishing.
- Schank, R. (1982). Dynamic memory: a theory of reminding and learning in computers and people. Cambridge, UK, Cambridge University Press.
- Schildt, H. (1989). Born to Code in C, McGraw Hill.
- Schneeweiss, W. G. (1989). Boolean Functions with Engineering Applications and Computer Programs, Springer-Verlag.
- Schurmann, J. (1996). Pattern classification: a unified view of statistical and neural approaches, John Wiley & Sons.

- Schwartzmann, D. I. and J. J. Vidal (1975). "An Algorithm for Determining the Topological Dimensionality of Point Clusters." IEEE Transactions on Computers **c-24**(12): 1175-1182.
- Springer, J. A., D. J. Evans and W. Yee (1989). "Underwater Noise due to rain: open ocean measurements." Journal of the Acoustical Society of America **85**: 726-731.
- Sejnowski, T. J. and P. Gorman (1988). "Learned Classification of Sonar Targets Using a Massively Parallel Network." IEEE Transactions on Acoustics, Speech, and Signal Processing **36**(7): 1135-1140.
- Short, R. D. and K. Fukunaga (1980). A new nearest neighbor distance measure. IEEE Int. Conf. Pattern Recognition, Miami Beach, FL.
- Short, R. D. and K. Fukunaga (1981). "The optimal distance measure for nearest neighbor classification." IEEE Transactions on Information Theory **27**: 622-627.
- Silva, J. P. M. (1997). On Computing Minimum Size Prime Implicants. IEEE/ACM International Workshop on Logic Synthesis.
- Simula, O., E. Alhoniemi, J. Hollmen and J. Vesanto (1996). Monitoring and Modeling of Complex Processes Using Hierarchical Self-Organizing Maps. IEEE International Symposium on Circuits and Systems (ISCAS-96), Atlanta, Georgia, USA.
- Skodras, A. N., C. A. Christopoulos and T. Ebrahimi (2000). JPEG2000: The Upcoming Still Image Compression Standard. RECPAD 2000, Porto, Portugal.
- Sokal, R. R. and P. H. A. Sneath (1963). Principals of Numerical Taxonomy, W.H. Freeman.
- Solinsky, J. C. and E. A. Nash (1991). Neural-network performance assessment in sonar applications. IEEE Conference on Neural Networks for Ocean Engineering, 1991.
- Song, B. C. and J. B. Ra (2002). "A fast search algorithm for vector quantization using L2-norm pyramid of codewords." IEEE Transactions on Image Processing **11**(1): 10-15.
- Specht, D. F. (1990). "Probabilistic Neural Networks." Neural Networks **3**: 109-118.
- Srinivasa, N. and N. Ahuja (1999). "A topological and temporal correlator network for spatiotemporal pattern learning, recognition, and recall." IEEE TRANSACTIONS ON NEURAL NETWORKS **10**(2): 356-371.
- Stefanick, T. (1987). Strategic antisubmarine warfare and naval strategy. Lexington, Mass. :, Lexington Books.,
- Stiles, B. W. and J. Ghosh (1995). Habituation based neural classifiers for spatio-temporal signals. ICASSP-95 - Intl. Conf. on Acoustics, Speech, and Signal Processing.
- Stockdale, D. L. (1998). Application of Zernike moments, rough sets and neural networks to handwritten character recognition. Mathematics and Computer Science. San Diego, SDSU: 236.
- Stoffel, D., W. Kunz and S. Gerber (1997). AND/OR Reasoning Graphs for Determining Prime Implicants in Multi-Level Combinational Networks. Asia and South Pacific Design Automation Conference ASP-DAC, Chiba, Japan, IEEE CS Press, Los Alamitos.
- Swonger, C. W. (1972). Sample Set Condensation for a Condensed Nearest Neighbor Decision Rule for Pattern Recognition. Frontiers of Pattern Recognition. S. Watanabe, Academic Press: 511-519.
- Tai, S. C., C. C. Lai and Y. C. Lin (1996). "Two fast nearest neighbor searching algorithms for image vector quantization." IEEE Transactions on Communications **44**(12): 1623-1628.
- Takacs, B. and H. Wechsler (1998). Face recognition using binary image metrics. Automatic Face and Gesture Recognition, Third IEEE International Conference on, Nara, Japan.
- Tanamaru, J. and A. Inubushi (1995). A compact representation of binary patterns for invariant recognition. IEEE International Conference on Systems, Man and Cybernetics.
- Tesei, A., C. S. Regazzoni and G. Tacconi (1994). Comparison between different HOS-based tests for detection of ship-radiated signals in non-Gaussian noise. OCEANS '94.
- Tims, A. C. and T. A. Henriquez (1979). Hydrophone phase stability analysis at frequencies below 100 Hz. NRL report :. Washginton, D.C. :, Naval Research Laboratory., **8314**: iii, 21 p. : ill. ; 26 cm.
- Tomek, I. (1976). "An experiment with the edited nearest neighbor rule." IEEE Transactions on Systems, Man, and Cybernetics.
- Tomek, I. (1976). "Two Modifications of CNN." IEEE Transactions on Systems, Man, and Cybernetics: 769-772.
- Torgo, L. and J. Gama (1997). "Regression using Classification Algorithms." Intelligent Data Analysis **1**(4): 275-292.
- Toussaint, G. T. and R. S. Poulsen (1979). Some new algorithms and software implementation methods for pattern recognition research. IEEE Computer Society Third International Computer Software and Applications Conference (COMPSAC'79).
- Trunk, G. V. (1968). "Statistical Estimation of the Intrinsic Dimensionality of Data Collections." Information and Control **12**: 508-525.
- Tryba, V. and K. Goser (1991). Self-Organizing Maps for Process Control in Chemistry. Artificial Neural Networks. T. K. Kohonen, K. Makisara, O. Simula and J. Kangas. Amsterdam, Netherlands: 847-852.
- Tyack, P. L. and T. Howald (1993). "Biological sources of noise in coastal waters." Journal of the Acoustical Society of America **94**(3): 1819.

- Ullmann, J. R. (1974). "Automatic Selection of Reference Data for use in a Nearest Neighbor Method of Pattern Classification." IEEE Transactions on Information Theory: 541-543.
- Ultsch, A. and H. P. Simeon (1989). Exploratory Data Analysis Using Kohonen Networks on Transputers, Department of Computer Science, University of Dortmund, FRG.
- Ultsch, A. and H. P. Siemon (1990). Kohonen's Self-Organizing Neural Networks for Exploratory Data Analysis. Intl. Neural Network Conf. INNC90, Paris.
- Ultsch, A. (1993). Self-Organized Feature Maps for Monitoring and Knowledge Acquisition of Chemical Process. ICANN 93 - International Conference on Artificial Neural Networks.
- Ultsch, A. and H. Li (1993). Automatic Acquisition of Symbolic Knowledge from Subsymbolic Neural Networks. International Conference on Signal Processing, Peking, 1993, Peking.
- Ultsch, A., G. Guimarães and W. Schmidt (1996). Classification and prediction of hail using self-organizing neural networks. ICNN'96 - International Conference on Neural Networks, Washington.
- Urick, R. J. (1982). Sound propagation in the sea. Los Altos, Calif. :, Peninsula.,
- Urick, R. J. (1986). Ambient noise in the sea. Los Altos, CA :, Peninsula Publishing.,
- Urquhart, R. B. (1983). "Mapping Techniques in Pattern Recognition." Geographical Research 1(2): 108-121.
- Utela, P., J. Kangas and L. Leinonen (1992). Self-Organizing Map in Acoustic Analysis and ON-Line Visual Imaging of Voice Articulation. Artificial Neural Networks 2. I. Aleksander and J. Taylor. Amsterdam, Netherlands. 1: 791-794.
- Van-Houtte, P., K. Deegan and K. Khorasani (1991). Passive sonar processing using neural networks. IEEE International Joint Conference on Neural Networks, 1991.
- Vapnik, V. N. (2000). The nature of statistical learning theory. New York :, Springer.,
- Varsta, M., J. Heikkonen and J. R. Millán (1997). Context Learning with self-organizing map. WSOM'97 - Workshop on Self-Organizing Maps, Espoo, Finland.
- Varsta, M., J. Heikkonen and J. Lampinen (2000). Analytical comparison of the Temporal Kohonen Map and the Recurrent Self Organizing Map. ESANN 00.
- Vesanto, J. (1999). "SOM-based data visualization." Intelligent Data Analysis 3: 111-126.
- Vesanto, J. and E. Alhoniemi (2000). "Clustering of the Self-Organizing Map." IEEE Transactions on Neural Networks.
- Voegtlin, T. (2000). Context Quantization and Contextual Self-Organizing Maps. IJCNN'2000.
- Voegtlin, T. and P. Dominey (2001). Recursive Self-Organizing Maps. Advances in Self-Organizing Maps. N. Allison, H. Yin, L. Allison and J. Slack, Springer: 210-215.
- Waibel, A., T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang (1989). "Phoneme recognition using time-delay neural networks." IEEE Transactions on Acoustics, Speech and Signal Processing 37(3): 328-339.
- Walter, J. A. and K. J. Schulten (1993). "Implementation of Self-Organizing Neural Networks for Visual-Motor Control of an Industrial Robot." IEEE Transactions on Neural Networks 4(1): 86-96.
- Walter, J. A. and H. Ritter (1996). Investment learning with hierarchical PSOM. Advances in Neural Information Processing Systems 8. D. S. Touretzky, M. Mozer and M. Hasselmo : 570-576.
- Walter, J. A. (1998). PSOM Network: Learning with Few Examples. Intl. Conf. On Robotics and Automation (ICRA), IEEE.
- Watson, I. D. (1997). Applying case-based reasoning : techniques for enterprise systems. San Francisco, Calif. :, Morgan Kaufmann.,
- Webb, A. (1999). Statistical pattern recognition. London [England] : New York, NY :, Arnold ; Co-published in the United States of America by Oxford University Press.,
- Wegener, I. (1987). The complexity of Boolean functions, J. Wiley.
- Wenz, G. M. (1962). "Acoustic Ambient Noise in the Ocean: Spectra and Sources." Journal of the Acoustical Society of America.
- Werbos, P. J. (1974). Beyond regression: New tools for predictions and analysis in the behavioral sciences. Cambridge, MA, USA, Harvard University.
- Wettschereck, D. and T. Dietterich (1995). "An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms." Machine Learning 19(1): 5-28.
- Wettschereck, D., T. Mohri and D. W. Aha (1997). "A review and empirical comparison of feature weighting Methods for a class of lazy learning algorithms." AI Review 11: 273-314.
- Wigner, E. P. (1932). "On the quantum correction for thermodynamic equilibrium." Physical Review 40: 749-759.
- Wilfong (1991). Nearest Neighbor Problems. 7th ACM Symposium on Computational Geometry.
- Wilson, D. L. (1972). "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data." IEEE Transactions on Systems, Man, and Cybernetics 2(3): 408-421.
- Wilson, D. R. and T. R. Martinez (1997). Instance Pruning Techniques. 14th International Conference on Machine Learning, Morgan Kaufmann Publishers.
- Wilson, D. R. and T. R. Martinez (1997). "Improved Heterogeneous Distance Functions." Journal of Artificial Intelligence Research 11: 1-34.

-
- Xie, Y. (1992). "Acoustical radiation from thermally stressed sea ice." Journal of the Acoustical Society of America **89**(5): 2215-2231.
- Yen, J. and C.-W. Chang (1994). A Multi-Prototype Fuzzy c-Means Algorithm European Congress on Intelligent Techniques and Soft Computing (EUFIT 94), Aachen, Germany.
- Zarndt, F. (1995). A Comprehensive Case Study: An Examination of Machine Learning and Connectionist Algorithms. Department of Computer Science, Brigham Young University.
- Zhang, H. and G. Sun (2002). "Optimal reference subset selection for nearest neighbor classification by tabu search." Pattern Recognition **35**(7): 1481-1490.
- Zhang, J. (2002). "Selecting Typical Instances in Instance Based Learning."
- Zhao, Q. and T. Higuchi (1996). "Evolutionary Learning of Nearest-Neighbor Multilayer Perceptrons." IEEE Transactions on Neural Networks **7**(3): 762-767.