FUZZY MODELS AND ALGORITHMS FOR PATTERN RECOGNITION AND IMAGE PROCESSING

THE HANDBOOKS OF FUZZY SETS SERIES

FUZZY MODELS AND ALGORITHMS FOR PATTERN RECOGNITION AND IMAGE PROCESSING

James C. Bezdek
University of West Florida

James Keller
University of Missouri

Raghu Krisnapuram
Colorado School of Mines

Nikhil R. Pal
Indian Statistical Institute

Springer

Library of Congress Cataloging-in-Publication Data

Fuzzy models and algorithms for pattern recognition and image processing / James C. Bezdek ... [et al.].
p. cm. – (The handbooks of fuzzy sets series)
Includes bibliographical references and index.
ISBN 0-387-24515-4 (softcover : alk. paper)
ISBN 0-7923-8521-7 (hardcover) © 1999 Kluwer Academic Publishers
1. Optical pattern recognition. 2. Fuzzy algorithms. 3. Cluster analysis. 4. Image processing. 5. Computer vision. I. Bezdek, James C., 1939- II. Series.

TA1650.F89 2005
006.4'2--dc22

2005042541

Printed in the United States of America.

9 8 7 6 5 4 3 2 1 SPIN 11384601

springeronline.com

Contents

Series Foreword

Fuzzy sets were introduced in 1965 by Lotfi Zadeh with a view to reconcile mathematical modeling and human knowledge in the engineering sciences. Since then, a considerable body of literature has blossomed around the concept of fuzzy sets in an incredibly wide range of areas, from mathematics and logic to traditional and advanced engineering methodologies (from civil engineering to computational intelligence). Applications are found in many contexts, from medicine to finance, from human factors to consumer products, from vehicle control to computational linguistics, and so on.... Fuzzy logic is now used in the industrial practice of advanced information technology.

As a consequence of this trend, the number of conferences and publications on fuzzy logic has grown exponentially, and it becomes very difficult for students, newcomers, and even scientists already familiar with some aspects of fuzzy sets, to find their way in the maze of fuzzy papers. Notwithstanding circumstantial edited volumes, numerous fuzzy books have appeared, but, if we except very few comprehensive balanced textbooks, they are either very specialized monographs, or remain at a rather superficial level. Some are even misleading, conveying more ideology and unsustained claims than actual scientific contents.

What is missing is an organized set of detailed guidebooks to the relevant literature, that help the students and the newcoming scientist, having some preliminary knowledge of fuzzy sets, get deeper in the field without wasting time, by being guided right away in the heart of the literature relevant for her or his purpose. The ambition of the HANDBOOKS OF FUZZY SETS is to address this need. It will offer, in the compass of several volumes, a full picture of the current state of the art, in terms of the basic concepts, the mathematical developments, and the engineering methodologies that exploit the concept of fuzzy sets.

This collection will propose a series of volumes that aim at becoming a useful source of reference for all those, from graduate students to senior researchers, from pure mathematicians to industrial information engineers as well as life, human and social sciences scholars, interested in or working with fuzzy sets. The original feature of these volumes is that each chapter - except in the case of this volume, which was written entirely by the four authors - is written by one or several experts in the topic concerned. It provides an introduction to the topic, outlines its development, presents the major results, and supplies an extensive bibliography for further reading.

The core set of volumes are respectively devoted to fundamentals of fuzzy sets, mathematics of fuzzy sets, approximate reasoning and information systems, fuzzy models for pattern recognition and image processing, fuzzy sets in decision research and statistics, fuzzy systems in modeling and control, and a guide to practical applications of fuzzy technologies.

D. Dubois H. Prade
Toulouse

Preface

The authors Rather than compile many chapters written by various authors who use different notations and semantic descriptions for the same models, we decided to have a small team of four persons write the entire volume. Each of us assumed the role of lead author for one or more of the chapters, and the other authors acted like consultants to the lead author. Each of us helped the lead author by contributing examples, references, diagrams or text here and there; and we all reviewed the entire volume three times. Whether this approach was successful remains to be seen.

The plan What we tried to do is this: identify the important work that has been done in fuzzy pattern recognition, describe it, analyze it, and illustrate it with examples that an interested reader can follow. As with all projects of this kind, the material inevitably reflects some bias on the part of its authors (after all, the easiest examples to give already live in our own computers). Moreover, this has become an enormous field, and the truth is that it is now far too large for us to even *know about* many important and useful papers that go unrecognized here. We apologize for our bias and our ignorance, and accept any and all blame for errors of fact and/or omission. How current is the material in the book? Knuth (1968) stated that "It is generally very difficult to keep up with a field that is economically profitable, and so it is only natural to expect that many of the techniques described here eventually be superseded by better ones". We cannot say it better.

The numbering system The atomic unit for the numbering system is the chapter. Figures, tables, examples and equations are all numbered consecutively within each chapter. For example, Figure 3.5 is Figure 5 of Chapter 3. The beginning and end of examples are enclosed by goofy looking brackets, like this:

Example 5.4 Did you ever have to finally decide? To pick up on one and let the other one ride, so many changes,.......

The algorithms: art, science and voodoo There are a lot of algorithms in the book. We ran many, but not certainly not all, of the experiments ourselves. We have given pseudo code for quite a few algorithms, and it is really pseudo in the sense that it is a mixture of three or four programming languages and writing styles. Our intent is to maximize clarity and minimize dependence on a particular language, operating system, compiler, host platform, and so on. We hope you can read the pseudo code, and that you can convert it into working programs with a minimum of trouble.

Almost all algorithms have parameters that affect their performance. Science is about quantitative models of our physical world, while art tries to express the qualitative content of our lives. When you read this book you will encounter lots of parameters that are user-defined, together with evasive statements like "pick a value for k that is close to 1", or "don't use high values for m". What do instructions such as these mean? Lots of things: (i) we don't have better advice; (ii) the inventor of the algorithm tried lots of values, and values in the range mentioned produced the best results for her or him; (iii) 0.99 is closer to 1 than 0.95, and 22 is higher than 1.32, you may never know which choice is better, and (unfortunately) this can make all the difference in your application; (iv) sometimes we don't know why things work the way they do, but we should be happy if they work right this time - call it voodoo, or call it luck, but if it works, take it.

Is this cynical? No, it's practical. Science is *NOT* exact, it's a sequence of successively better approximations by models we invent to the physical reality of processes we initiate, observe or control. There's a lot of art in science, and this is nowhere more evident than in pattern recognition, because here, the data always have the last word. We are always at the mercy of an unanticipated situation in the data; unusual structures, missing observations, improbable events that cause outliers, uncertainty about the interactions between variables, useless choices for numerical representation, sensors that don't respect our design goals, computers that lose bits, computer programs that have an undetected flaw, and so on. When you read about and experiment with algorithmic parameters, have an open mind - anything is possible, and usually is.

The data Most of the numerical examples use small data sets that may seem contrived to you, and some of them are. There is much to be said for the pedagogical value of using a few points in the plane when studying and illustrating properties of various models. On the other hand, there are certain risks too. Sometimes conclusions that are legitimate for small, specialized data sets become invalid in the face of large numbers of samples, features and classes. And of course, time and space complexity make their presence felt in very unpredictable ways as problem size grows.

There is another problem with data sets that everyone probably knows about, but that is much harder to detect and document, and that problem goes under the heading of, for example, "*will the <u>real</u> Iris data please stand up?*". Anderson's (1935) Iris data, which we think was first published in Fisher (1936), has become a popular set of labeled data for testing - and especially for comparing - clustering algorithms and classifiers. It is of course entirely appropriate and in the spirit of scientific inquiry to make and publish comparisons of models and their performance on common data sets, and the pattern recognition community has used Iris in perhaps a thousand papers for just this reason - - - <u>or have we</u>?

During the writing of this book we have discovered - perhaps others have known this for a long time, but we didn't - that there are at least two (and hence, probably half a dozen) different, well publicized versions of Iris. Specifically, vector 90, class 2 (Iris Versicolor) in Iris has the coordinates (5.5, 2.5, 4, 1.3) on p. 566, Johnson and Wichern (1992); and has the coordinates (5.5, 2.5, 5, 1.3) on p. 224 in Chien (1978). YIKES !! For the record, we are using the Iris data as published in Fisher (1936) and repeated in Johnson and Wichern (1992). We will use *Iris (?)* when we are not sure what data were used.

What this means is that many of the papers you have come to know and love that compare the performance of this and that using Iris may in fact have examples of algorithms that were executed using different data sets! What to do? Well, there isn't much we can do about this problem. We have checked our own files, and they all contain the data as listed in Fisher (1936) and Johnson and Wichern (1992). That's not too reassuring, but it's the best we can do. We have tried to check which Iris data set was used in the examples of other authors that are discussed in this book, but this is nearly impossible. We do not guarantee that all the results we discuss for "the" Iris data really pertain to the same numerical inputs. Indeed, the "Lena" image is the Iris data of image processing, - after all, the original Lena was a poor quality, 6 bit image, and more recent copies, including the ones we use in this book, come to us with higher resolution. To be sure, there is only one analog Lena (although PLAYBOY ran many), but there are probably many different digital Lenae.

Data get corrupted many ways, and in the electronic age, it should not surprise us to find (if we can) that this is a fairly common event. Perhaps the best solution to this problem would be to establish a central repository for common data sets. This has been tried several times without much success. Out of curiosity, on September 7, 1998 we fetched Iris from the anonymous FTP site "ftp.ics.uci.edu" under the directory "pub/machine-learning-databases", and discovered not one, but *two* errors in it! Specifically, two vectors in Iris Sestosa were wrong: vector 35 in Fisher (1936) is (4.9, 3.1, 1.5, 0.2) but in the machine learning electronic database it had coordinates (4.9, 3.1, 1.5, 0.1); and vector 38 in Fisher is (4.9, 3.6, 1.4, 0.1), but in the electronic database it was (4.9, 3.1, 1.5, 0.1). Finally, we are aware of several papers that used a version of Iris obtained by multiplying every value by 10, so that the data are integers, and the papers involved discuss 10*Iris as if they thought it was Iris. We don't think there is a way to correct all the databases out there which contain similar mistakes (we trust that the machine learning database will be fixed after our alert), but we have included a listing of Iris in Appendix 2 of this book (and, we hope it's right). What all this means for you, the pattern recognition aficionado is this: *pattern recognition* **is** *data, and not all data are created equally, much less replicated faithfully!*

Numerical results We have tried to give you all the information you need to *replicate* the outputs we report in numerical examples. There are a few instances where this was not possible (for example, when an iterative procedure was initialized randomly, or when the results were reported in someone's paper 10 or 15 years ago, or when the authors of a paper we discuss simply could not supply us with more details), and of course it's always possible that the code we ran implemented something other than we thought it did, or it simply had undetected programming errors. Also, we have rounded off or truncated the reported results of many calculations to make tables fit into the format of the book. Let us know if you find substantial differences between outputs you get (or got) and the results we report.

The references More than one reference system is one too many. We chose to reference books and papers by last names and years. As with any system, this one has advantages and disadvantages. Our scheme lets you find a paper quickly if you know the last name of the first author, but causes the problem of appending "a", "b" and so on to names that appear more than once in the same year. There may be a mistake or two, or even $\boldsymbol{O}(n)$ of them. Again, please let us know about it. We have divided the references into two groups: those actually cited in the text, and a second set of references that point to related material that, for one reason or another, just didn't find their way into the text discussion. Many of these uncited papers are excellent - please have a look at them.

The acronyms Acronyms, like the plague, seem to spread unchecked through the technical literature of pattern recognition. We four are responsible for quite a few of them, and so, we can hardly hold this bad habit against others. This book has several hundred acronyms in it, and we know you won't remember what many of them mean for more than a few pages. Consequently, Appendix 1 is a tabulation of the acronyms and abbreviations used in the text.

Acknowledgments The authors wish to acknowledge their gratitude to the following agencies, who graciously supplied partial support as shown during the writing of this book:

J. C. Bezdek :	ONR grant # N00014-96-1-0642 NSF grant # IRI-9003252
J. M. Keller :	ONR grant # N00014-96-1-0439 ARO MURI grant # DAAG55-97-1-0014
R. Krishnapuram:	ONR grant # N00014-96-1-0439 NSFgrant # IRI-9800899

We also want to express our thanks to Andrea Baraldi, Alma Blonda, Larry Hall, Lucy Kuncheva and Thomas Runkler, all of whom were kind enough to review various parts of the manuscript and/or supplied us with computations for several examples that we could not find in the literature, and whose helpful comments save us at least a few embarrassments.

The quotes Everyone nowadays seems to have a pithy quote at each chapter head, at the end of each email, on their web page, tattooed on their leg, etc., so we wanted to have some too. Rather than choose one quote for the book that all of us could live with (quite a range of tastes exists amongst us four), we decided to each supply one quote for this preface. We give the quotes here, but don't identify who contributed each one. That will be revealed in the pages of this volume - but only to those readers alert enough to *recognize the patterns*.

"What use are all these high-flying vaunts of yours?
O King of Birds! You will be the world's laughing stock.
What a marvel would it be if the hare
were to void turd the size of elephant dung!"

Vishnu Sharma, in Panchatantra, circa AD 400

"Only the mediocre are always at their best"

Blue Wave, circa 1995

"All uncertainty is fruitful ... so long as it is accompanied by the wish to understand"

Antonio Machado, Juan de Mairena, 1943

"You gotta pay your dues if you want to play the blues, and you know that don't come easy"

Ringo Starr, circa 1973

You may think you know which of us contributed each of these quotes - but you might be surprised. Life is full of surprises, and so is this book. We hope you enjoy both.

Jim Bezdek
Jim Keller
Rags Krishnapuram
Nik Pal

1 Pattern Recognition

1.1 Fuzzy models for pattern recognition

There is no lack of definitions for the term pattern recognition. Here are a few that we like.

Fukunaga (1972, p. 4): "pattern recognition consists of two parts: feature selection and classifier design."

Duda and Hart (1973, p. vii) "pattern recognition, a field concerned with machine recognition of meaningful regularities in noisy or complex environments".

Pavlidis (1977, p. 1): "the word *pattern* is derived from the same root as the word *patron* and, in its original use, means something which is set up as a perfect example to be imitated. Thus pattern recognition means the identification of the ideal which a given object was made after."

Gonzalez and Thomason (1978, p. 1) : "Pattern recognition can be defined as the categorization of input data into identifiable classes via the extraction of significant features or attributes of the data from a background of irrelevant detail."

Bezdek (1981, p. 1) : "pattern recognition is *a search* for *structure* in *data.*"

Schalkoff (1992, p. 2) " Pattern recognition (PR) is the science that concerns the description or classification (recognition) of measurements."

And here is our favorite, because it comes from the very nice book by Devijver and Kittler (1982, p. 2), titled *Pattern Recognition: A Statistical Approach*: "pattern recognition is a very broad field of activities with very fuzzy borders" !!!

What all these definitions should tell you is that it's pretty hard to know what to expect from a book with the term pattern recognition in its title. You will find texts that are mostly about computer science topics such as formal language theory and automata design (Fu, 1982), books about statistical decision theory (Fukunaga, 1972, 1991), books about fuzzy mathematics and models (Bezdek, 1981), books about digital hardware (Serrano-Gotarredona et al., 1998), handbooks (Ruspini et al., 1998), pure math books, books that contain only computer programs, books about graphical approaches, and so on. The easiest, and we think, most accurate overall description of this field is to say that it is about feature analysis, clustering, and classifier design, and that is what this book is about - the use of fuzzy models in these three disciplines.

Regardless of how it is defined, there are two major approaches to pattern recognition, numerical and *syntactic*. With the exception of Section 4.10, this book is exclusively concerned with the numerical approach. We characterize numerical pattern recognition with the four major areas shown in Figure 1.1. The nodes in Figure 1.1 are *not* independent. In practice, a successful pattern recognition system is developed by iteratively revisiting the four modules until the system satisfies (or is at least optimized for) a given set of performance requirements and/or economic constraints.

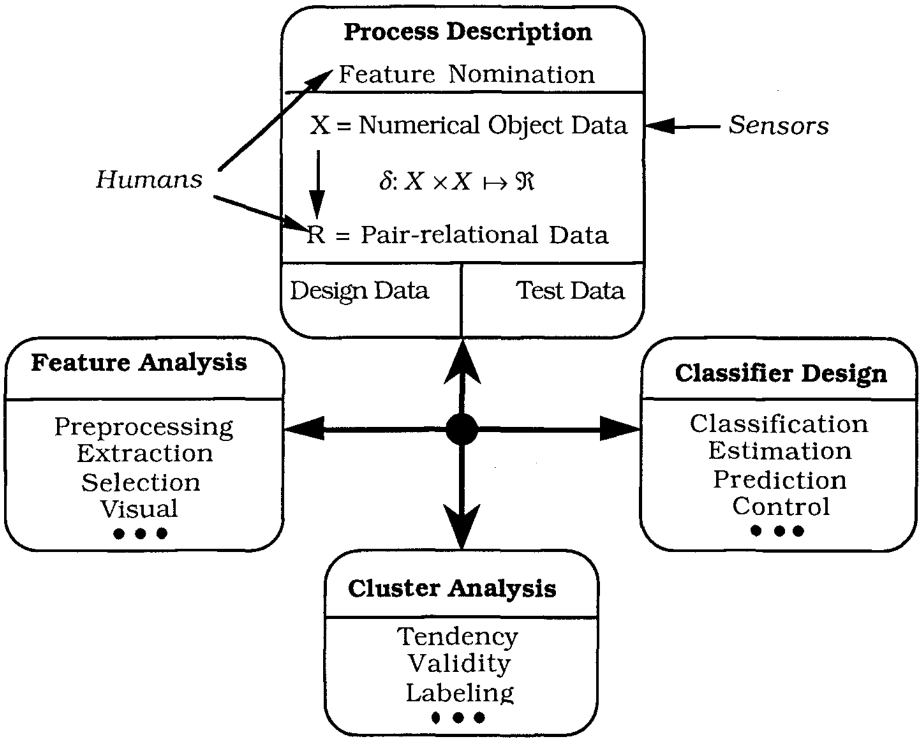


Figure 1.1 Typical elements of numerical pattern recognition

The upper block of Figure 1.1 - *process description* - is always done by humans. Things that must be accomplished here include the selection of a model type, features to be measured and sensors that can collect the data. This important phase of system design is not well represented in the literature because there are many factors such as time, space, weight, cost, speed, etc. that are too problem-dependent to admit much generality. You need to give careful thought to process description because your decisions here will be reflected in the ultimate performance of your system.

Notation Vectors are boldface ($\mathbf{x}$, $\mathbf{v}$, $\mathbf{V}$, etc.); $\mathbf{x} \in \Re^p$ is the $p \times 1$ matrix $\mathbf{x} = (x_1, \ldots, x_p)^T$. Matrices and set names are not shown boldface (even though a $c \times p$ matrix U is a vector in $\Re^{cp} = \Re^c \times \Re^p$). For the matrix $U \in \Re^{cp}$, we may write the <u>i-th row</u> as $\mathbf{U}_{(i)} \in \Re^p$, and the <u>k-th column</u> as $\mathbf{U}_k \in \Re^c$. By this convention, when interpreting U as a $cp \times 1$ column vector, we may write $\mathbf{U} = (\mathbf{U}_1, \ldots, \mathbf{U}_p) = (\mathbf{U}_{(1)}, \ldots, \mathbf{U}_{(p)})^T \in \Re^{cp}$. When interpreting the rows or columns of a matrix as a set, we use set brackets; e.g., the c rows $\mathbf{U} = (\mathbf{U}_{(1)}, \ldots, \mathbf{U}_{(c)}) \in \Re^{cp} \leftrightarrow U = \{\mathbf{U}_{(1)}, \ldots, \mathbf{U}_{(c)}\} \subset \Re^p$. We use **0** for the zero vector in all vector spaces; specifically, in both $\Re^p$ and $\Re^{cp}$.

Two data types are used in numerical pattern recognition: *object data* (feature or pattern vectors); and (pairwise) *relational data* (similarities, proximities, etc.). Object data are represented throughout the volume as $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$, a set of n feature vectors in *feature space* $\Re^p$. Writers in some fields call the features of each object "attributes", and others call them "characteristics". The j-th object is a physical entity such as a tank, medical patient, stock report, etc. *Column vector* $\mathbf{x}_j$ is it's numerical representation; x_{kj} is the k-th *feature* or *attribute value* associated with object j. Features can be either continuously or discretely valued in $\Re$.

We will also deal with non-numerical data called *categorical data* in Chapter 4. Categorical data have no natural order. For example, we can represent animals with numerical attributes such as number of legs, weight, etc. ; or we might describe each one with categorical attributes such as skin texture, which itself has values such as furry, feathery, etc. When needed, we denote the objects themselves as $O = \{o_1, o_2, \ldots, o_n\}$. Chapter 2 is about clustering numerical object data.

Instead of object data, we may have a set of (mn) numerical *relationships*, say $\{r_{jk}\}$, between *pairs* of objects (o_j, o_k) in $O_1 \times O_2$, $|O_1| = m$, $|O_2| = n$. The number r_{jk} represents the extent to which $o_j \in O_1$ is related to $o_k \in O_2$ in the sense of some binary relation ρ. It is convenient to array the relational values as an $m \times n$ *relation matrix* $R = [r_{jk}] = [\rho(o_j, o_k)]$. Many functions can convert object data into relational data. For example, every metric (distance measure) δ on $\Re^p \times \Re^p$ produces a square (dis)-similarity relation matrix $R(X; \delta)$ on the n objects represented by X, as shown in Figure 1.1. If every r_{jk} is in $\{0, 1\}$, R is a *crisp* binary relation. If any r_{jk} is in $[0, 1]$, we call R a *fuzzy* binary relation. Chapter 3 is about clustering relational data.

One of the most basic structures in pattern recognition is the *label vector.* No matter what kind of data you have (including the case of n objects as opposed to numerical data that represent them), there are four types of class labels - crisp, fuzzy, probabilistic and possibilistic. Letting n be the number of objects (or feature vectors or number of rows and columns in relational data) integer c denote the number of classes, $1 \le c \le n$. Ordinarily, c will not be 1 or n, but we admit this possibility to handle special cases that sometimes arise.

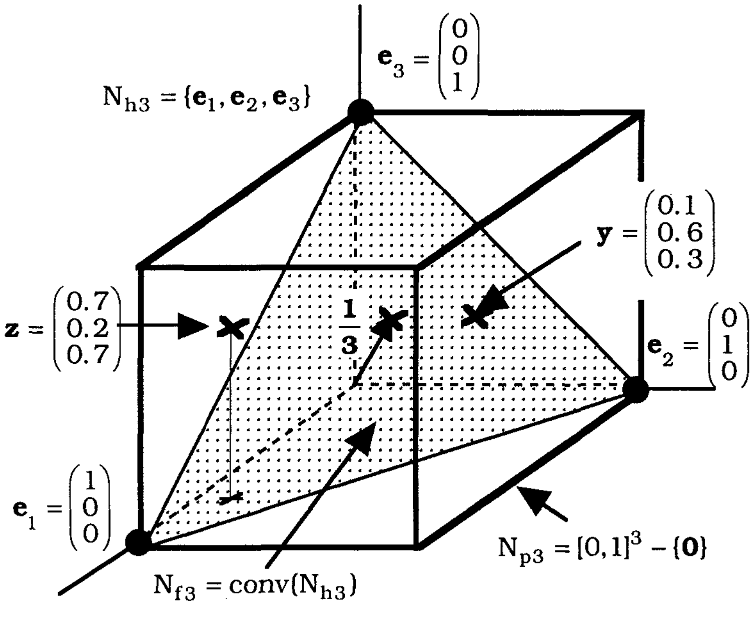
We define three sets of label vectors in $\Re^c$ as follows:

$$N_{pc} = \left\{ \mathbf{y} \in \Re^c : y_i \in [0, \ 1] \ \forall \ i, \ y_i > 0 \ \exists \ i \right\} = [0,1]^c - \{\mathbf{0}\}; \tag{1.1}$$

$$N_{fc} = \left\{ \mathbf{y} \in N_{pc} : \sum_{i=1}^{c} y_i = 1 \right\} \quad ; \tag{1.2}$$

$$N_{hc} = \left\{ \mathbf{y} \in N_{fc} : y_i \in \{0,1\} \, \forall \, i \right\} = \left\{ \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_c \right\} \quad . \tag{1.3}$$

In (1.1) $\mathbf{0}$ is the *zero vector* in $\Re^c$. Note that $N_{hc} \subset N_{fc} \subset N_{pc}$. Figure 1.2 depicts these sets for c = 3. N_{hc} is the canonical (unit vector) basis of Euclidean c-space, so $\mathbf{e}_i = (0, \ 0 \ , \ldots, \underbrace{1}_{i} , \ldots, \ 0)^T$, the i-th vertex of N_{hc}, is the *crisp label* for class i, $1 \le i \le c$.



$N_{h3} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$
$\mathbf{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
$\mathbf{y} = \begin{pmatrix} 0.1 \\ 0.6 \\ 0.3 \end{pmatrix}$
$\mathbf{z} = \begin{pmatrix} 0.7 \\ 0.2 \\ 0.7 \end{pmatrix}$
$\frac{1}{3}$
$\mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
$N_{p3} = [0,1]^3 - \{\mathbf{0}\}$
$N_{f3} = \mathrm{conv}(N_{h3})$

Figure 1.2 Label vectors for c = 3 classes

The set N_{fc}, a piece of a hyperplane, is the convex hull of N_{hc}. The vector $\mathbf{y} = (0.1, 0.6, 0.3)^T$ is a constrained label vector; its entries lie between 0 and 1, and sum to 1. The *centroid* of N_{fc} is the equimembership vector $\mathbf{1/c} = (1/c,\ldots,1/c)^T$. If $\mathbf{y}$ is a label vector for some $\mathbf{x} \in \Re^p$ generated by, say, the fuzzy c-means clustering method, we call $\mathbf{y}$ a *fuzzy label* for $\mathbf{x}$. If $\mathbf{y}$ came from a method such as maximum likelihood estimation in mixture decomposition, $\mathbf{y}$ would be a *probabilistic label*. In this case, $\mathbf{1/c}$ is the unique point of equal probabilities for all c classes.

$N_{pc} = [0, 1]^c\text{-}\{\mathbf{0}\}$ is the unit hypercube in $\Re^c$, *excluding the origin*. Vectors such as $\mathbf{z} = (0.7, 0.2, 0.7)^T$ with each entry between 0 and 1 that are otherwise unrestricted are *possibilistic labels* in N_{p3}. Possibilistic labels are produced by possibilistic clustering algorithms (Krishnapuram and Keller, 1993) and by computational neural networks that have unipolar sigmoidal transfer functions at each of c output nodes (Zurada, 1992).

Most pattern recognition models are based on finding statistical or geometrical properties of substructures in the data. Two of the key concepts for describing geometry are angle and distance. Let A be any positive-definite p×p matrix. For vectors $\mathbf{x}, \mathbf{v} \in \Re^p$, the functions $\langle\ \rangle_A : \Re^p \times \Re^p \mapsto \Re$, $\|\ \|_A : \Re^p \mapsto \Re^+$, and $\delta_A : \Re^p \times \Re^p \mapsto \Re^+$

$$\langle \mathbf{x}, \mathbf{v} \rangle_A = \mathbf{x}^T A \mathbf{v} \quad ; \qquad (1.4)$$

$$\|\mathbf{x}\|_A = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle_A} = \sqrt{\mathbf{x}^T A \mathbf{x}} \quad ; \text{ and} \qquad (1.5)$$

$$\delta_A(\mathbf{x}, \mathbf{v}) = \|\mathbf{x} - \mathbf{v}\|_A = \sqrt{(\mathbf{x} - \mathbf{v})^T A (\mathbf{x} - \mathbf{v})} \quad , \qquad (1.6)$$

are the inner product (dot product, scalar product), *norm* (length), and *norm metric* (distance) induced on $\Re^p$ by weight matrix A. We say that $\mathbf{x}$ and $\mathbf{v}$ are *orthogonal* (normal, perpendicular) if their dot product is zero, $\langle \mathbf{x}, \mathbf{v} \rangle_A = \mathbf{x}^T A \mathbf{v} = 0$. Sometimes we write $\mathbf{x} \perp_A \mathbf{v}$ to indicate this, and note particularly that orthogonality is always *relative to* matrix A that induces the inner product.

Equation (1.6) defines an infinite family of inner product induced distances, the most important three of which, together with their common names and inducing matrices, are:

$$\|\mathbf{x} - \mathbf{v}\|_I = \sqrt{(\mathbf{x} - \mathbf{v})^T (\mathbf{x} - \mathbf{v})} \qquad \textit{Euclidean}, A=I_p \quad ; \qquad (1.7)$$

$$\|\mathbf{x}-\mathbf{v}\|_{D^{-1}} = \sqrt{(\mathbf{x}-\mathbf{v})^T D^{-1}(\mathbf{x}-\mathbf{v})} \qquad \textit{Diagonal}, \ A=D^{-1} \quad ; \qquad (1.8)$$

$$\|\mathbf{x}-\mathbf{v}\|_{M^{-1}} = \sqrt{(\mathbf{x}-\mathbf{v})^T M^{-1}(\mathbf{x}-\mathbf{v})} \qquad \textit{Mahalanobis}, \ A=M^{-1}. \qquad (1.9)$$

Notation In (1.7) I_p is the $p \times p$ *identity matrix.* Henceforth, we drop the subscript I_p, writing the Euclidean forms of (1.4)-(1.6) more simply as $\langle \mathbf{x}, \mathbf{v} \rangle$, $\|\mathbf{x}\|$ and $\|\mathbf{x}-\mathbf{v}\|$ respectively.

Equations (1.8) and (1.9) use $M = \text{cov}(X) = \sum_{k=1}^{n} (\mathbf{x}_k - \bar{\mathbf{v}})(\mathbf{x}_k - \bar{\mathbf{v}})^T / n$, the *covariance matrix* of X, and $\bar{\mathbf{v}} = \sum_{k=1}^{n} \mathbf{x}_k / n$, the *grand mean* of X. We will always indicate sample means as in statistics, with an overbar. The matrix D is the diagonal matrix extracted from M by deletion of its off-diagonal entries, D = diag(M). D is *not* the diagonalized form of M.

A second infinite family of lengths and distances that are commonly used in pattern recognition are the *Minkowski norm* and *Minkowski norm metrics*

$$\|\mathbf{x}\|_q = \left(\sum_{j=1}^{p} |x_j|^q \right)^{\frac{1}{q}} , \ q \ge 1 \qquad ; \qquad (1.10)$$

$$\delta_q(\mathbf{x}, \mathbf{v}) = \|\mathbf{x}-\mathbf{v}\|_q = \left(\sum_{j=1}^{p} |x_j - v_j|^q \right)^{\frac{1}{q}} , \ q \ge 1 \qquad . \qquad (1.11)$$

Only three Minkowski distances are commonly used in pattern recognition, and the Minkowski 2-norm is just the Euclidean norm, $\|\mathbf{x}-\mathbf{v}\|_2 = \|\mathbf{x}-\mathbf{v}\|$:

$$\|\mathbf{x}-\mathbf{v}\|_1 = \left(\sum_{j=1}^{p} |x_j - v_j| \right) \qquad \text{City Block (1-norm); q=1;} \qquad (1.12)$$

$$\|\mathbf{x}-\mathbf{v}\|_2 = \left(\sum_{j=1}^{p} |x_j - v_j|^2 \right)^{\frac{1}{2}} \qquad \text{Euclidean (2-norm); q=2;} \qquad (1.13)$$

$$\|\mathbf{x}-\mathbf{v}\|_\infty = \underbrace{\max}_{1 \le j \le p} \{ |x_j - v_j| \} \qquad \text{Sup or Max norm; } q \to \infty. \qquad (1.14)$$

A *classifier* is any function $\mathbf{D}:\Re^p \mapsto N_{pc}$. The value $\mathbf{y} = \mathbf{D}(\mathbf{z})$ is the label vector for $\mathbf{z}$ in $\Re^p$. $\mathbf{D}$ is a *crisp classifier* if $\mathbf{D}[\Re^p] = N_{hc}$; otherwise, the classifier is fuzzy or probabilistic or possibilistic. Designing a classifier simply means finding the parameters of a "good" $\mathbf{D}$. This can be done with data, or it might be done by an expert without data. If the data are labeled, finding $\mathbf{D}$ is called *supervised learning*; otherwise, the problem is *unsupervised learning*. Notice that we use the terms supervised and unsupervised to specifically connote the use of labeled or unlabeled data - it is the *labels* that do (or do not) supervise the design. When an expert designs a classifier, this is certainly supervised design, but in a much broader sense than we mean here. Chapter 4 is about fuzzy models for classifier design.

Since definite class assignments are usually the ultimate goal of classification and clustering, outputs of algorithms that produce label vectors in N_{pc} or N_{fc} are usually transformed into crisp labels. Most non-crisp classifiers are converted to crisp ones using the function $\mathbf{H}:N_{pc} \mapsto N_{hc}$,

$$\mathbf{H}(\mathbf{y}) = \mathbf{e}_i \Leftrightarrow \|\mathbf{y}-\mathbf{e}_i\| < \|\mathbf{y}-\mathbf{e}_j\| \Leftrightarrow y_i > y_j \quad ; \quad j \neq i \quad . \tag{1.15}$$

In (1.15) ties are resolved arbitrarily. $\mathbf{H}$ finds the crisp label vector $\mathbf{e}_i$ in N_c closest (in the Euclidean sense) to $\mathbf{y}$. Alternatively, $\mathbf{H}$ finds the index of the *maximum coordinate* of $\mathbf{y}$, and assigns the corresponding crisp label to the object vector, say $\mathbf{z}$, that $\mathbf{y}$ labels. The rationale for using $\mathbf{H}$ depends on the algorithm that produces $\mathbf{y}$. For example, using (1.15) for outputs from the k-nearest neighbor rule is simple majority voting. If $\mathbf{y}$ is obtained from mixture decomposition, using $\mathbf{H}$ is Bayes decision rule - label $\mathbf{z}$ by its class of maximum posterior probability. And if the labels are fuzzy, this is called defuzzification by the maximum membership rule. We call the use of $\mathbf{H}$ *hardening*.

1.2 Why fuzzy pattern recognition?

Rather than conclude the volume with the information in this subsection, it is provided here to answer a basic question you might have at this point: *should you read on?* Retrieval from the *Science Citation Index* for years 1994-1997 on titles and abstracts that contain the keyword combinations "fuzzy" + either "clustering" or "classification" yielded 460 papers. Retrievals against "fuzzy" + either "feature selection" or "feature extraction" yielded 21 papers. This illustrates that the literature contains a large body of work on fuzzy clustering and classifier design, and relatively fewer studies of fuzzy models for feature analysis. Work in this last area is widely scattered because feature analysis is very data and problem-dependent, and hence, is almost always done on a case by case basis.

A more interesting metric for the importance of fuzzy models in pattern recognition lies in the diversity of applications areas represented by the titles retrieved. Here is a partial sketch:

Chemistry: analytical, computational, industrial, chromatography, food engineering, brewing science.

Electrical Engineering: image and signal processing, neural networks, control systems, informatics, automatics, automation, robotics, remote sensing and control, optical engineering, computer vision, parallel computing, networking, instrumentation and measurement, dielectrics, speech recognition, solid state circuits.

Geology/Geography: photogrammetry, geophysical research, geochemistry, biogeography, archeology.

Medicine: magnetic resonance imaging, medical diagnosis, tomography, roentgenology, neurology, pharmacology, medical physics, nutrition, dietetic sciences, anesthesia, ultramicroscopy, biomedicine, protein science, neuroimaging, drug interaction.

Physics: astronomy, applied optics, earth physics.

Environmental Sciences: soil sciences, forest and air pollution, meteorology, water resources.

Thus, it seems fair to assert that this branch of science and engineering has established a niche as a useful way to approach pattern recognition problems. The rest of this volume is devoted to some of the basic models and algorithms that comprise fuzzy numerical pattern recognition.

1.3 Overview of the volume

Chapter 2 discusses clustering with objective function models using object data. This chapter is anchored by the crisp, fuzzy and possibilistic c-means models and algorithms to optimize them that are discussed in Section 2.2. There are many generalizations and relatives of these three families. We discuss relatives and generalizations of the c-means models for both volumetric (cloud shaped) and shell clusters in Section 2.3. Roughly speaking, these two cases can be categorized as point and non-point prototype models. Section 2.3 also contains a short subsection on recent developments in the new area of robust clustering. Chapter 2 contains a long section on methods for validation of clusters after they are found - the important and very difficult problem of cluster validity. Separate subsections discuss methods that attempt to validate volumetric and shell type clusters; and this section concludes with a discussion of fuzzy versions of several well known statistical indices of validity. This is followed by a short section on feature analysis with references to a very few fuzzy methods for problems in this domain. Finally, we close Chapter 2 (and all subsequent chapters as well) with a section that contains comments and related references for further reading.

Chapter 3 is about two types of relational clustering: methods that use decompositions of relation matrices; and methods that rely on optimization of an objective function of the relational data. This is a much smaller field than clustering with objective function methods. The main reason that relational models and algorithms are less well developed than those for object data is that sensors in fielded systems almost always collect object data. There are, however, some very interesting applications that depend on relational clustering; for example, data mining and information retrieval in very large databases. We present the main topics of this area in roughly the same chronological order as they were developed. Applications of relational clustering are also discussed in the handbook volume devoted to information retrieval.

Chapter 4 discusses fuzzy models that use object data for classifier design. Following definitions and examples of the nearest single and multiple prototype classifiers, we discuss several sequential methods of prototype generation that were not covered in Chapter 2. Next, k-nearest neighbor rule classifiers are presented, beginning with the classical crisp k-nearest neighbor rule, and continuing through both fuzzy and possibilistic generalizations of it. Another central idea covered in Chapter 4 is the use of the fuzzy integral for data fusion and decision making in the classification domain. Following this, rule based designs are introduced through crisp and fuzzy decision trees in Section 4.6, which contains material about the extraction of fuzzy rules for approximation of functions from numerical data with clustering.

Chapter 4 next presents models and algorithms that draw their inspiration from *neural-like networks* (NNs). Two chapters in Nguyen and Sugeno (1998) by Pedrycz et al.,(1998) and Prasad (1998) discuss the use of fuzzy neurons and fuzzy NNs in the context of control and functional approximation. These chapters provide good ancillary reading to our presentation of related topics in the context of pattern recognition. The *feed forward* multilayered perceptron trained by *back propagation* (FFBP) is the dominant structure underlying "fuzzy neural networks" (neurofuzzy computing, etc.), so our discussion begins with this network as the standard classifier network. Then we present some generalizations of the standard node functions that are sometimes called fuzzy neurons. We discuss and illustrate perceptrons, multilayered perceptrons, and aggregation networks for classification. Then we discuss the crisp and several fuzzy generalizations of *adaptive resonance theory* (ART), including a short subsection on radial basis function networks. Section 4.9 is concerned with the increasingly important topic of classifier fusion (or multistage classification). The last section in Chapter 4 is a short section on the use of fuzzy models in syntactic pattern recognition. Our Chapter 4 comments include some material on feature analysis in the context of classifier design.

Chapter 5 is about image processing and computer vision. It is here that the models and algorithms discussed in previous chapters find realizations in an important application domain. Chapter 5 begins with low level vision approaches to image enhancement. Then we discuss edge detection and edge following algorithms. Several approaches to the important topic of image segmentation are presented next, followed by boundary description and surface approximation models. The representation of image objects as fuzzy regions is followed by a section on spatial relations. The last section in Chapter 5 discusses high level vision using fuzzy models. Chapter 7.3.2 of volume 7 of this handbook (Bezdek and Sutton, 1998) contains an extended discussion of fuzzy models for image processing in medical applications.

1.4 Comments and bibliography

There are many good treatments of deterministic, statistical and heuristic approaches to numerical pattern recognition, including the texts of Duda and Hart (1973), Tou and Gonzalez (1974), Devijver and Kittler (1982), Pao (1989) and Fukunaga (1991). Approaches based on neural-like network models are nicely covered in the texts by Zurada (1992) and Haykin (1994).

The earliest reference to the use of fuzzy sets in numerical pattern recognition was Bellman, Kalaba and Zadeh (1966). RAND Memo RM-4307-PR, October, 1964, by the same authors had the same title, and was written before Zadeh (1965). Thus, the first *application* envisioned for fuzzy models seems to have been in pattern recognition.

Fuzzy techniques for numerical pattern recognition are now fairly mature. Good references include the texts by Bezdek (1981), Kandel (1982), Pal and Dutta-Majumder (1986) and the edited collection of 51 papers by Bezdek and Pal (1992). Chi et al. (1997) is the latest entrant into this market, with a title so close to ours that it makes you wonder how many of these entries the market will bear. Surveys of fuzzy models in numerical pattern recognition include Keller and Qiu(1988), Pedrycz (1990b), Pal (1991), Bezdek (1993), Keller and Krishnapuram (1994), Keller et al. (1994) and Bezdek et al. (1997a).

2 Cluster Analysis for Object Data

2.1 Cluster analysis

Figure 2.1 portrays cluster analysis. This field comprises three problems: tendency assessment, clustering and validation. Given an unlabeled data set, ① *is there* substructure in the data? This is *clustering tendency* - should you look for clusters at all? Very few methods - fuzzy or otherwise - address this problem. Panayirci and Dubes (1983), Smith and Jain (1984), Jain and Dubes (1988), Tukey (1977) and Everitt (1978) discuss statistical and informal graphical methods (visual displays) for deciding what - if any - substructure is in unlabeled data.

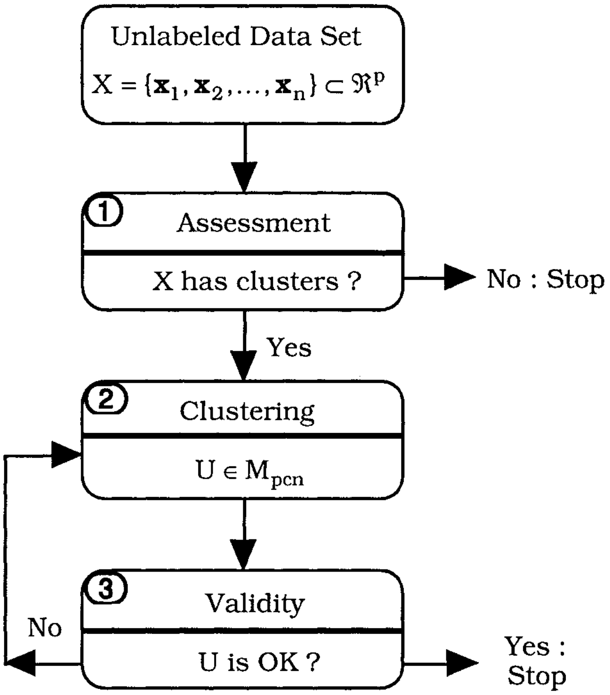


Figure 2.1 Cluster analysis: three problems

Once you decide to look for clusters (called U in ②, Figure 2.1), you need to choose a model whose measure of mathematical similarity may capture structure in the sense that a human might perceive it. This question - what criterion of similarity to use? - lies at the heart of all clustering models. We will be careful to distinguish between a model, and methods (algorithms) used to solve or optimize it. There are objective function (global criteria) and graph-theoretic (local criteria) techniques for both relational and object data.

Different algorithms produce different partitions of the data, and it is never clear which one(s) may be most useful. Once clusters are obtained, how shall we pick the best clustering solution (or solutions)? Problem ③ in Figure 2.1 is *cluster validity*, discussed in Section 2.4.

Problem ② in Figure 2.1 is *clustering* (or unsupervised learning) in unlabeled data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, which is the assignment of (hard *or* fuzzy *or* probabilistic *or* possibilistic) label vectors to the $\{\mathbf{x}_k\}$. The word *learning* refers to learning good labels (and possibly prototypes) for the clusters in the data.

A *c-partition* of X is a c×n matrix $U = [\mathbf{U}_1\ \mathbf{U}_k \ldots \mathbf{U}_n] = [u_{ik}]$, where $\mathbf{U}_k$ denotes the k-th column of U. There are three sets of c-partitions whose columns correspond to the three types of label vectors discussed in Chapter 1

$$M_{pcn} = \left\{ U \in \Re^{cn} : \mathbf{U}_k \in N_{pc} \forall k; 0 < \sum_{k=1}^{n} u_{ik}\ \forall i \right\} \quad ; \tag{2.1}$$

$$M_{fcn} = \left\{ U \in M_{pcn} : \mathbf{U}_k \in N_{fc}\ \forall k \right\} \quad ; \tag{2.2}$$

$$M_{hcn} = \left\{ U \in M_{fcn} : \mathbf{U}_k \in N_{hc} \forall k \right\} \quad . \tag{2.3}$$

Equations (2.1), (2.2) and (2.3) define, respectively, the sets of possibilistic, fuzzy or probabilistic, and crisp c-partitions of X. Each column of U in M_{pcn} (M_{fcn}, M_{hcn}) is a label vector from N_{pc} (N_{fc}, N_{hc}). Note that $M_{hcn} \subset M_{fcn} \subset M_{pcn}$. Our notation is chosen to help you remember these structures; M = (membership) matrix, h=crisp (hard), f=fuzzy (or probabilistic), p=possibilistic, c=number of classes and n=number of data points in X.

Notation For U in M_{fcn} c=1 is represented uniquely by the hard 1-partition $\mathbf{1}_n = \underbrace{[1\ \ 1\ \ \cdots\ \ 1]}_{n\ \ \text{times}}$, which asserts that all n objects belong to a single cluster; and c=n is represented uniquely by $U = I_n$, the n×n identity matrix, up to a permutation of columns. In this case each object is in its own singleton cluster. Crisp partitions have a familiar set-theoretic description that is equivalent to (2.1). When $U = \{X_1, \ldots, X_c\}$ is a crisp c-partition, the c crisp subsets $\{X_i\} \subset X$ satisfy $\bigcup_i X_i = X$; $X_i \cap X_j = \emptyset$ if $i \neq j$; and $X_i \neq \emptyset\ \forall i$. We denote the *cardinality* of a crisp set of n elements as $|X| = n$, and $|X_i| = n_i\ \forall i$.

Choosing c=1 or c=n rejects the hypothesis that X contains clusters in these cases. The lack of a column sum constraint for $U \in (M_{pcn} - M_{fcn})$ means that there are infinitely many U's in both $(M_{p1n} - M_{f1n})$ and $(M_{pnn} - M_{fnn})$.

The constraint $0 < \sum_{k=1}^{n} u_{ik} \; \forall i$ in equation (2.1) guarantees that each row in a c-partition contains at least one non-zero entry, so the corresponding cluster is not empty. Relaxing this constraint results in enlarging M_{pcn} to include matrices that have zero rows (empty clusters). From a practical viewpoint this is not desirable, but we often need this superset of M_{pcn} for theoretical reasons. We designate the sets of *degenerate* (crisp, fuzzy, possibilistic) *c-partitions* of X as $(M_{hcn0}, M_{fcn0}, M_{pcn0})$.

The reason these matrices are called *partitions* follows from the interpretation of their entries. If U is crisp or fuzzy, u_{ik} is taken as the *membership* of $\mathbf{x}_k$ in the i-th partitioning fuzzy subset (cluster) of X. If U is probabilistic, u_{ik} is usually the (posterior) probability $p(i \mid \mathbf{x}_k)$ that, given $\mathbf{x}_k$, it came from class (cluster) i. We indicate the statistical context by replacing $U = [u_{ik}]$ with $P = [p_{ik}] = [p(i \mid \mathbf{x}_k)]$. When U is possibilistic, u_{ik} is taken as the possibility that $\mathbf{x}_k$ belongs to class (cluster) i.

Clustering algorithms produce *sets* of label vectors. For fuzzy partitions, the usual method of defuzzification is the application of (1.15) to each column $\mathbf{U}_k$ of matrix U, producing the maximum membership matrix we sometimes call U_{MM} from U. We will formalize this operation as equation (2.10).

Example 2.1 Let X = {x_1 = peach, x_2 = plum, x_3 = nectarine}, and let c=2. Typical 2-partitions of these three objects are:

	$U_1 \in M_{h23}$			$U_2 \in M_{f23}$			$U_3 \in M_{p23}$		
Object	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
Peaches	1.0	0.0	0.0	1.0	0.2	0.4	1.0	0.2	0.5
Plums	0.0	1.0	1.0	0.0	0.8	0.6	0.0	0.8	0.6

The nectarine, x_3, is labeled by the last column of each partition, and in the crisp case, it must be (erroneously) given full membership in one of the two crisp subsets partitioning this data. In U_1, x_3 is labeled "plum". Non-crisp partitions enable models to (sometimes!)

avoid such mistakes. The last column of U_2 allocates most (0.6) of the membership of x_3 to the plums class; but also assigns a lesser membership (0.4) to x_3 as a peach. U_3 illustrates possibilistic label assignments for the objects in each class.

Finally, observe that hardening each column of U_2 and U_3 with (1.15) in this example makes them identical to U_1. Crisp partitions of data do not possess the information content to suggest fine details of infrastructure such as hybridization or mixing that are available in U_2 and U_3. Consequently, extract information of this kind before you harden U!

Columns like the ones for the nectarine in U_2 and U_3 serve a useful purpose - lack of strong membership in a single class is a signal to "take a second look". In this example the nectarine is a peach-plum *hybrid*, and the memberships shown for it in the last column of either U_2 or U_3 seem more plausible *physically* than crisp assignment of $\mathbf{x}_3$ to an incorrect class. M_{pcn} and M_{fcn} can be more realistic than M_{hcn} because boundaries between many classes of real objects are badly delineated (i.e., really fuzzy). M_{fcn} reflects the degrees to which the classes share $\{\mathbf{x}_k\}$, because of the constraint inherited from each fuzzy label vector (equation (1.2)) we have $\sum_{i=1}^{c} u_{ik} = 1$. M_{pcn} reflects the degrees of typicality of $\{\mathbf{x}_k\}$ with respect to the prototypical (ideal) members of the classes.

We believe that Bill Wee wrote the first Ph.D. thesis about fuzzy pattern recognition (Wee, 1967); his work is summarized in Wee and Fu (1969). Ruspini (1969) defined M_{fcn}, and Ruspini (1970) discussed the first fuzzy clustering method that produced constrained c-partitions of unlabeled (relational) data. Gitman and Levine (1970) first attempted to decompose "mixtures" (data with multimodality) using fuzzy sets. Other early work includes Woodbury and Clive (1974), who combined fuzziness and probability in a hybrid clustering model. In the same year, Dunn (1974a) and Bezdek (1974a) published papers on the fuzzy c-means clustering model. Texts that contain good accounts of various clustering algorithms include Duda and Hart (1973), Hartigan (1975), Jain and Dubes (1988), Kaufman and Rouseeuw (1990), Miyamoto (1990), Johnson and Wichern (1992), and the most recent members of the fold, Chi et al. (1996a) and Sato et al. (1997).

2.2 Batch point-prototype clustering models

Clustering models and algorithms that optimize them always deliver a c-partition U of X. Many clustering models estimate other parameters too. The most common parameters besides U that are associated with clustering are sets of vectors we shall denote by $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c\} \subset \Re^p$. The vector $\mathbf{v}_i$ is interpreted as a *point prototype* (centroid, cluster center, signature, exemplar, template, codevector) for the points associated with cluster i. Point prototypes are regarded as compact representations of cluster structure.

As just defined, $\mathbf{v}_i$ is a point in $\Re^p$, hence a *point-prototype*. Extensions of this idea to prototypes that are not just points in the feature space include $\mathbf{v}_i$'s that are linear varieties, hyperspherical shells, and regression models. General prototype models are covered in Section 2.3. Probabilistic clustering with normal mixtures produces simultaneous estimates of a $c \times n$ partition P (posterior probabilities), c mean vectors $\mathbf{M} = \{\mathbf{m}_1, \ldots, \mathbf{m}_c\}$, c covariance matrices $\{S_1, \ldots, S_c\}$ and c prior probabilities $\mathbf{p} = (p_1, \ldots, p_c)^T$. Some writers regard the triple $(p_i, \mathbf{m}_i, S_i)$ as the prototype for class i; more typically, however, $\mathbf{m}_i$ is considered the point prototype for class i, and other parameters such as p_i and S_i are associated with it through the model.

The basic form of iterative point prototype clustering algorithms in the variables (U, **V**) is

$$(U_t, \mathbf{V}_t) = \mathcal{C}(X : U_{t-1}, \mathbf{V}_{t-1}), t > 0 \quad , \tag{2.4a}$$

where $\mathcal{C}$ stands for the clustering algorithm and t is the index of iteration or recursion. Non-iterative models are dealt with on a case by case basis. When $\mathcal{C}$ is based on optimization of an objective function and *joint* optimization in (U, **V**) is possible, conditions (2.4a) can be written as $(U_t, \mathbf{V}_t) = \mathcal{C}(X : \mathcal{H}_{\mathcal{C}}(U_{t-1}, \mathbf{V}_{t-1}))$, where $\mathcal{H}_{\mathcal{C}}$ is determined by some optimality criterion for the clustering model. More typically however, alternating *optimization* (AO) is used, which takes the form of coupled equations such as

$$U_t = \mathcal{F}_{\mathcal{C}}(\mathbf{V}_{t-1}) \; ; \; \mathbf{V}_t = \mathcal{G}_{\mathcal{C}}(U_t) \; [\mathbf{V}\text{-initialization}]; \text{ or} \tag{2.4b}$$

$$\mathbf{V}_t = \mathcal{G}_{\mathcal{C}}(U_{t-1}); U_t = \mathcal{F}_{\mathcal{C}}(\mathbf{V}_t) \; [\text{U-initialization}]. \tag{2.4c}$$

The iterate sequences in (2.4b) or (2.4c) are equivalent. Both are exhibited to point out that you can start (initialize) and end (terminate) iteration with either U or **V**. Specific implementations use one or the other, and properties of either sequence (such as convergence) automatically follow for iteration started at the opposite set of variables. Examples of clustering models that have (U, **V**) as joint parameters are the batch hard, fuzzy and possibilistic c-means models. Alternating optimization of these models stems from functions $\mathcal{F}_{\mathcal{C}}$ and $\mathcal{G}_{\mathcal{C}}$ which arise from first order necessary conditions for minimization of the appropriate c-means objective function.

A. The c-means models

The c-means (or k-means) families are the best known and most well developed families of batch clustering models. Why? Probably because they are *least squares* models. The history of this methodology is long and important in applied mathematics because least-squares models have many favorable mathematical properties. (Bell (1966, p. 259) credits Gauss with the invention of the method of least squares for parameter estimation in 1802, but states that Legendre apparently published the first formal exposition of it in 1806.) The optimization problem that defines the *hard* (H), *fuzzy* (F) and *possibilistic* (P) *c-means* (HCM, FCM and PCM, respectively) models is:

$$\underset{(U,\mathbf{V})}{\underbrace{\min}}\left\{J_m(U,\mathbf{V};\mathbf{w})=\sum_{i=1}^{c}\sum_{k=1}^{n}u_{ik}^m D_{ik}^2+\sum_{i=1}^{c}w_i\sum_{k=1}^{n}(1-u_{ik})^m\right\},\text{ where} \tag{2.5}$$

$U \in M_{hcn}$, M_{fcn} or M_{pcn} for HCM, FCM or PCM respectively ;

$\mathbf{V}=(\mathbf{v}_1,\mathbf{v}_2,\ldots,\mathbf{v}_c)\in\Re^{cp}$; $\mathbf{v}_i\in\Re^p$ is the i-th point prototype ;

$\mathbf{w}=(w_1,w_2,\ldots,w_c)^T$; $w_i\in\Re^+$ is the i-th penalty term (PCM) ;

$m \ge 1$ is the degree of fuzzification ;

$D_{ik}^2=\|\mathbf{x}_k-\mathbf{v}_i\|_A^2$.

Note especially that **w** in (2.5) is a fixed, user-specified vector of positive weights; it is *not* part of the variable set in minimization problem (2.5).

Caveat: Model optima versus human expectations. The presumption in (2.5) is that "good" solutions for a pattern recognition problem - here clustering - correspond to "good" solutions of a mathematical optimization problem chosen to represent the physical process. Readers are warned not to expect too much from their models. In (2.5) the implicit assumption is that pairs (U, **V**) that are at least local minima for J_m will provide (i) good clusters U, and (ii) good prototypes **V** to represent those clusters. What's wrong with this? Well, it's easy to construct a simple data set upon which the *global* minimum of J_m leads to algorithmically suggested substructure that humans will disagree with (example 2.3). The problem? Mathematical models have a very rigid, well-defined idea of what best means, and it is often quite different than that held by human evaluators. There may not be any relationship between clusters that humans regard as "good" and the various types of extrema of any objective function. Keep this in mind as you try to understand your disappointment about the terrible clusters your favorite algorithm just found. The HCM, FCM and PCM clustering models are summarized in Table 2.1. Problem (2.5) is well defined for any distance function on $\Re^p$. The method chosen to approximate solutions of (2.5) depends primarily on D_{ik}. J_m is differentiable in U unless it is crisp, so first order necessary conditions for U are readily obtainable. If D_{ik} is differentiable in **V** (e.g., whenever D_{ik} is an inner product norm), the most popular technique for solving (2.5) is grouped coordinate descent (or *alternating optimization* (AO)).

Table 2.1 Optimizing $J_m(U, \mathbf{V}; \mathbf{w})$ **when** $D_{ik} = \|\mathbf{x}_k - \mathbf{v}_i\|_A$

	Minimize	*First order necessary conditions* for (U, **V**) when $D_{ik} = \|\mathbf{x}_k - \mathbf{v}_i\|_A > 0 \ \forall\, i, k$ (inner product norm case only)	
HCM	$J_1(U, \mathbf{V}; \mathbf{w})$ over $(U, \mathbf{V})$ in $M_{hcn} \times R^{cp}$ $w_i = 0 \ \forall\, i$	$u_{ik} = \begin{Bmatrix} 1; D_{ik} \le D_{ij}, j \ne i \\ 0; \quad \text{otherwise} \end{Bmatrix} \forall\, i, k;$	(2.6a)
		$\mathbf{v}_i = \dfrac{\sum_{k=1}^{n} u_{ik}\mathbf{x}_k}{\sum_{k=1}^{n} u_{ik}} = \dfrac{\sum_{\mathbf{x}_k \in X_i} \mathbf{x}_k}{n_i} = \bar{\mathbf{v}}_i \quad \forall\, i$	(2.6b)
FCM	$J_m(U, \mathbf{V}; \mathbf{w})$ over $(U, \mathbf{V})$ in $M_{fcn} \times R^{cp}$ $m > 1$ $w_i = 0 \ \forall\, i$	$u_{ik} = \left[\sum_{j=1}^{c} \left(\dfrac{D_{ik}}{D_{jk}} \right)^{\frac{2}{m-1}} \right]^{-1} \quad \forall\, i, k;$	(2.7a)
		$\mathbf{v}_i = \left(\sum_{k=1}^{n} u_{ik}^m \mathbf{x}_k \Big/ \sum_{k=1}^{n} u_{ik}^m \right) \quad \forall\, i$	(2.7b)
PCM	$J_m(U, \mathbf{V}; \mathbf{w})$ over $(U, \mathbf{V})$ in $M_{pcn} \times R^{cp}$ $w_i > 0 \ \forall\, i$	$u_{ik} = \left[1 + \left(D_{ik}^2 / w_i \right)^{\frac{1}{m-1}} \right]^{-1} \quad \forall\, i, k;$	(2.8a)
		$\mathbf{v}_i = \left(\sum_{k=1}^{n} u_{ik}^m \mathbf{x}_k \Big/ \sum_{k=1}^{n} u_{ik}^m \right) \quad \forall\, i$	(2.8b)

Column 3 of Table 2.1 shows the first order necessary conditions $U_t = \mathcal{F}_C(V_{t-1})$; $V_t = \mathcal{G}_C(U_t)$ for U and **V** at local extrema of J_m that each model requires at extreme points of its functional when the distance measure in (2.5) is an inner product norm metric. Derivations of the FCM and PCM conditions are made by zeroing the gradient of J_m with respect to **V**, and the gradient of (one term of) the

LaGrangian of J_m with respect to U. The details for HCM and FCM can be found in Bezdek (1981), and for PCM, see Krishnapuram and Keller (1993).

The second form, $\bar{\mathbf{v}}_i$ for $\mathbf{v}_i$ in (2.6b), emphasizes that optimal HCM-AO prototypes are simply the mean vectors or centroids of the points in crisp cluster i, $n_i = |\mathbf{U}_{(i)}|$, where $\mathbf{U}_{(i)}$ is the i-th row of U. Conditions (2.7) converge to (2.6) and $J_m \to J_1$ as m→1 from above. At the other extreme for (2.7), $\underbrace{\lim}_{m\to\infty}\{u_{ik}\} = 1/c \quad \forall\, i,k$ as m increases without bound, and $\underbrace{\lim}_{m\to\infty}\{\mathbf{v}_i\} = \bar{\mathbf{v}} = \sum_{k=1}^{n}\mathbf{x}_k \Big/ n \quad \forall\, i$ (Bezdek, 1981).

Computational singularity for u_{ik} in HCM-AO is manifested as a tie in (2.6a) and may be resolved arbitrarily by assigning the membership in question to any one of the points that achieves the minimum. Singularity for u_{ik} in FCM-AO occurs when one or more $\|\mathbf{x}_k - \mathbf{v}_{i,t}\|_A^2 = 0$ at any iterate. In this case (rare in practice), (2.7a) cannot be calculated. When this happens, assign 0's to each non-singular class, and distribute positive memberships to the singular classes arbitrarily subject to constraint $\sum_{i=1}^{c} u_{ik} = 1$. As long as the w_i's are positive (which they are by user specification), PCM-AO cannot experience this difficulty. Constraints on the $\{u_{ik}\}$ are enforced by the necessary conditions in Table 2.1, so the denominators for computing each $\mathbf{v}_i$ are always positive.

Table 2.2 specifies the c-means AO algorithms based on the necessary conditions in Table 2.1 *for the inner product norm case.* The case shown in Table 2.2 corresponds to (2.4b), initialization and termination on cluster centers **V**. The rule of thumb $c \le \sqrt{n}$ in the second line of Table 2.2 can produce a large upper bound for c. For example, this rule, when applied to clustering pixel vectors in a 256×256 image where n=65,536, suggests that we might look for c = 256 clusters. This is done, for example, in image compression, but for segmentation, the largest value of c that might make sense is more like 20 or 30. In most cases, a reasonable choice for c_{max} can be made based on auxiliary information about the problem. For example, segmentation of magnetic resonance images of the brain requires at most c = 8 to 10 clusters, as the brain contains no more than 8-10 tissue classes.

All three algorithms can get stuck at undesirable terminal estimates by initializing with cluster centers (or equivalently, rows of U_0) that have the same values because $\mathbf{U}_{(i)}$ and $\mathbf{v}_i$ are functions of just each other. Consequently, identical rows (and their prototypes) will remain identical unless computational roundoff forces them to become different. However, this is easily avoided, and should never present a problem.

Table 2.2 The HCM/FCM/PCM-AO algorithms

Store	Unlabeled Object Data $X \subset \Re^p$
	☛ number of clusters: $1 < c < n$
	☛ maximum number of iterations: T
	☛ weighting exponent: $1 \le m < \infty$ (m=1 for HCM-AO)
Pick	☛ inner product norm for J_m: $\|\mathbf{x}\|_A^2 = \mathbf{x}^T A \mathbf{x}$
	☛ termination measure: $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|$ = big value
	☛ termination threshold: $0 < \varepsilon$ = small value
	☛ weights $w_i > 0 \ \forall\ i$ ($\mathbf{w} = \mathbf{0}$ for FCM-AO/HCM-AO)
Guess	☛ initial prototypes: $\mathbf{V}_0 = (\mathbf{v}_{1,0}, \ldots, \mathbf{v}_{c,0}) \in \Re^{cp}$ (2.4b)
	$t \leftarrow 0$
	REPEAT
	$\quad t \leftarrow t+1$
Iterate	$\quad U_t = \mathcal{F}_c(\mathbf{V}_{t-1})$ where $\mathcal{F}_c(\mathbf{V}_{t-1})$ (cf. 2.6a, 2.7a or 2.8a)
	$\quad \mathbf{V}_t = \mathcal{G}_c(U_t)$ where $\mathcal{G}_c(U_t)$ (cf. 2.6b, 2.7b or 2.8b
	UNTIL (t=T or $E_t \le \varepsilon$)
	$(U, \mathbf{V}) \leftarrow (U_t, \mathbf{V}_t)$

The rows of U are completely decoupled in PCM because there is no constraint that $\sum_{i=1}^{c} u_{ik} = 1$. This can be an advantage in noisy data sets, since noise points and outliers can be assigned low memberships in all clusters. On the other hand, removal of the constraint that $\sum_{i=1}^{c} u_{ik} = 1$ also means that PCM-AO has a higher tendency to produce identical rows in U unless the initial prototypes are sufficiently distinct and the specified weights $\{w_i\}$ are estimated reasonably correctly (Barni et al., 1996, Krishnapuram and Keller, 1996). However, this behavior of PCM can sometimes be used to advantage for cluster validation - that is, to determine c, the number of clusters that are most plausible (Krishnapuram and Keller (1996)). *Krishnapuram and Keller recommend two ways to choose the* weights **w** for PCM-AO,

$$w_i = K\left(\sum_{k=1}^{n} u_{ik}^m D_{ik}^2 \Big/ \sum_{k=1}^{n} u_{ik}^m \right), K > 0 \qquad ; \text{or} \qquad (2.9a)$$

$$w_i = \sum_{\mathbf{x}_k \in \mathbf{U}_{(i)\alpha}} D_{ik}^2 \Big/ \left|\mathbf{U}_{(i)\alpha}\right| \quad , \qquad (2.9b)$$

where $\mathbf{U}_{(i)\alpha}$ is an α-cut of $\mathbf{U}_{(i)}$, the i-th row of the initializing c-partition for PCM-AO. An initial c-partition of X is required to use (2.9), and this is often taken as the terminal partition from a run of FCM-AO prior to the use of PCM-AO. However, (2.9a) and (2.9b) are not good choices when the data set is noisy. It can be shown (Davé and Krishnapuram, 1997) that the membership function corresponds to the idea of "weight function" in robust statistics and the weights $\{w_i\}$ correspond to the idea of "scale". Therefore, robust statistical methods to estimate scale can be used to estimate the $\{w_i\}$ in noisy situations. Robust clustering methods will be discussed in Section 2.3.

Clustering algorithms produce *partitions*, which are *sets* of n label vectors. For non-crisp partitions, the usual method of defuzzification is the application of (1.15) to each column $\mathbf{U}_k$ of matrix U. The crisp partition corresponding to the *maximum membership* partition of any $U \in M_{pcn}$ is

$$\mathbf{U}_k^{\mathbf{H}} = \mathbf{H}(\mathbf{U}_k) = \mathbf{e}_i \Leftrightarrow u_{ik} > u_{jk} \quad j \neq i; \forall k \quad . \qquad (2.10)$$

The action of $\mathbf{H}$ on U will be denoted by $U^{\mathbf{H}} = [\mathbf{H}(\mathbf{U}_1)\cdots\mathbf{H}(\mathbf{U}_n)]$. The conversion of a probabilistic partition $P \in M_{fcn}$ by Bayes rule (decide $\mathbf{x}_k \in$ class i if and only if $p(i\,|\,\mathbf{x}_k)] \geq p(j\,|\,\mathbf{x}_k)]$for $j \neq i$) results in the crisp partition $P^{\mathbf{H}}$. We call this the *hardening* of U with $\mathbf{H}$.

Example 2.2 HCM-AO, FCM-AO and PCM-AO were applied to the unlabeled data set X_{30} illustrated in Figure 2.2 (the labels in Figure 2.2 correspond to HCM assignments at termination of HCM-AO). The coordinates of these data are listed in the first two columns of Table 2.3. There are c=3 visually compact, well-separated clusters in X_{30}.

The AO algorithms in Table 2.2 were run on X_{30} using Euclidean distance for J and E_t until $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\| \leq \varepsilon = 0.01$. All three algorithms quickly terminated (less than 10 iterations each) using this criterion. HCM-AO and FCM-AO were initialized with the first three vectors in the data. PCM-AO was initialized with the final prototypes given by FCM-AO, and the weight w_i for each PCM-AO cluster was set equal to the value obtained with (2.9a) using the terminal FCM-AO values. The PCM-AO weights were w_1=0.20, w_2=0.21 and w_3 = 1.41. FCM-AO and PCM-AO both used m = 2 for the membership exponent, and all three algorithms fixed the number of clusters at c = 3. Rows $\mathbf{U}_{(i)}$ of U are shown as columns in Table 2.3.

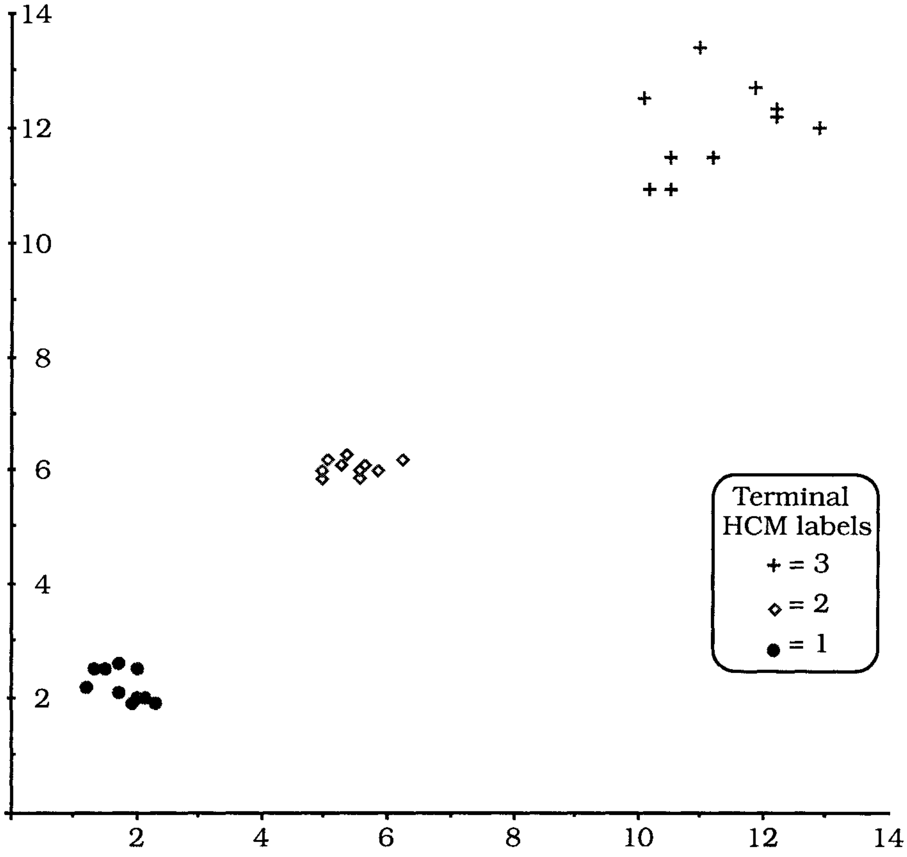


Figure 2.2 Unlabeled data set $\mathbf{X}_{30}$

The terminal HCM-AO partition in Table 2.3 (shaded to visually enhance its crisp memberships) corresponds to visual assessment of the data and its terminal labels appear in Figure 2.2. The cells in Table 2.3 that correspond to maximum memberships in the terminal FCM-AO and PCM-AO partitions are also shaded to help you visually compare these three results.

The clusters in this data are well separated, so FCM-AO produces memberships that are nearly crisp. PCM-AO memberships also indicate well separated clusters, but notice that this is evident not by many memberships being near 1, but rather, by many memberships being near zero.

Table 2.3 Terminal partitions and prototypes for X_{30}

	DATA		HCM-AO			FCM-AO			PCM-AO		
PT.	x_1	x_2	$U^T_{(1)}$	$U^T_{(2)}$	$U^T_{(3)}$	$U^T_{(1)}$	$U^T_{(2)}$	$U^T_{(3)}$	$U^T_{(1)}$	$U^T_{(2)}$	$U^T_{(3)}$
1	1.5	2.5	1.00	0.00	0.00	0.99	0.01	0.00	0.36	0.01	0.01
2	1.7	2.6	1.00	0.00	0.00	0.99	0.01	0.00	0.38	0.01	0.01
3	1.2	2.2	1.00	0.00	0.00	0.99	0.01	0.00	0.27	0.01	0.01
4	2	2	1.00	0.00	0.00	0.99	0.01	0.00	0.94	0.01	0.01
5	1.7	2.1	1.00	0.00	0.00	1	0.00	0.00	0.81	0.01	0.01
6	1.3	2.5	1.00	0.00	0.00	0.99	0.01	0.00	0.26	0.01	0.01
7	2.1	2	1.00	0.00	0.00	0.99	0.01	0.00	0.83	0.01	0.01
8	2.3	1.9	1.00	0.00	0.00	0.98	0.02	0.00	0.52	0.01	0.01
9	2	2.5	1.00	0.00	0.00	0.99	0.01	0.00	0.51	0.01	0.01
10	1.9	1.9	1.00	0.00	0.00	0.99	0.01	0.00	0.86	0.01	0.01
11	5	6.2	0.00	1.00	0.00	0.00	1.00	0.00	0.01	0.57	0.02
12	5.5	6	0.00	1.00	0.00	0.00	1.00	0.00	0.01	0.91	0.02
13	4.9	5.9	0.00	1.00	0.00	0.01	0.99	0.00	0.01	0.46	0.02
14	5.3	6.3	0.00	1.00	0.00	0.00	1.00	0.00	0.01	0.78	0.02
15	4.9	6	0.00	1.00	0.00	0.01	0.99	0.00	0.01	0.48	0.02
16	5.8	6	0.00	1.00	0.00	0.01	0.99	0.00	0.01	0.52	0.02
17	5.5	5.9	0.00	1.00	0.00	0.00	1.00	0.00	0.01	0.82	0.02
18	5.2	6.1	0.00	1.00	0.00	0.00	1.00	0.00	0.01	0.87	0.02
19	6.2	6.2	0.00	1.00	0.00	0.02	0.97	0.01	0.01	0.23	0.02
20	5.6	6.1	0.00	1.00	0.00	0.00	1.00	0.00	0.01	0.79	0.02
21	10.1	12.5	0.00	0.00	1.00	0.01	0.02	0.97	0.00	0.00	0.32
22	11.2	11.5	0.00	0.00	1.00	0.00	0.01	0.99	0.00	0.00	0.63
23	10.5	10.9	0.00	0.00	1.00	0.01	0.04	0.95	0.00	0.00	0.30
24	12.2	12.3	0.00	0.00	1.00	0.00	0.01	0.99	0.00	0.00	0.89
25	10.5	11.5	0.00	0.00	1.00	0.00	0.02	0.98	0.00	0.00	0.40
26	11	14	0.00	0.00	1.00	0.01	0.02	0.97	0.00	0.00	0.40
27	12.2	12.2	0.00	0.00	1.00	0.00	0.02	0.98	0.00	0.00	0.89
28	10.2	10.9	0.00	0.00	1.00	0.01	0.05	0.94	0.00	0.00	0.25
29	11.9	12.7	0.00	0.00	1.00	0.00	0.01	0.99	0.00	0.00	0.84
30	12.9	12	0.00	0.00	1.00	0.01	0.03	0.96	0.00	0.00	0.53
$\bar{v}_1$	$\bar{v}_2$	$\bar{v}_3$	v_1	v_2	v_3	v_1	v_2	v_3	v_1	v_2	v_3
1.77	5.39	11.3	1.77	5.39	11.3	1.77	5.39	11.28	1.92	5.37	11.8
2.22	6.07	12.0	2.22	6.07	12.0	2.22	6.07	12.0	2.08	6.07	12.2

For example, the third cluster has many relatively low maximum memberships, but the other memberships for each of points 21 to 30 in cluster 3 are all zeroes. The greatest difference between fuzzy and possibilistic partitions generated by these two models is that FCM-AO memberships (are forced to) sum to 1 on each data point, whereas PCM-AO is free to assign labels that don't exhibit dependency on points that are not clearly part of a particular cluster. Whether this is an advantage for one algorithm or the other depends on the data in hand. Experiments reported by various authors in the literature support trying both algorithms on the data, and then selecting the output that seems to be most useful.

Because the columns of U in M_{pcn} are independent, PCM actually seeks c independent possibilistic clusters, and therefore it can locate all c clusters at the same spot even when an algorithm such as FCM is used for initialization. In some sense this is the price PCM pays -

losing the ability to distinguish between different clusters - for the advantage of getting possibilistic memberships that can isolate noise and outliers.
The bottom three rows of Table 2.3 enable you to compare the point prototypes produced by the three algorithms to the sample means $\{\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \bar{\mathbf{v}}_3\}$ of the three clusters. HCM-AO and FCM-AO both produce exact (to two decimal places) replicates; PCM-AO cluster centers for the second and third clusters are close to $\bar{\mathbf{v}}_2$ and $\bar{\mathbf{v}}_3$, while the PCM-AO prototype for cluster 1 differs by about 15% in each coordinate. This is because PCM memberships vary considerably within a cluster, depending on how close the points are to the prototype.

Applying (2.10) to the FCM-AO and PCM-AO partitions in Table 2.3 results in the same terminal partition as found by HCM-AO (i.e., the shaded cells in Table 2.3 show that $U^{\mathbf{H}}_{FCM} = U^{\mathbf{H}}_{PCM} = U_{HCM}$). This happens because the data set is small and well-structured. In large, less well structured data sets, the three algorithms may produce partitions that, when hardened, can be significantly different from each other. Needless to say, the utility of a particular output is dependent on the data and problem at hand, and this determination is, unfortunately, largely up to you.

B. Semi-supervised clustering models

Objective functions such as J_1 and J_m that minimize sums of squared errors are well known for their propensity to find solutions that "balance" the number of members in each cluster. This illustrates the sometimes confusing and always frustrating fact that lower values of J_m do NOT necessarily point to better partitions of X.

Semi-supervised c -means clustering models attempt to overcome this limitation. In this category are models due to Pedrycz (1985), Hirota and Iwama (1988) and Bensaid et al. (1996a). They are applicable in domains where users may have a small set of labeled data that can be used to supervise clustering of the remaining data (this is often the case, for example, in medical image segmentation). Algorithms in this category are clustering algorithms that use a finite *design* set $X^d \subset \Re^p$ of *labeled* (crisp or otherwise) data to help clustering algorithms partition a finite set $X^u \subset \Re^p$ of *unlabeled* data. These algorithms terminate *without* the capability to label additional points in $\Re^p$ - that is, they do not build classifier functions. X^d is used to guide FCM-AO to a good c-partition of X^u. Let $X = X^d \cup X^u$, $|X^d| = n_d$, $|X^u| = n_u$, $|X| = n_d + n_u = n$. Without loss we assume that the labeled data are the first n_d points in X,

$$X = \left\{ \underbrace{\mathbf{x}_1^d, \mathbf{x}_2^d, \ldots, \mathbf{x}_{n_d}^d}_{\text{labeled}} \right\} \cup \left\{ \underbrace{\mathbf{x}_1^u, \mathbf{x}_2^u, \ldots, \mathbf{x}_{n_u}^u}_{\text{unlabeled}} \right\} = X^d \cup X^u. \quad (2.11)$$

Pedrycz (1985) defined pointer $b_k = 1$ if $\mathbf{x}_k$ is labeled, and $b_k = 0$ otherwise. Then he defined the matrix $F = [f_{ik}]_{c \times n}$ with the given label vectors in appropriate columns and zero vectors elsewhere. Pedrycz modified J_m at (2.5) to the new functional

$$\tilde{J}_m(U, \mathbf{V}) = \alpha \sum_{i=1}^{c} \sum_{k=1}^{n} (u_{ik} - b_k f_{ik})^m D_{ik}^m + \sum_{i=1}^{c} \sum_{k=1}^{n} (u_{ik})^m D_{ik}^m, \quad (2.12)$$

where $\alpha > 0$ and U in M_{fcn} is a c $\times$n matrix to be found by minimizing (2.12). Under the same assumptions as in Table 2.2, Pedrycz derived first order necessary conditions for $\tilde{J}_m$ by differentiating (2.12) with respect to U and **V** in the usual fashion. The formula for **V** remains (2.7b), while (2.7a) is replaced by the more complicated expression

$$u_{ik} = \frac{1}{1 + \alpha^{1/(m-1)}} \left(\frac{1 + \left[\alpha^{1/(m-1)}\right]\left[1 - b_k \sum_{j=1}^{c} f_{jk}\right] + \alpha^{1/(m-1)} b_k f_{ik}}{\sum_{j=1}^{c} \left(D_{ik}/D_{jk}\right)^{2/(m-1)}} \right). \quad (2.13)$$

Replacing (2.7a) with (2.13) yields the semi-supervised FCM-AO of Pedrycz which we call *ssfcm-AO*. $\tilde{J}_m$ includes a new term whose minimization "forces" U to follow F for the patterns that are already labeled. Weight factor α is used to balance unequal cluster population. Notice especially that U is a new partition of all of X, so at termination the supervising labels are replaced by the computed labels.

The approach to semi-supervision taken by Bensaid et al. (1996a) is to heuristically alter the equations for FCM-AO given in (2.7). Their premise is that the supervising data are labeled correctly, so the n_d labels (crisp or fuzzy) in U^d should be *fixed*. They use AO scheme (2.4c), and take the initial matrix as $U_0 = [U^d | U_0^u]$, where only U_0^u is initialized. The terminal matrix has the form $U_f = [U^d | U_f^u]$.

In ordinary HCM-AO or FCM-AO, once U_0 is determined, the next step is to compute cluster centers $\{\mathbf{v}_{i,0}\}$ using all n columns of U_0. However, since the last n_u columns of U_0 are user-initialized, these authors compute the first set of cluster centers using only the n_d columns in U^d. This is justified by their belief that using only the labeled data to find the initial cluster centers makes them "well-seeded". Consequently, they calculate

$$\mathbf{v}_{i,0} = \sum_{k=1}^{n_d} \left(u_{ik,0}^d\right)^m \mathbf{x}_k^d \Big/ \sum_{k=1}^{n_d} \left(u_{ik,0}^d\right)^m, \ 1 \le i \le c \qquad . \tag{2.14}$$

Next, AO c-means ordinarily calculates U_1 using the $\{\mathbf{v}_{i,0}\}$ to update all n columns of U_0. However, Bensaid et al. use the functional form at (2.7a) and update only the n_u columns in U^u by calculating, for $1 \le i \le c$; $1 \le k \le n_u$,

$$u_{ik,t}^u = \left[\sum_{j=1}^{c} \left(\left\| \mathbf{x}_k^u - \mathbf{v}_{i,t-1} \right\|_A \Big/ \left\| \mathbf{x}_k^u - \mathbf{v}_{j,t-1} \right\|_A \right)^{\frac{2}{m-1}} \right]^{-1}, \ t=1,\ldots,T. \tag{2.15}$$

The cluster centers are then allowed to migrate in feature space, by using *all n columns of U* to subsequently recompute the $\{\mathbf{v}_{i,t}\}$ after the first pass. To counter the possible effect of unequal cluster populations, the few samples that *are* labeled are *weighted* more heavily than their unlabeled counterparts. This is done by introducing non-negative weights $\mathbf{w} = (w_1, w_2, \ldots, w_{n_d})^T$ as follows:

$$\mathbf{v}_{i,t} = \left(\frac{\sum_{k=1}^{n_d} w_k \left(u_{ik,t}^d\right)^m \mathbf{x}_k^d + \sum_{k=1}^{n_u} \left(u_{ik,t}^u\right)^m \mathbf{x}_k^u}{\sum_{k=1}^{n_d} w_k \left(u_{ik,t}^d\right)^m + \sum_{k=1}^{n_u} \left(u_{ik,t}^u\right)^m} \right), \ 1 \le i \le c; \ t=1,\ldots,T \tag{2.16}$$

$\mathbf{x}_k^d$ is replicated w_k times by this weighting scheme. Equations (2.14)-(2.16) comprise the basis of the *semi-supervised FCM* (ssFCM) algorithm of Bensaid et al. (1996a). The major difference between ssFCM and ssfcm-AO is that Pedrycz's scheme is an attempt to solve a new (different than (2.5)) optimization problem, whereas Bensaid et al.'s method is a heuristically defined approach based on (2.5) that is not a true optimization problem (and hence, does not bear the designation AO).

Each point in X^d can be given a different weight in ssFCM. The vector of weights in (2.16) is analogous to the factor α in (2.13): it is chosen by the user to induce the labeled data to drive the clustering algorithm towards a solution that avoids the problem of population balancing that is illustrated in our next example.

Example 2.3 Figure 2.3(a) shows the results of processing a data set given in Bensaid et al. (1996a) called X_{43}, which has c = 2 visually apparent clusters, with FCM-AO. Cluster 1 (X_1) on the left has 40 points, while cluster 2 (X_2) on the right has only 3.

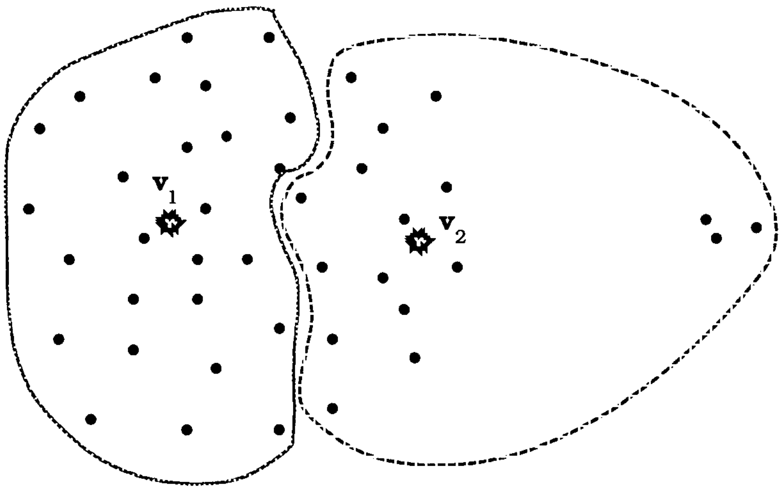


Figure 2.3(a) A hardened FCM-AO partition of X_{43}

Data very similar to these appear on p. 220 of Duda and Hart (1973), where they were used to illustrate the tendency of J_1 to split large clusters. Figure 2.3(a) is essentially the same as Figure 6.13(a) in Duda and Hart, except that our figure is a crisp partition of X_{43} obtained by hardening the terminal partition of a run of FCM-AO. The basic parameters used were the Euclidean norm for both J_2 and E_t, c = m = 2 and ε = 0.0001. The terminal cluster centers are indicated by the symbol (✵). Notice how the large number of points in X_1 draws $\mathbf{v}_2$ far to the left of its visually desirable position. Here, unequal cluster sizes cause J_2 to identify a visually disagreeable solution. This exemplifies our caveat about mathematical models: J_2 prefers this partition to the one we would choose because its measure of similarity and method of aggregation is only a crude and very limited representation of what we do in order to see the obvious structure in the data.

View 2.3(b) shows a partition obtained by hardening the terminal ssFCM (Bensaid et al., 1996a) partition of X_{43} found using the same basic parameters as for FCM-AO. As shown in Figure 2.3(b), ssFCM used four points from X_1 and 1 point from X_2 as the supervising labeled data (so $n_d = 5$), and equal weights $w_i = 6$ for each supervising point. Apparently ssFCM overcomes the problem illustrated in Figure 2.3(a). The three points in X_2 are isolated from the large cluster in the hardened partition, and $\mathbf{v}_2$ occupies a position that is visually correct. However, the supervising points for the left cluster were well chosen in the sense that they, like $\mathbf{v}_1$, occupy the visually apparent center of X_1. Loosely speaking, the weight $w_1 = 6$ essentially gives these four points 6 times as much influence as any unlabeled individual during iteration of ssFCM.

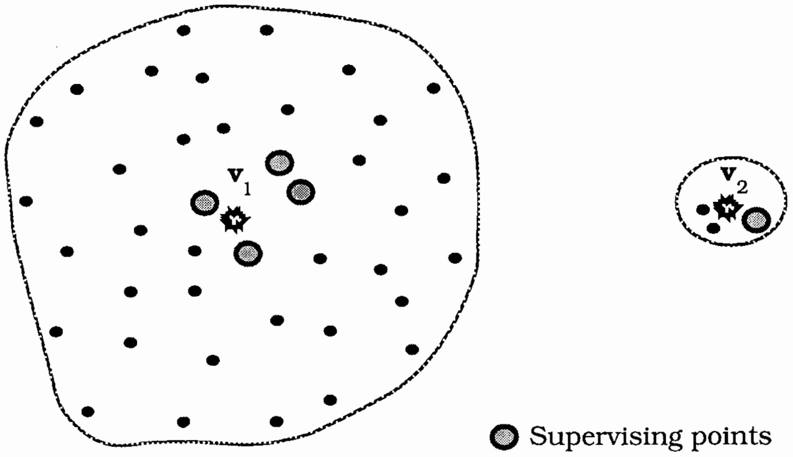


Figure 2.3(b) A hardened ssFCM-AO partition of X_{43}

View 2.3(c) shows a partition obtained by hardening the terminal ssfcm-AO (Pedrycz) partition of X_{43} found using the same basic parameters and supervising points as for ssFCM, with scale factor α in (2.12) set at $\alpha = 200$. Figures 2.3(a) and 2.3(c) are very similar, and based on the difference between views 2.3(b) and 2.3(c), it appears that ssFCM-AO is superior to ssfcm-AO. However, this may be due more to a fortuitous choice of supervising points and weights than by any inherent superiority in ssFCM. Some insight into this aspect of comparing various methods is gained by simply altering the points used for supervision of ssfcm-AO, with all other parameters fixed.

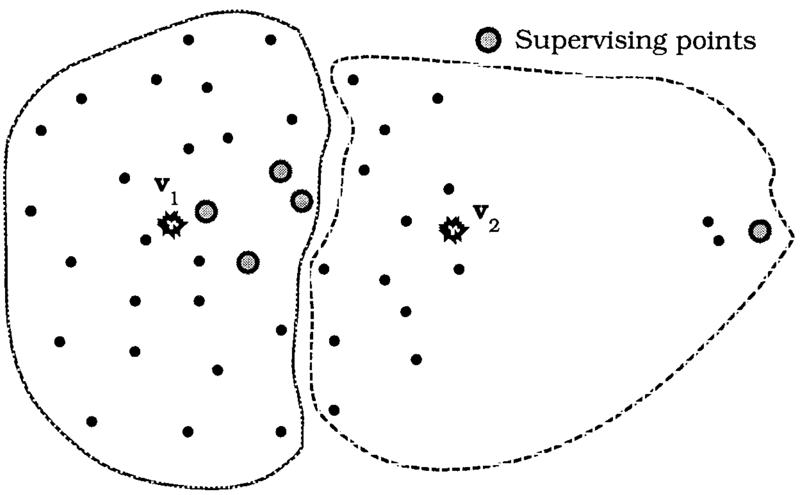


Figure 2.3(c) A hardened ssfcm-AO partition of X_{43}

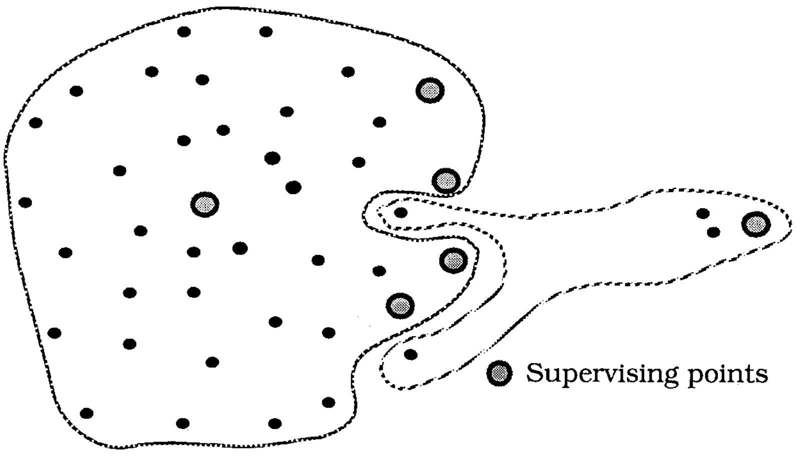


Figure 2.3(d) Another hardened ssfcm-AO partition of X_{43}

Figure 2.3(d) shows the Pedrycz (1985) ssfcm-AO result when the six points shown there are used to supervise it. For this alternate choice of supervision, the algorithm of Pedrycz produces a hardened partition that is quite similar to the one shown in view 2.3(b); only 2 of the 43 points are mislabeled in this crisp partition of the data. We have no doubt that *some* combination of supervising points and algorithmic parameters would enable ssfcm-AO to produce the same partition that ssFCM does here. This emphasizes an important point. Most clustering methods will produce almost identical results if you have enough time to find the parameters that yield the same solution. For p > 3, the luxury of choosing supervising data that are "just right" is not available even when some of the data are labeled. See Bensaid et al. (1996a) for further discussion about this problem.

C. Probabilistic Clustering

Chapter 6 of Duda and Hart (1973) gives a very readable account of the decomposition of normal mixtures using maximum likelihood. This subsection follows their development closely. More complete accounts of probabilistic clustering are found in Everitt and Hand (1981), Titterington et al. (1985), and McLachan and Basford (1988).

The *expectation-maximization* (EM) algorithm is used to optimize the maximum likelihood model of the data, and it generates probabilistic labels for X under the assumptions of statistical mixtures (Titterington et al., 1985). We briefly discuss the connection between this scheme and fuzzy clustering. X is assumed to be drawn from a mixed population of c p-variate statistical distributions that have, for i = 1, 2, ... c: *random variables* $\{\mathbf{X}_i\}$; *prior probabilities* $\{\pi_i\}$; and class-conditional *probability density functions* (PDFs) $\{g(\mathbf{x} \mid i)\}$. The PDF formed by taking the convex combination

$$f(\mathbf{x}) = \sum_{i=1}^{c} \pi_i g(\mathbf{x} \mid i) \tag{2.17}$$

is a distribution which is called a *mixture* of the components $\{\pi_i g(\mathbf{x} \mid i)\}$. Each component $g(\mathbf{x} \mid i)$ can be continuous or discrete, and (2.17) can have some components of either type. The case that dominates applications is when every component of $f(\mathbf{x})$ is *multivariate normal*. Then the PDF of component i has the familiar form

$$n(\mu_i, \Sigma_i) = g(\mathbf{x} \mid i) = e^{-\frac{1}{2}\|\mathbf{x}-\mu_i\|^2_{\Sigma_i^{-1}}} \Big/ (2\pi)^{\frac{p}{2}} \sqrt{\det \Sigma_i} \quad \text{, where} \tag{2.18a}$$

$$\mu_i = (\mu_{1i}, \ldots, \mu_{pi})^T = \text{population } \textit{mean vector} \text{ of class i; and} \tag{2.18b}$$

$$\Sigma_i = \left[\mathrm{cov}(X_i)\right] = \begin{bmatrix} \sigma_{i,11} & \sigma_{i,12} & \cdots & \sigma_{i,1p} \\ \sigma_{i,21} & \sigma_{i,22} & \cdots & \sigma_{i,2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{i,p1} & \sigma_{i,p2} & \cdots & \sigma_{i,pp} \end{bmatrix} . \tag{2.18c}$$

Σ_i is the (positive definite) *population covariance matrix* of class i. $\sigma_{i,jk} = \mathrm{cov}(X_{ij}, X_{ik})$ is the population covariance between variables j and k for class i and the norm in (2.18a) is the Mahalanobis norm for class i (equation (1.9)) computed with the population parameters (μ_i, Σ_i).

Let the *a posteriori probability* that, given **x**, **x** came from class i, be denoted by $\pi(i\,|\,\mathbf{x})$. Bayes rule relates the elements of (2.18) to the probabilities $\{\pi(i\,|\,\mathbf{x})\}$ as follows:

$$\pi(i|\mathbf{x}) = \pi_i g(\mathbf{x}|\,i)/f(\mathbf{x}) \quad . \tag{2.19}$$

For a sample of n points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ assumed to be drawn *independently and identically distributed* (iid) from (2.17) with PDFs as in (2.18), the c × n *posterior matrix* $\Pi = [\pi_{ik} = \pi(i\,|\,\mathbf{x}_k)]$ is in M_{fcn}, the set of constrained c-partitions of X. π_{ik} (or more accurately, estimate p_{ik} of it) is a probability that plays much the same role in statistical clustering that the membership value u_{ik} plays in fuzzy clustering.

The scheme usually employed for mixture decomposition assumes the Gaussian functional form (2.18a) for the $\{g(\mathbf{x}\,|\,i)\}$. Under this hypothesis, we attach an unknown parameter vector $\boldsymbol{\beta}_i$ to each PDF in the mixture:

$$f(\mathbf{x}:\mathbf{B}) = \sum_{i=1}^{c} \pi_i g(\mathbf{x}|\,i;\boldsymbol{\beta}_i) \quad . \tag{2.20}$$

The mixture density now depends on a *parameter vector* $\mathbf{B} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_c)$ where the parameters for class i are $\boldsymbol{\beta}_i = (\pi_i, \mu_i, \Sigma_i)$ for component PDFs as in (2.18a). Alternatively, estimates for posterior matrix Π can be viewed as the parameters of (2.20) since equation (2.19) couples Π to the parameters **B** we use here.

One of the most popular ways to estimate **B** using either labeled or unlabeled data is *maximum likelihood estimation* (MLE). Given $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, we form the log-likelihood function $\ln[L(\mathbf{B}; X)]$ of the samples as a function of **B** and try to maximize it (ln denotes the logarithmic function to the base e). When the component densities are Gaussian, $g(\mathbf{x}|\,i) = n(\mu_i, \Sigma_i)$, first order necessary conditions for the MLE $(p_i, \mathbf{m}_i, S_i)$ of each $\boldsymbol{\beta}_i = (\pi_i, \mu_i, \Sigma_i)$ are known (Wolfe, 1970). Letting $P = [\,p_{ik}]$ denote the MLE of $\Pi = [\pi_{ik}]$, Wolfe showed that

$$p_i = \sum_{k=1}^{n} p_{ik} \Big/ n \quad ; \quad 1 \le i \le c \qquad ; \qquad (2.21a)$$

$$\mathbf{m}_i = \sum_{k=1}^{n} p_{ik}\mathbf{x}_k \Big/ \sum_{k=1}^{n} p_{ik} \quad ; \quad 1 \le i \le c \qquad ; \qquad (2.21b)$$

$$S_i = \sum_{k=1}^{n} p_{ik}(\mathbf{x}_k - \mathbf{m}_i)(\mathbf{x}_k - \mathbf{m}_i)^T \Big/ \sum_{k=1}^{n} p_{ik} \quad ; \quad 1 \le i \le c \text{; and} \qquad (2.21c)$$

$$p_{ik} = \frac{p_i g(\mathbf{x}_k | i; (p_i, \mathbf{m}_i, S_i))}{\sum_{j=1}^{c} p_j g(\mathbf{x}_k | j; (p_j, \mathbf{m}_j, S_j))} \quad ; \quad 1 \le i \le c \quad ; \quad 1 \le k \le n. \qquad (2.22)$$

The vector of means, $\mathbf{M} = (\mathbf{m}_1, \ldots, \mathbf{m}_c)$ at (2.21b) is a set of c point prototypes analogous to the vector $\mathbf{V}$ of cluster centers in the c-means models; and the matrix P at (2.22) is the obvious analog to U for the FCM model. Equations (2.21) are a set of highly non-linear equations that are coupled through Bayes rule (2.22) to the MLE estimators for the population parameters of the c component normal densities. Hence, numerical methods are used to find candidates for local maxima of $\ln[L(\mathbf{B}; X)]$. The AO algorithm given in Table 2.4 that is based on equations (2.21) and (2.22) for normal mixtures is called the *Gaussian mixture decomposition* (GMD-AO) AO algorithm : it is a special case of the EM algorithm.

Table 2.4 The GMD-AO algorithm for normal mixtures

Store	Unlabeled Object Data $X \subset \Re^p$
Pick	☛ number of clusters: $1 < c < n$. Rule of thumb: $c \le \sqrt{n}$ ☛ maximum number of iterations: T ☛ norm for termination: $E_t = \lVert P_t - P_{t-1} \rVert$ = big value ☛ termination criterion: $0 < \varepsilon$ = small value
Guess	☛ initial partition: $P_0 \in M_{fcn}$ (2.4c)
Iterate	$t \leftarrow 0$ REPEAT $t \leftarrow t + 1$ $\mathbf{B}_t = \mathcal{G}_{EM}(P_{t-1})$ where $\mathcal{G}_{EM}(P_{t-1})$ is (2.21a-c) $P_t = \mathfrak{F}_{EM}(\mathbf{B}_t)$ where $P_t = \mathfrak{F}_{EM}(\mathbf{B}_t)$ is (2.22) UNTIL ($t = T$ or $E_t \le \varepsilon$) $(P, \mathbf{B}) \leftarrow (P_t, \mathbf{B}_t)$

When the covariance structure of one or more components is arbitrary, $\ln[L(\mathbf{B}; X)]$ may not have a finite maximum (Duda and Hart, 1973). Nonetheless, numerical solutions of this system are known to produce useful estimators in many real problems, so probabilistic clustering (that is, estimates of the matrix P) is popular. The efficacy of this approach depends mainly on the data satisfying the assumption that it be drawn iid from mixture (2.17) with components (2.18a). If this is not the case (and often, it is not), substructure found by maximizing ln[L(**B**; X)] can be very misleading.

Note that Mahalanobis distances are needed in (2.18a), and therefore in (2.22) as well, so each iteration of GMD-AO requires the inversion of the c $(p\times p)$ covariance matrices at (2.21c). Hence, GMD-AO is quite a bit more computationally intense than any of the c-means algorithms. In view of this, and because the EM algorithm is very sensitive to initialization, many authors prefer to preprocess X with a simpler scheme to improve the chance that GMD-AO gets started at a point in its parameter space that is close to a useful solution (Duda and Hart, 1973).

It has been shown that FCM-AO can provide good initializations for GMD-AO (Bezdek and Dunn, 1975, Bezdek et al., 1985, Davenport et al., 1988). Gath and Geva (1989b) also studied the estimation of components of normal mixtures using a fuzzy clustering approach that we discuss in Section 2.3. These papers point out the similarities of and differences between estimates produced by FCM-AO and GMD-AO algorithms. See Hathaway and Bezdek (1986b) for an example that proves that the point prototypes $\{\mathbf{v}_i\}$ from FCM-AO cannot be a statistically consistent estimator of the means $\{\mathbf{m}_i\}$ from an arbitrary univariate mixture. Another approach to initialization by clustering with a deterministic competitive learning model is given by McKenzie and Alder (1994). Many fuzzy clustering algorithms have been proposed that are either hybrids of or are related to GMD-AO, or that use the maximum likelihood principle in some other way. Among these, we mention the paper by Kaufman and Rouseeuw (1990), and we discuss the model of Gath and Geva (1989a) in Section 2.4.

Example 2.4 Data set X_{30} from Table 2.3 was processed with the GMD-AO algorithm using the same initialization and termination parameters as in Example 2.2. Specifically, $\mathbf{v}_{1,0} = (1.5, 2.5)^T$, $\mathbf{v}_{2,0} = (1.7, 2.6)^T$ and $\mathbf{v}_{30} = (1.2, 2.2)^T$. The results are shown in Table 2. 5. Since GMD-AO produces a partition in M_{fcn}, we compare $P_{GMD\text{-}AO}$ to the terminal U_{FCM} produced by FCM-AO for m=2 that appears in the third set of columns in Table 2.3. The hardened versions of both partitions are also identical, $P^{\mathbf{H}}_{GMD} = U^{\mathbf{H}}_{FCM} = U_{HCM}$.

Table 2.5 Terminal EM and FCM partitions and prototypes for X_{30}

Pt.	Data Pt.		GMD-AO			FCM-AO (m=2)		
	x_1	x_2	$\mathbf{P}^T_{(1)}$	$\mathbf{P}^T_{(2)}$	$\mathbf{P}^T_{(3)}$	$\mathbf{U}^T_{(1)}$	$\mathbf{U}^T_{(2)}$	$\mathbf{U}^T_{(3)}$
1	1.5	2.5	1.00	0.00	0.00	0.99	0.01	0.00
2	1.7	2.6	1.00	0.00	0.00	0.99	0.01	0.00
3	1.2	2.2	1.00	0.00	0.00	0.99	0.01	0.00
4	2.0	2.0	1.00	0.00	0.00	0.99	0.01	0.00
5	1.7	2.1	1.00	0.00	0.00	1.00	0.00	0.00
6	1.3	2.5	1.00	0.00	0.00	0.99	0.01	0.00
7	2.1	2.0	1.00	0.00	0.00	0.99	0.01	0.00
8	2.3	1.9	1.00	0.00	0.00	0.98	0.02	0.00
9	2.0	2.5	1.00	0.00	0.00	0.99	0.01	0.00
10	1.9	1.9	1.00	0.00	0.00	0.99	0.01	0.00
11	5.0	6.2	0.00	1.00	0.00	0.00	1.00	0.00
12	5.5	6.0	0.00	1.00	0.00	0.00	1.00	0.00
13	4.9	5.9	0.00	1.00	0.00	0.01	0.99	0.00
14	5.3	6.3	0.00	1.00	0.00	0.00	1.00	0.00
15	4.9	6.0	0.00	1.00	0.00	0.01	0.99	0.00
16	5.8	6.0	0.00	1.00	0.00	0.01	0.99	0.00
17	5.5	5.9	0.00	1.00	0.00	0.00	1.00	0.00
18	5.2	6.1	0.00	1.00	0.00	0.00	1.00	0.00
19	6.2	6.2	0.00	1.00	0.00	0.02	0.97	0.01
20	5.6	6.1	0.00	1.00	0.00	0.00	1.00	0.00
21	10.1	12.5	0.00	0.00	1.00	0.01	0.02	0.97
22	11.2	11.5	0.00	0.00	1.00	0.00	0.01	0.99
23	10.5	10.9	0.00	0.00	1.00	0.01	0.04	0.95
24	12.2	12.3	0.00	0.00	1.00	0.00	0.01	0.99
25	10.5	11.5	0.00	0.00	1.00	0.00	0.02	0.98
26	11.0	14.0	0.00	0.00	1.00	0.01	0.02	0.97
27	12.2	12.2	0.00	0.00	1.00	0.00	0.02	0.98
28	10.2	10.9	0.00	0.00	1.00	0.01	0.05	0.94
29	11.9	12.7	0.00	0.00	1.00	0.00	0.01	0.99
30	12.9	12.0	0.00	0.00	1.00	0.01	0.03	0.96
$\bar{\mathbf{v}}_1$	$\bar{\mathbf{v}}_2$	$\bar{\mathbf{v}}_3$	$\mathbf{m}_1$	$\mathbf{m}_2$	$\mathbf{m}_3$	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$
1.77	5.39	11.3	1.77	5.39	11.27	1.77	5.39	11.28
2.22	6.07	12.0	2.22	6.07	11.99	2.22	6.07	12.00

To two significant digits, the terminal GMD-AO partition is crisp before hardening, $P_{GMD} = U_{HCM}$. This is not unexpected in view of the well separated, Gaussian-like clusters in X_{30}. A similar very crisp result can be obtained by setting the FCM parameter m to a value close to 1. The terminal estimates $\{\mathbf{m}_i\}$ of the (assumed normal) means are practically identical to the FCM cluster centers $\{\mathbf{v}_i\}$ and the sample means $\{\bar{\mathbf{v}}_i\}$. Also available at termination of GMD-AO are the covariance matrices,

$$S_1 = \begin{bmatrix} 0.11 & -.05 \\ -0.05 & 0.07 \end{bmatrix}; S_2 = \begin{bmatrix} 0.16 & 0.01 \\ 0.01 & 0.02 \end{bmatrix}; \text{and } S_3 = \begin{bmatrix} 0.84 & 0.24 \\ 0.24 & 0.57 \end{bmatrix};$$

and estimates of the prior probabilities, which were all $p_i = 1/3$ to six significant digits. The terminal covariance matrices are useful for analysis of the shape of each cluster - this will become evident in Section 2.3. To summarize, when the clusters are well separated and essentially spherical and well distributed throughout their basic domains as they are in X_{30}, the mixture decomposition and fuzzy c-means models produce almost identical results. This is, of course, not always the case.

D. Remarks on HCM/FCM/PCM

The remaining paragraphs of this section offer comments on various aspects of the point prototype c-means models. Generalizations based on prototypes that are not points in feature space is so large and important that we will devote Section 2.3 to this topic.

Choice of parameters. HCM-AO, FCM-AO and PCM-AO share the algorithmic parameters (U_0 /$\mathbf{V}_0$, T, $\|*\|_A$, $\|*\|_{err}$, c and ε). FCM-AO adds weighting exponent m, and PCM-AO adds m and the weight vector **w** to this list. Variation in *any* of these parameters can affect algorithmic results for a fixed set of unlabeled data. Choosing the most basic parameter, c = the number of clusters, will be discussed in Section 2.4.

Justifying your choice of m is always a challenge. FCM-AO will produce partitions that approach $\overline{U} = [1/c]$ as m increases. In theory, this happens as $m \to \infty$, but in practice terminal partitions usually have memberships close to (1/c) for values of m between 10 and 20. At the other extreme, as m approaches 1 from above, FCM reduces to HCM, and terminal partitions become more and more crisp. Thus, m controls the degree of fuzziness exhibited by the soft boundaries in U. Most users choose m in the range [1.1, 5], with m = 2 an overwhelming favorite. See Bezdek (1976) for an electrical analog of J_m that offers a physical explanation for preferring m = 2. Macbratney and Moore (1985) discuss an empirical scheme for choosing m.

Choice of norms. Many authors have used distances (1.7), (1.8) and (1.9) for D_{ik} with J_m. The results do not indicate a clear advantage for any distance, nor should they. The data themselves have the last say about which distance will provide the best results. Euclidean distance is the overwhelming favorite, probably because it is the one we live with. Mahalanobis distance is useful when there are large disparities in the ranges of the measured features because it rotates the basis of $\Re^p$ so that the data are scaled equally and are pairwise uncorrelated.

Differentiability of the inner product norm leads to the conditions for **V** shown in Table 2.1. When D_{ik} in (2.5) is replaced by a Minkowski norm other than the 2-norm, this property fails, and other means are necessary for finding the function $\mathcal{G}_c$ that is needed for $\mathbf{V}_t = \mathcal{G}_c(U_t)$. Authors who have studied the use of the city block (1- norm) and sup norm (∞ norm) for J_m include Bobrowski and Bezdek (1991), Jajuga (1991) and Kersten (1995). $\mathcal{G}_c$ in Table 2.2 is defined by a method such as linear programming or Newton's method when these norms are used. Kersten (1995) has shown that $\mathcal{G}_c$ for the 1-norm is the weighted (or fuzzy) median. Trauwaert (1987) also discusses several issues and algorithms related to the use of the 1-norm in fuzzy clustering.

Initialization. Initialization is important. Many examples have been published that show termination at different solutions when c-means AO algorithms are initialized from different starting points. There is no general agreement about a good initialization scheme. The three most popular methods are: (i) using the first c distinct points in the data; (ii) using c points randomly drawn from a hyperbox containing X; and (iii) using c points uniformly distributed along the diagonal of the hyperbox containing X. Some authors recommend initializing FCM with the output of HCM, and in turn, some initialize PCM with FCM. Be careful not to initialize any of these algorithms with equal rows in U_0 or equal cluster centers in $\mathbf{V}_0$, because iterative updates cannot subsequently change them unless there is a benevolent roundoff error that upsets equality.

Termination. The tradeoff between speed and accuracy is affected by both E_t and ε. The 1, 2 and sup norms have all been used for E_t. The most popular norm for E_t is certainly the Euclidean norm, but the 1-norm probably provides comparable results at a savings in time. The choice of ε controls the duration of iteration as well as the quality of terminal estimates. If ε is too small, limit cycles may occur. Most authors report good success with ε in the interval [0.01, 0.0001].

The choice of initialization in **V** as in (2.4b) or U as in (2.4c) is almost a matter of taste. However, there is a large and important difference in terms of storage and speed. When initialization and termination are made on U as in (2.4c), (cn) variables must all be close before termination occurs, and two $c \times n$ matrices must be stored. On the other hand, only (cp) variables must have small successive differences when initializing and terminating with **V**, and storage for successive iterates of **V** requires only two $c \times p$ matrices. To see the difference, suppose you apply c-means to a 256×256 monochromatic image. Then n=65,536 whereas p=1. Since c is common to both schemes, initializing and terminating on **V** realizes an $O(10^5)$ savings in storage and usually reduces CPU (*central processing unit*) time considerably. On the other hand, ε will usually have to be smaller to achieve termination at estimates comparable to using U for initialization and termination instead of **V**. Some authors normalize the termination error by the number of variables being estimated, comparing ε to either

$$\frac{\|U_t - U_{t-1}\|}{cn} \text{ or } \frac{\|\mathbf{V}_t - \mathbf{V}_{t-1}\|}{cp}.$$

Convergence. Convergence always means in some well-defined mathematical sense; termination is when and where an algorithm stops. Sequences converge, algorithms terminate - hopefully, close to a point of convergence. FCM-AO generates an iterate sequence that contains a subsequence that converges q-linearly from any initialization to a local minimum or saddle point (local maxima are impossible) of J_m. Convergence theory for FCM (and HCM as well) began with Bezdek (1980), and has progressed through a series of papers that include Selim and Ismail (1984), Ismail and Selim (1986), Hathaway and Bezdek (1986a, b), Sabin (1987), Bezdek et al. (1987b), Hathaway and Bezdek (1988), Kim et al. (1988) and Wei and Mendel (1994).

Acceleration. In this category are methods that seek to improve computational properties (speed and storage) of the c-means AO algorithms. The algorithms in Table 2.2 require many distance calculations as well as fractional exponentiation at each half iterate. For large data sets (e.g., images), this can mean relatively slow iteration and lots of storage. Two methods for acceleration are used: exploitation of special properties of the data; and alteration the equations used for iteration.

As an example of exploiting special properties of the data, consider the case when X is an 8 bit image. Since there are only 256 possible values for each x_k, a great reduction in both memory and CPU time can be realized by using the frequency of occurrence of each gray level as follows. Let the number of pixels with gray level q be f_q. All of these pixels will have the same membership in all c clusters. Let u_{iq} be the membership of all pixels with gray level q for cluster i, $1 \le i \le c$. There are 256 values of u_{iq}, one for each gray level.

Now consider the computation of, say, the i-th cluster center for FCM with equation (2.7b) when the image has 256×256 = 65,536 pixels:

$$v_i = \sum_{k=1}^{65536} u_{ik}^m x_k \Big/ \sum_{k=1}^{65536} u_{ik}^m = \sum_{q=0}^{255} f_q u_{iq}^m x_k \Big/ \sum_{q=0}^{255} f_q u_{iq}^m \ ,$$

The second form on the right follows because all of the pixels (exactly f_q of them) with gray level q will have the identical membership u_{iq}. This means that only 256 membership vectors in $\Re^c$ need to be stored and used, as opposed to 65,536. Histogramming easily obtains the $\{f_q\}$, and equations (2.6) - (2.8) for *all* the c-means models can be implemented much more efficiently.

The *approximate FCM* (AFCM-AO) algorithm of Cannon et al. (1986a) was the first technique studied to speed up FCM-AO by changing equations (2.7), and it also made use of the special nature of image data. AFCM-AO reduced the CPU time for FCM-AO by replacing the exact necessary conditions at (2.7) with approximate ones that could be implemented using six look-up tables. However, AFCM-AO was restricted to discrete data (such as image data), and contained several approximations that degraded output quality. AFCM-AO saved roughly an order of magnitude in time, but has been largely overshadowed by more recent developments such as those given by Kamel and Selim (1994), Shankar and Pal (1994), Cheng et al. (1995) and Hershfinkel and Dinstein (1996).

E. The Reformulation Theorem

Minimization of the c-means functionals can be attempted in many ways besides AO. Hathaway and Bezdek (1995) show that problem (2.5) can be *reformulated* by eliminating either U or **V** from J_m by direct substitution of the necessary conditions for one or the other from Table 2.1 into equation (2.5). This idea had its roots in Bezdek (1976), where the reformulation of J_m for FCM was exhibited, but the effect of using it to replace the original optimization problem was not discussed. The reformulations of J as a function of **V** alone for the three cases are:

$$R_1(\mathbf{V};\mathbf{0}) = \sum_{k=1}^{n} \min\{D_{1k}, D_{2k}, \ldots, D_{ck}\} \quad ; \qquad (2.23a)$$

$$R_m(\mathbf{V};\mathbf{0}) = \sum_{k=1}^{n} \left(\sum_{i=1}^{c} D_{ik}^{1/(1-m)} \right)^{1-m} \quad ; \qquad (2.23b)$$

$$R_m(\mathbf{V};\mathbf{w}) = \sum_{i=1}^{c} \sum_{k=1}^{n} \left(D_{ik}^{1/(1-m)} + w_i^{1/(1-m)} \right)^{1-m} \quad . \qquad (2.23c)$$

Let J denote a particular instance of (2.5) (hard, fuzzy or possibilistic) and R denote the corresponding reformulated version in (2.23). Let $\mathbf{M}$ be M_{hcn}, M_{fcn}, or M_{pcn}, and $U = \mathfrak{F}(\mathbf{V})$ denote the function of $\mathbf{V}$ defined by the right hand side of (2.6a), (2.7a) or (2.8a), depending on the model used. Let the distances D_{ik}, for i=1,...,c and k=1,...,n, be continuous functions of $\mathbf{V} \in \mathbf{V}$, where $\mathbf{V}$ is an open subset of $\Re^{ct}$ and $\mathbf{V} \in \Re^{ct}$. (Using t instead of p admits more general prototypes than points in $\Re^p$; the theorem holds for many non-point prototype cases which are discussed later.) Let $\mathbf{V}$ be such that the corresponding distances satisfy $D_{ik} > 0$, for i=1,...,c and k=1,...,n. Then:

For the hard, fuzzy or possibilistic cases:

(i) (U, $\mathbf{V}$) globally minimizes J over $\mathbf{M} \times \mathbf{V} \Rightarrow \mathbf{V}$ globally minimizes R over $\mathbf{V}$; and

(ii) $\mathbf{V}$ globally minimizes R over $\mathbf{V} \Rightarrow (\mathfrak{F}(\mathbf{V}), \mathbf{V})$ globally minimizes J over $\mathbf{M} \times \mathbf{V}$.

For either the fuzzy or possibilistic case:

(iii) (U, $\mathbf{V}$) locally minimizes J $\Rightarrow \mathbf{V}$ locally minimizes R; and

(iv) $\mathbf{V}$ locally minimizes R $\Rightarrow (\mathfrak{F}(\mathbf{V}), \mathbf{V})$ locally minimizes J.

This theorem can be used to convert problem (2.5) into an equivalent unconstrained optimization problem where R in (2.23) is minimized with *any* optimization scheme. Without the reformulation theorem, there is no assurance that AO and R - based solutions will even be similar, much less the same. This theorem guarantees that while approaches based on reformulation will undoubtedly have different computational properties (such as speed and storage), they will not produce markedly different clustering solutions from AO-based solutions.

For example, consider optimization of J_m using a *genetic algorithm* (GA) approach (Goldberg, 1989). One of the most important issues for using GA is representation of the model in a form that is amenable to the GA paradigm. J_m in (2.5) is not well suited to be a fitness function for this type of optimization because of the constraints on the $\{u_{ik}\}$. Reformulations of J_m in terms of $\mathbf{V}$ alone are much more likely to be good fitness functions, because the number of parameters that must be represented and estimated in the reformulation is far less than in the original form. For example, consider segmenting an unlabeled magnetic resonance image of size 256×256 in its standard three dimensional parameter space. For c=

10 tissue classes, the number of unknowns sought by FCM-AO is c(n+p)= 10(65,536+3) = 655,363; but the reformulated version, FCM-R, seeks only cp = 10(3) = 30. Moreover, (2.5) demands maintaining 65,536 constraints on the memberships, which makes the use of GA on J_m quite impractical.

Several authors have experimented with the GA approach to solving (2.5) via (2.23). The reformulation theorem gives this idea a solid footing. Desired (but not guaranteed) advantages include elimination of dependency on good initialization as well as avoidance of local trap states. For small problems (that is, small values of n and p), the GA approach seems to deliver good solutions. As the size of the data set increases, however, it is less clear that optimization of J_m by GA methods is an improvement to AO schemes. This is a fairly new area: representative papers include Hathaway and Bezdek (1994a) and Hall et al. (1994).

2.3 Non point-prototype clustering models

An important aspect of clustering with the EM method is that the eigenstructure of the covariance matrices $\{S_i\}$ at (2.21c) lets clusters assume locally different hyperellipsoidal shapes. This enables the GMD-AO algorithm to represent each cluster drawn from a normal mixture more accurately than the c-means models. When using the norm $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|_A^2$ in (2.5), only one cluster shape can be matched well - the hyperellipsoidal shape determined by the eigenstructure of the fixed weight matrix A. This is fine if all c clusters have that shape, but most real data have more variability than this. The deficiency of global minima for least squared error functions that is illustrated by Figure 2.3(a) is due at least in part to exactly this limitation, and ssFCM can be viewed as an attempt to trick the FCM functional in hopes of overcoming this problem. Knowledge of this deficiency may have been one of the reasons that Gustafson and Kessel (1979) introduced the model we discuss next.

The basic variables that can be altered in the c-means models are the way proximities $\{D_{ik}\}$ of $\mathbf{x}_k$ to the point prototypes $\{\mathbf{v}_i\}$ are measured; and the kind of prototypes that are used. Geometric shapes in clusters can be matched either by adjusting the norm (and hence, the shape of open and closed unit balls in feature space), or by changing the fitting prototypes $\mathbf{V}$. There have been many studies on the effect of changing A in $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|_A^2$ on the assumption that *all* of the clusters in X have roughly the same A-norm geometry and that they are "cloud-like" - that is, they are more or less uniformly distributed over their convex hulls. Gustafson and Kessel (1979) introduced the first fuzzy method for *localized* shape matching via individual norms that adapted to the shapes of individual clusters. Algorithms of this kind are called *adaptive* because individual norms change at each iteration in an attempt to adapt to the geometry of individual clusters.

Some authors prefer to interpret the weight matrix A inducing an individual norm for a cluster as part of the prototype of the cluster. This leads to the idea of choosing *non-point prototypes* to match the shape of the expected clusters. A linear prototype, rather than a point, could be used to represent line-like clusters, and a circular prototype could be used to find ring-shaped clusters. Clusters which have no "interior points" are called *shell* clusters to distinguish them from cloud type or volumetric structures. Figure 2.4 illustrates this idea for three two-dimensional structures that are best represented by different kinds of prototypes. The volumetric cloud on the left is represented by a point prototype, while the linear and circular shells in the other two views are best represented by more general (non-point) prototypes, namely, a line and a circle.

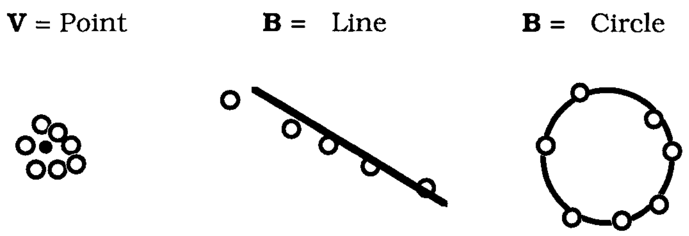


Figure 2.4 Appropriate prototypes for various clusters

Sets of c point prototypes are called **V**, and the collection of c non-point prototypes will be denoted by **B**. Sometimes, but not always, a model admits **V** as a special case of **B**. Non-point prototype-based clusters are usually defined by least-squared error models that attempt to fit the prototypes to the clusters. The generalization of problem (2.5) for non-point prototypes is

$$\underset{(U,\mathbf{B})}{\min}\left\{J_m(U,\mathbf{B};\mathbf{w})=\sum_{i=1}^{c}\sum_{k=1}^{n}u_{ik}^m D_{ik}^2+\sum_{i=1}^{c}w_i\sum_{k=1}^{n}\left(1-u_{ik}\right)^m\right\}, \text{ where} \tag{2.24a}$$

$$\mathbf{B}=(\beta_1,\beta_2,\ldots,\beta_c);\ \beta_i \text{ is the i-th non-point prototype; and} \tag{2.24b}$$

$$D_{ik}^2=S(\mathbf{x}_k,\beta_i) \text{ measures the proximity or similarity of } \mathbf{x}_k \text{ to the i-th non-point prototype.} \tag{2.24c}$$

The exact nature of β_i (and hence S) depends on the particular model. A variety of models and AO algorithms have been developed by varying β_i and S. In AO algorithms the membership update equation $U_t = \mathcal{F}_C(\mathbf{B}_{t-1})$ is still given by (2.6a), (2.7a) and (2.8a) for the hard, fuzzy and possibilistic cases respectively, except that D_{ik} now represents an appropriate proximity measure rather than $\|\mathbf{x}_k - \mathbf{v}_i\|_A$. The prototype update equation $\mathbf{B}_t = \mathcal{G}_{\mathcal{C}}(U_t)$ depends on the particular choice of the prototype. In this section we discuss several non-point prototype models.

A. The Gustafson-Kessel (GK) Model

Gustafson and Kessel (1979) proposed that the matrix A in equation (2.5) be a third variable. They put $\mathbf{A} = (A_1, \ldots, A_c)$, A_i being a positive-definite $p \times p$ matrix, and modified (2.5) to

$$\min_{\substack{\underbrace{(U,\,\mathbf{V},\mathbf{A})} \\ \det(A_i)=\rho_i}} \left\{ J_{m,GK}(U,\mathbf{V},\mathbf{A}) = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^m \|\mathbf{x}_k - \mathbf{v}_i\|_{A_i}^2 \right\} \quad . \tag{2.25}$$

The variables estimated by the GK model are the triplet (U, **V**, **A**) where **V** is still a vector of point prototypes. This model predates possibilistic partitions by some 15 years, so the weights $\{w_i\}$ in (2.5) are all zero. The important idea here is that the i-th cluster in U might be best matched by a hyperellipsoidal shape generated by the eigenstructure of the variable matrix A_i, much like S_i does for GMD-AO at (2.21c). The additional constraint that $\det(A_i) = \rho_i > 0$ guarantees that A_i is positive-definite; ρ_i is a user defined constant for each cluster. Gustafson and Kessel showed that minimization of $J_{m,GK}$ with respect to A_i leads to the necessary condition

$$A_i = \left[\rho_i \det(C_i)\right]^{1/p} C_i^{-1},\ 1 \le i \le c \quad . \tag{2.26}$$

In (2.26) C_i is the *fuzzy covariance matrix* of cluster i,

$$C_i = \sum_{k=1}^{n} u_{ik}^m (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^T \Big/ \sum_{k=1}^{n} u_{ik}^m,\ 1 \le i \le c;\ m \ge 1, \tag{2.27}$$

where $\mathbf{v}_i$ is the i-th point prototype or cluster center. In the sequel we may represent the set of fuzzy covariance matrices calculated with (2.27) as the vector $\mathbf{C} = (C_1, \ldots, C_c) \in \Re^{c(p \times p)}$. Gustafson and Kessel used $\rho_i = 1$ for all i. With this choice, the fixed norm $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|_A^2$ used for the c distances from $\mathbf{x}_k$ to the current $\{\mathbf{v}_i\}$ during calculation of (2.7a) is replaced in the GK-AO algorithm with the c distances

$$D^2_{ik,GK} = \det(C_i)^{1/p}\|\mathbf{x}_k - \mathbf{v}_i\|^2_{C_i^{-1}}, \ 1 \le i \le c \quad . \tag{2.28}$$

When crisp covariance matrices are used, the distance measure in (2.28) is the one suggested by Sebestyen (1962). This distance measure was also used by Diday (1971), Diday et al. (1974) and Diday and Simon (1976) in their *adaptive distance dynamic clusters* (ADDC) algorithm. Thus, the GK-AO algorithm can be viewed as the fuzzification of ADDC, and may be regarded as the first (locally) adaptive fuzzy clustering algorithm.

For AO optimization of $J_{m,GK}(U, \mathbf{V}, \mathbf{A})$, the partition U and centers $\{\mathbf{v}_i\}$ are updated using (2.7a, b) as in FCM-AO, and the covariance matrices are updated with (2.26). The GK-AO algorithm is more sensitive to initialization than FCM-AO because its search space is much larger. Typically, FCM-AO is used to provide a reasonably good initialization for this algorithm. Experimental evidence indicates that $1 \le m \le 2$ gives good results, with m = 1.5 often being the recommended value.

Example 2.5 Figure 2.5 shows two data sets that were processed with the FCM-AO, GK-AO, and the GMD-AO algorithms. The parameter m was set at 2.0 for FCM-AO, and 1.5 for the GK-AO algorithm. All runs were initialized with the first two points in the left views ($\mathbf{v}_{1,0} = (30, 35)^T$, $\mathbf{v}_{2,0} = (42, 45)^T$) or first three points in the right views ($\mathbf{v}_{1,0} = (21, 104)^T$, $\mathbf{v}_{2,0} = (22, 101)^T$ and $\mathbf{v}_{3,0} = (22, 104)^T$). The termination criterion was $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\| \le \varepsilon = 0.001$. The Euclidean norm was used for J_2.

The left side of Figure 2.5 contains points drawn from a mixture of c = 2 fairly circular Gaussians. The clusters on the right are drawn from a mixture of c = 3 bivariate normals, one of which (in the upper right portion of each view) has a covariance structure that tends toward linear correlation between x and y. The three clusters on the right exhibit visually different shapes, so we expect GK-AO and GMD-AO to use their localized adaptivity to find these clouds more accurately than FCM-AO, which has a fixed norm-inducing weight matrix.

Terminal partitions hardened with (2.10) are shown in Figure 2.5 by assigning different symbols to the crisp clusters. The shaded areas in views a, b, d, e and f correspond to the points that, when compared with the labels of the samples drawn, were labeled incorrectly. For the data on the left, FCM-AO tends to divide the data into circular (because the norm is Euclidean) clusters of roughly equal size (the problem illustrated in Figure 2.3(a)). The GK-AO result in view 2.5(b) shows an even stronger encroachment of the right cluster into the left one. View 2.5(c) shows that GMD-AO labels every point in the 2 clusters data correctly; it reproduces the a priori labels flawlessly.

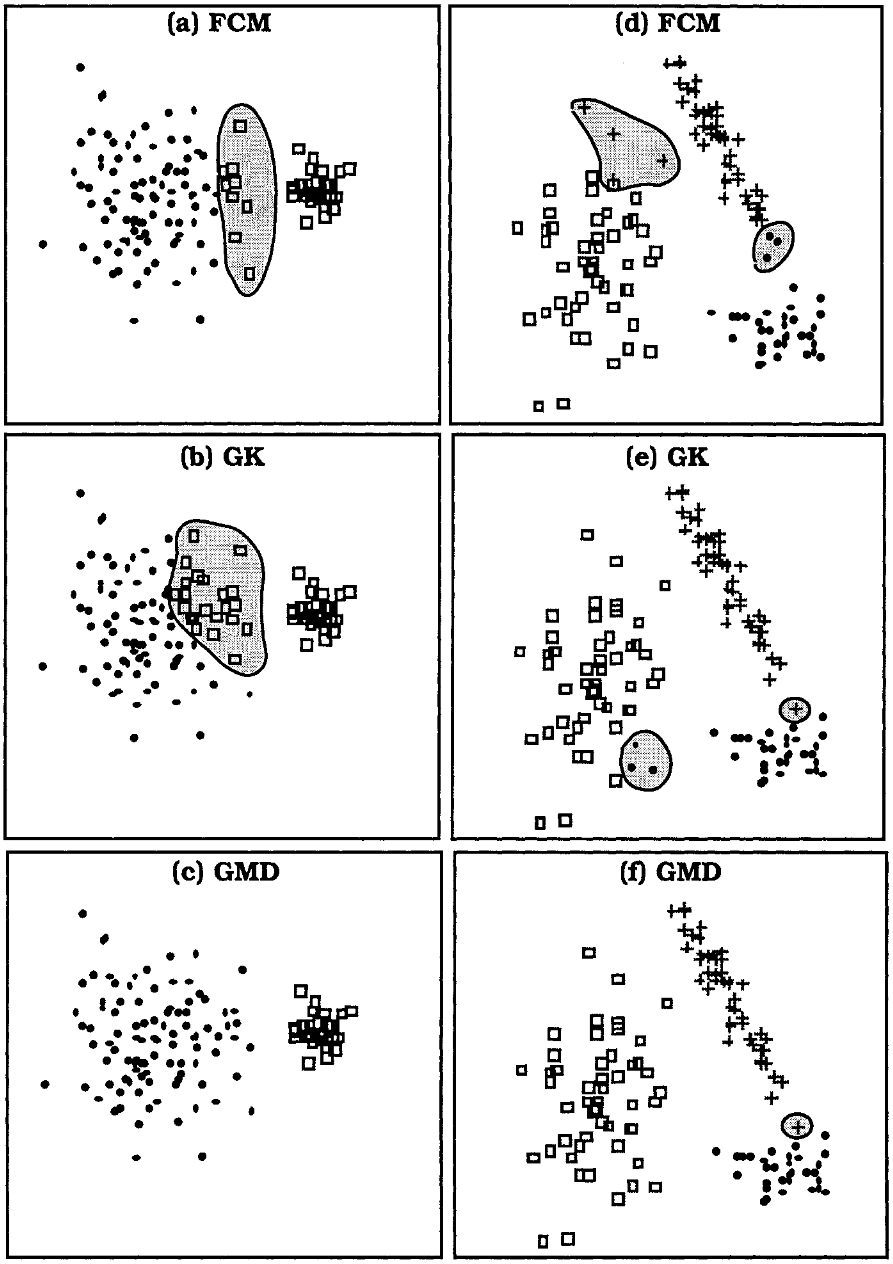


Figure 2.5 Hardened partitions for 2 sets of Gaussian clusters

The GMD-AO model also produces visually better results with the three clusters on the right side of Figure 2.5. Here FCM-AO draws four points up towards the linear cluster from the centrally located large group, and loses three points from the linear cluster to the lower cluster on the right (7 mistakes). GK-AO draws three points to the left from the lower right hand cluster and loses one point to the linear cluster (4 mistakes). GMD-AO reproduces the a priori labels almost flawlessly, missing just one point in the bottom right cluster to the linear cluster (1 mistake).

Visually, GMD gives much better results than FCM or GK for both data sets in Figure 2.5. Is GMD *generally* superior? Well, if the data are really from a mixture of normals, the GMD model matches the data better than the other two models. But if the geometry of the data does not fit the pattern expected for draws from a mixture of normals very well, GMD does not produce better results than other models. Moreover, (2.28) reduces to the Euclidean distance when $C_i = \sigma_i^2 I$. If this is true for all i = 1,...,c, the behavior of GK and FCM are very similar.

Bezdek and Dunn (1975) studied the efficacy of replacing GMD-AO parameter (P, **M**) with terminal (U, **V**)'s from FCM, and then calculating the remaining MLE of components (the priors and covariance matrices) of normal mixtures non-iteratively. Hathaway and Bezdek (1986b) proved that this strategy could not produce correct MLEs for (P, **M**) in even the univariate case.

Gath and Geva (1989a) discuss an algorithm they called *fuzzy maximum likelihood estimation* (FMLE). Specifically, they used the fuzzy covariance matrix C_i at (2.27) <u>with m = 1</u> (this does *not* mean or require that the partition matrix U is crisp) to define an *exponential distance* $D^2_{ik,GG} = \left(\frac{\det(C_i)^{1/2}}{p_i}\right) e^{\frac{1}{2}\|\mathbf{x}_k - \mathbf{v}_i\|^2_{C_i^{-1}}}$, where p_i is the estimate of the prior probability of class i shown in (2.21a). This distance was then used in FCM formula (2.7a) with m = 2, resulting in the memberships $\left\{\tilde{u}_{ik} = \left[\sum_{j=1}^{c}\left(D_{ik,GG}/D_{ij,GG}\right)^2\right]^{-1}\right\}$, which were taken as estimates for the posterior probabilities in equation (2.22). It is not hard to verify that this with updating scheme $\tilde{u}_{ik}$ is identical to p_{ik} in (2.22). It is not hard to show that the update equations for FMLE are identical to those for GMD-AO. Thus, FMLE is essentially equivalent to GMD-AO with the $\{p_{ik}\}$ interpreted as fuzzy memberships. We will illustrate FMLE in Section 2.4 in conjunction with several measures of cluster validity defined in Gath and Geva (1989a).

Although the GK algorithm was developed for, and both it and GMD-AO are used to detect ellipsoidal clusters, since lines and planes can be viewed as extremely elongated or flat ellipsoids, these two models can also be used to detect lines and planes. Other algorithms that generate prototypes of this kind are described in the next subsection. Chapter 5 contains a more detailed discussion of how the clustering algorithms described in this subsection can be used for boundary description.

B. Linear manifolds as prototypes

The earliest reference to the explicit use of non-point prototypes in connection with generalizations of FCM was Bezdek et al. (1978). These authors discussed a primitive method for fitting fuzzy clusters with lines in the plane. The *fuzzy c-varieties* (FCV) models (Bezdek et al. 1981a,b) grew out of this effort, and were the first generalizations of FCM that explicitly used many kinds of non-point prototypes. FCV uses r-dimensional linear varieties, $0 \le r \le p-1$ as prototypes in (2.24a). This model predates possibilistic partitions, so the weights $\{w_i\}$ in (2.24) are zero for the FCV objective function. The linear variety (or manifold) of dimension r through the point $\mathbf{v}_i \in \Re^p$ spanned by the linearly independent vectors $\{\mathbf{b}_{i1}, \mathbf{b}_{i2}, \ldots, \mathbf{b}_{ir}\} \subset \Re^p$ is

$$L_{ri} = \{\mathbf{y} \in \Re^p \mid \mathbf{y} = \mathbf{v}_i + \sum_{j=1}^{r} t_j \mathbf{b}_{ij}\,;\, t_j \in \Re\} \quad , \tag{2.29}$$

so $\beta_i = \{\mathbf{v}_i, \mathbf{b}_{i1}, \mathbf{b}_{i2}, \ldots, \mathbf{b}_{ir}\}$ are the parameters of L_{ri}. These prototypes can be thought of as "flat" sets in $\Re^p$. Dimension r is the number of directions in which the flatness extends. FCV uses the perpendicular distance from $\mathbf{x}_k$ to L_{ri} as the distance measure in (2.24a). When the $\{\mathbf{b}_{ij}\}$ are an orthonormal basis for their span, the orthogonal projection theorem yields

$$D^2_{L_{r,ik}} = \|\mathbf{x}_k - \mathbf{v}_i\|^2_A - \sum_{j=1}^{r} \left\langle \mathbf{x}_k - \mathbf{v}_i, \mathbf{b}_{ij} \right\rangle^2_A \quad . \tag{2.30}$$

$D_{L_{r,ik}}$ is just the A-norm length of $(\mathbf{x}_k - \mathbf{v}_i)$ minus the A-norm length of its unique best approximation by a vector in the span of the $\{\mathbf{b}_{ij} \mid j=1,\ldots,r\}$. When $r = 0$ equation (2.30) reduces to $D^2_{ik} = \|\mathbf{x}_k - \mathbf{v}_i\|^2_A$ as used in (2.5), so for r = 0, FCV reduces to FCM. For r = 1, FCV becomes *Fuzzy c-Lines* (FCL), for r = 2, *Fuzzy c-Planes* (FCP), etc., and for r = p −1, *Fuzzy c-Hyperplanes* (FCHP). In the FCV-AO algorithms derived to optimize the FCV model, the fuzzy c-partition matrix U and the centers $\{\mathbf{v}_i\}$ are updated with FCM formulae (2.7a, b) except that the squared distance in (2.30) is used in place of $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|_A^2$. First order necessary conditions for minimizing the FCV functional now include the spanning vectors $\{\mathbf{b}_{ij}\}$, which are updated at each iteration by finding

$$\mathbf{b}_{ij} = \text{the j-th unit eigenvector of } C_i,\ j=1, 2, \ldots p \quad , \tag{2.31}$$

where the $\{\mathbf{b}_{ij}\}$ are arranged in the same order as the descending eigenvalues of C_i, and C_i is the fuzzy covariance matrix of cluster i given by (2.27). The eigenvectors are assumed to be arranged corresponding to a descending ordering of their eigenvalues. For r > 0 it is of course necessary to get the eigenvalues and eigenvectors of the fuzzy covariance matrices at each pass through FCV-AO. This is usually done with singular value decomposition, and makes FCV and its relatives more computationally intense than the simpler point prototype models.

Since FCV-AO uses perpendicular distance to the linear varieties, it does not take into account the *extent* (i.e., length, area, etc.) of the flat clusters being sought. For example, if r = 1, FCV seeks (infinitely long) lines, and thus can lump approximately collinear clusters together, even when they are very far apart (see Figure 24.1 in Bezdek (1981)). One solution to this problem is to choose a distance measure given by

$$D_{10,ik}^2 = \alpha \underbrace{D_{L_{1,ik}}^2}_{\text{lines}} + (1-\alpha) \underbrace{D_{L_{0,ik}}^2}_{\text{points}} \; ; \; 0 \le \alpha \le 1 \quad , \tag{2.32}$$

which is a *convex combination* of the perpendicular distance from $\mathbf{x}_k$ to $L_{1,ik}$ and the point distance from $\mathbf{x}_k$ to $\mathbf{v}_i$. See Figure 4.50 for a geometric interpretation of the distance in equation (2.32). Parameter α can vary from 0 for spherical clusters (having point prototypes) to 1 for linear clusters (having line prototypes).

The algorithm resulting from first order necessary conditions for $J_m(U, \mathbf{B})$ with distance (2.32) is called the *fuzzy c -elliptotypes* (FCE-AO) algorithm (Bezdek et al., 1981b). More generally, AO algorithms to optimize any convex combination of FCV terms with dimensions $\{r_i\}$ and convex weights $\{\alpha_i\}$ were derived by Bezdek et al. (1981a, b). The purpose of this is to allow the clusters to have shapes built from convex combinations of flat shapes. However, the actual prototypes from the convex combinations model are not easily recognizable geometric entities such as ellipses; rather, they are mathematical entities described in terms of level sets of certain functions.

While the parameters **V** and **A** in the GK model can be jointly viewed as "generalized" prototypes, FCV was the first generalization of FCM that explicitly used non-point prototypes for **B**. The FCV algorithms and the particular convex combination FCE have found various applications over the years (Jacobsen and Gunderson, 1983, Gunderson and Thrane, 1985, Yoshinari et al., 1993). However, a rough idea of the shape of the clusters in the data set must be known *a priori* (which is impossible for $p > 3$) to select proper values for the dimensions $\{r_i\}$ and convex weights $\{\alpha_i\}$. An important exception is rule extraction for function approximation in fuzzy input-output systems. FCE seems well suited to this problem because the input-output space of often $\Re^3$, and linear Takagi-Sugeno (1985) output functions can be fitted quite well with FCE (Runkler and Palm, 1996; Runkler and Bezdek, 1998c; and Example 4.17).

Adaptive fuzzy c-elliptotypes (AFCE). Perhaps the biggest drawback of FCV and convex combinations like FCE is that these models find c clusters with prototypical "shapes" that are all the same. The reason for this is that FCV uses the same real dimension (r) and its convex combinations all use the same "mixture of dimensions" for all c clusters, so cluster substructure having these characteristics is imposed on the data whether they possess it or not. This problem resulted in the first locally adaptive fuzzy clustering method (the GK model), and the next generation of locally adaptive clustering methods followed rapidly on the heels of the FCV models.

There are a number of ways to make FCV adaptive. The earliest scheme for local adaptation in the FCV models was due to Anderson et al. (1982). They suggested that the value of α used in convex combinations of the FCV functionals should be different for each cluster, reflecting a customized distance measure that best represents the shape of each cluster. When convex combinations are used, there is no dimensionality of prototypes. (We remind you that it is the *distances* in the FCV objective function that become convex combinations in Bezdek et al. (1981a, b), and not the fitting prototypes. The fitting prototypes in AFCE, as in FCE, are no longer recognizable geometric entities.) The basic idea in FCE is to mediate between geometric needs for point prototypes (central tendencies) and varietal structure (shape or dispersions). But convex combinations of FCV such as FCE fix the amount by which each factor contributes to the overall representation of all c clusters.

Anderson et al. (1982) regulated each cluster through the shape information possessed by the eigenstructure of its fuzzy covariance matrix. Adaptation is with respect to the convex weights in (2.32) used for each cluster. For $X \subset \Re^2$ the modification of FCE to AFCE is

$$\alpha_i = 1 - \frac{\lambda_{i,\min}}{\lambda_{i,\max}}, i = 1, 2 \quad , \qquad (2.33)$$

where $\lambda_{i,\max}$ is the larger eigenvalue and $\lambda_{i,\min}$ is the smaller eigenvalue of the 2×2 fuzzy covariance matrix C_i of cluster i, i=1,2. Equation (2.33) covers only the 2D case. Extensions to higher dimensions may be found in Phansalkar and Davé (1997) and Kim (1997). The AFCE-AO algorithms are exactly the same as the FCE-AO methods just described except that the convex weights in (2.32) are updated at each iteration with (2.33).

Example 2.6 Figure 2.6 shows the results of clustering two data sets with FCL-AO, AFCE-AO and GK-AO. Each of these models has a different kind of prototype (lines, elliptotypes and points, respectively); all three are configured for possible success with data of this kind. FCL, however, is more rigid than the other two because it does not have a feature that enables localized adaptation. The left panel depicts three intersecting noisy linear clusters of different sizes, and the right side shows three noisy linear clusters, two of which are collinear.

Run time protocols for this example were as follows. The covariance matrices for all three methods were initialized with the (U, **V**) output of the fifth iteration of FCM-AO, m = 2, c = 3 using the Euclidean norm. FCM-AO was itself initialized with the first 3 points in each data set (($\mathbf{v}_{1,0} = (80, 81)^T$, $\mathbf{v}_{2,0} = (84, 84)^T$ and $\mathbf{v}_{3,0} = (87, 89)^T$) in the left views, and ($\mathbf{v}_{1,0} = (10, 11)^T$, $\mathbf{v}_{2,0} = (11, 189)^T$ and $\mathbf{v}_{3,0} = (14, 14)^T$) in the right views). Termination of all three methods by either of $\|\mathbf{V}_{t+1} - \mathbf{V}_t\|_\infty \le 0.001$ or $\|U_{t+1} - U_t\|_\infty \le 0.01$ yielded the same results. FCV and AFCE both used m=2, and GK used m = 1.5 (our experience is that GK does much better with a value near 1.5 than it does with a value near 2).

The results shown are terminal partitions hardened with (2.10), each cluster identified by a different symbol. In the collinear situation for the right hand views, FCL finds two almost coincidental clusters and the points belonging to these two clusters are arbitrarily assigned to the two prototypes in view 2.6d. The AFCE result in 2.6f is much better, having just two squares that are erroneously grouped with the dots. GK makes perfect assignments, as shown in Figure 2.6e. Terminal values of α for AFCE were very nearly 1 for both data sets.

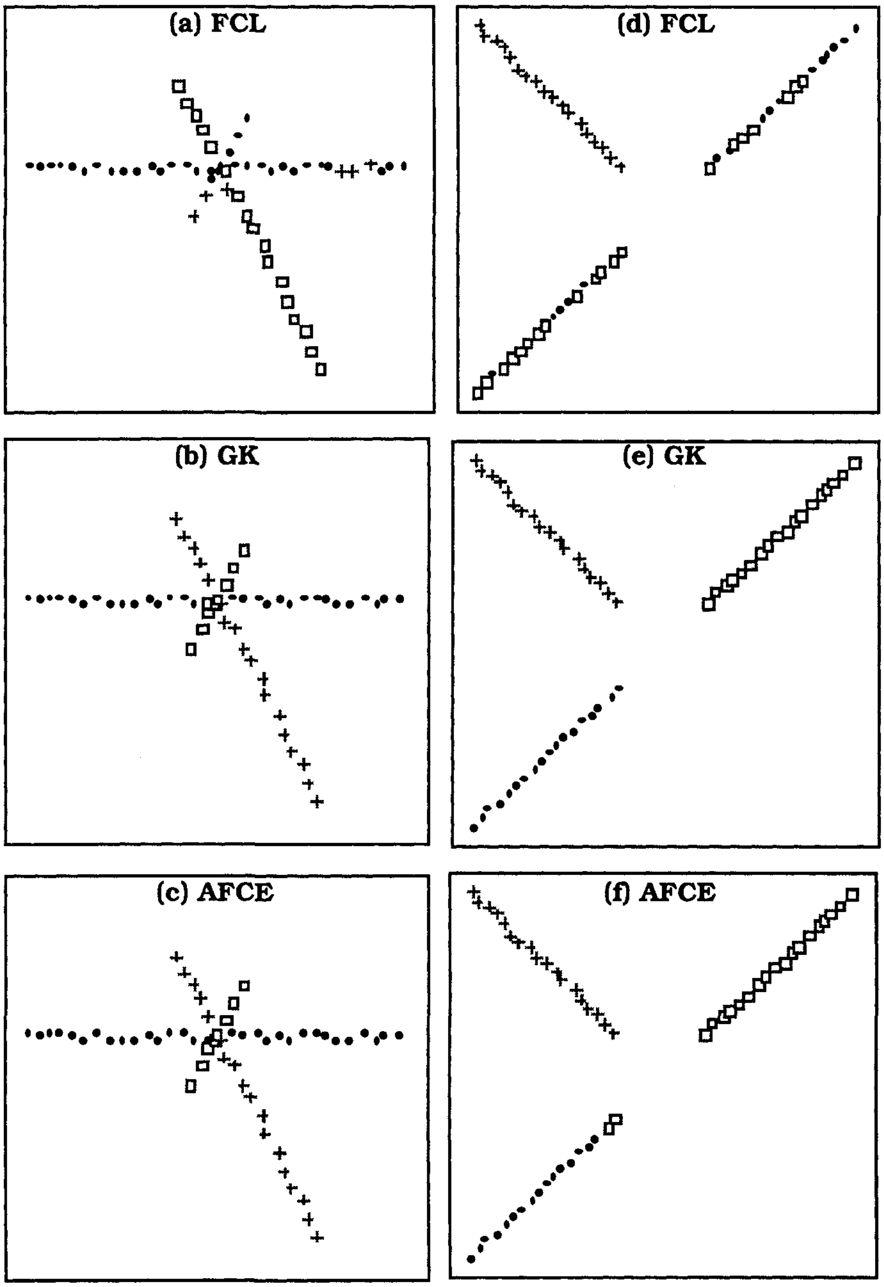


Figure 2.6 Detection of linear clusters with FCL, GK and AFCE

For the three well separated lines (the right views in Figure 2.6), all three values were 0.999; for the intersecting lines in the left views of Figure 2.6, the three terminal values of α were 0.999, 0.997 and

0.999. These are the expected results, since the clusters in both data sets are essentially linear, so the ratio of eigenvalues in (2.33) is essentially zero.

The tendency of FCV to disregard compactness is seen in the cluster denoted by the six "+" signs in panel 2.6(a). Here the pluses and dots are clearly interspersed incorrectly. For this data, both GK and AFCE produce flawless results. One possible explanation for this is that the FCV functional is more susceptible to being trapped by local minima, since it cannot adapt locally like GK and AFCE.

AFCE is called AFC (adaptive fuzzy clustering) in many of the later papers on this topic, especially those of Davé (1989a, 1990a). Because several other adaptive schemes discussed in this chapter are not based on FCE, we prefer to call this method AFCE. Davé and Patel (1990) considered the problem of discovering the unknown number of clusters. They proposed progressive removal of clusters that are good fits to subsets of the data. This idea was further developed for lines and planes in Krisnapuram and Freg (1992).

Adaptive Fuzzy c-Varieties (AFCV) Gunderson (1983) introduced a heuristic way to make the integer dimension (r_i) of the fitting prototype for class i independent of the dimensions used by other clusters sought in the data. His *adaptive fuzzy c - varieties* (AFCV) scheme is based on the eigenstructure of the fuzzy covariance matrices $\{C_i\}$ at (2.27) that are part of the necessary conditions for extrema of the FCV functional.

Gunderson observed that the distance calculations made in the necessary conditions for $\mathbf{U}_{(i)}$, the i-th row of partition matrix U shown at (2.7a), are independent of how the distances themselves are computed - that is, (2.7a) does not care what value of r is used in equation (2.30). He reasoned that making a heuristic adjustment to the optimality conditions by allowing different $D^2_{L_{r,ik}}$'s to be used in (2.7a) for different i's might enable FCV to seek manifolds of different dimensions for the various clusters. A second modification of the necessary conditions was to introduce a non-convex weight $\hat{\alpha}$ into distance equation (2.30) as follows:

$$\hat{D}^2_{L_{r_i,ik}} = \|\mathbf{x}_k - \mathbf{v}_i\|^2_A - \hat{\alpha}\left(\sum_{j=1}^{r_i}\langle \mathbf{x}_k - \mathbf{v}_i, \mathbf{b}_{ij}\rangle^2_A\right) \quad . \tag{2.34}$$

The user defined parameter $\hat{\alpha}$ in (2.34) essentially controls the importance of the non-point or r > 0 part of the distance calculation for each cluster, and not much guidance is given about its selection. Gunderson's modification of FCV also calls for the selection of (p-1) *shaping coefficients* $\{\sigma_r : 1 \le r \le p-1\}$ which are compared to ratios of eigenvalues from the fuzzy scatter matrices $\{C_i\}$ at (2.27). In particular, if $\{\lambda_{ip} \le \lambda_{i,p-1} \le \cdots \le \lambda_{i1}\}$ are the ordered eigenvalues of C_i, Gunderson adapts the dimension of each FCV prototype during iteration as follows: If there exists a least integer k, k = 1, 2, ..., p-1 so that $(\lambda_{i,k+1}/\lambda_{i,k}) < \sigma_k$; $1 \le i \le c$, set r_i = k; otherwise, set r_i = 0. The parameter σ_k is also user defined, and again, is fine tuned during iteration, much like many other algorithmic parameters, to secure the most acceptable solution to the user. Then, $\mathbf{U}_{(i)}$ is updated with r = r_i in (2.30). These two changes are analogous to the modifications of FCM that Bensaid et al. (1996a) used to create ssFCM: the resultant algorithm no longer attempts to solve a well-posed optimization problem.

Example 2.7 Figure 2.7 is adapted from Figure 5 in Gunderson (1983). Figure 2.7 shows the output obtained by applying Gunderson's adaptive FCV to a data set in $\Re^2$ that contains four visually apparent clusters. The two upper clusters are roughly circular, cloud-type structures while the two lower are elongated, linear structures.

Using the Euclidean norm, c =4 and m = 1.75 in (2.24), and $\hat{\alpha}$ = 0.95 in (2.34), Gunderson's algorithm iteratively chooses $r_1 = r_2 = 0$, so that the cloud shaped clusters are represented, respectively, by the point prototypes $\mathbf{v}_1$ and $\mathbf{v}_2$ as shown in Figure 2.7. And the algorithm settles on $r_3 = r_4 = 1$, so that the linear clusters have prototypes that are shown as the lines L_{13} and L_{14} in Figure 2.7. The value of σ_k is not specified.

Summarizing, Gunderson's method makes FCV adaptive with respect to the dimensions $\{r_i\}$ of the linear varieties $\{L_{ri}\}$. Different clusters are allowed to have representation as linear manifolds of possibly different dimensions. In contrast, the adaptive GK model does not provide non-point prototypes; instead, it adapts the norms $\{A_i\}$ of the clusters so that their level sets implicitly match the cluster shapes.

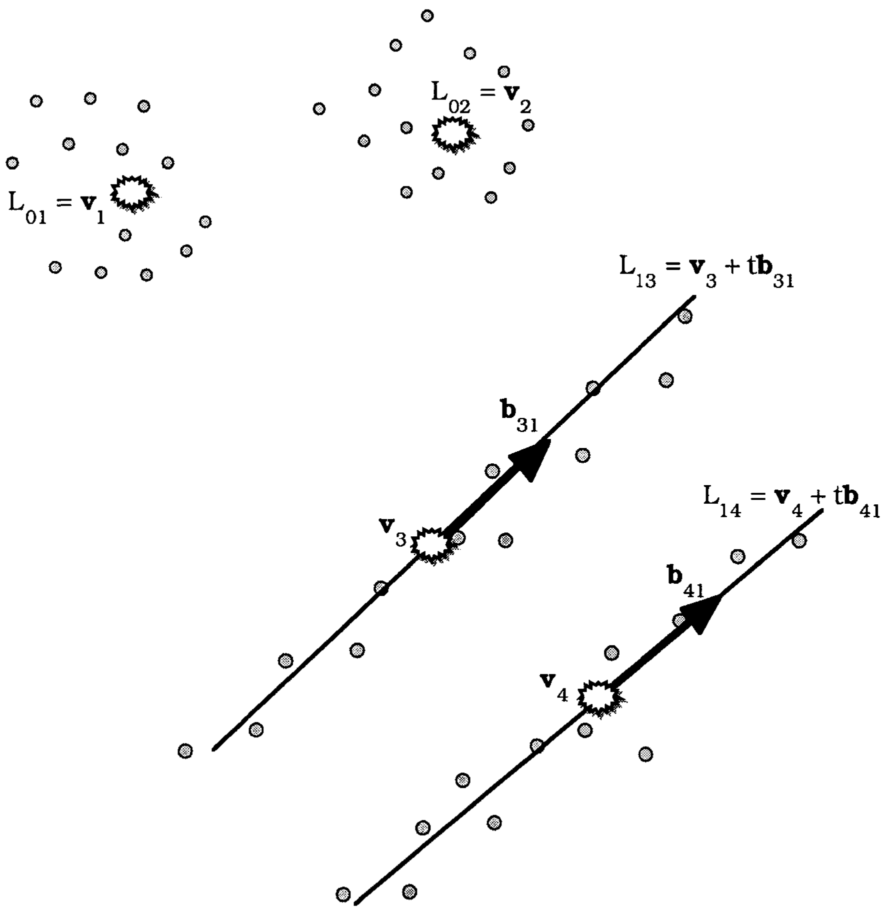


Figure 2.7 Gunderson's FCV for two clouds and two lines

C. Spherical Prototypes

Coray (1981) first suggested the use of circular prototypes for clusters resembling circular arcs - that is, shell-like structures, as opposed to cloud like structures (it is arguable whether linear clusters such as those in Figure 2.6 are clouds or shells - they seem to be the boundary case between the two types of structures). This line of research evolved to the *fuzzy c-shells* (FCS) algorithms (Davé and Bhamidipati, 1989, Davé, 1990b, 1992) and the *fuzzy c-spherical shells* (FCSS) algorithms of Krishnapuram et al. (1992). In these algorithms the i-th prototype is the hypersphere

$S_i(\mathbf{x}; \mathbf{v}_i, r_i) = \left\{\mathbf{x} \in \Re^p : \left\|\mathbf{x} - \mathbf{v}_i\right\| = r_i\right\}$ centered at $\mathbf{v}_i$ with radius r_i, so $\boldsymbol{\beta}_i = (\mathbf{v}_i, r_i)$. The proximity measure D_{ik} used by these two models is different, and hence, the parameters $\boldsymbol{\beta}_i$ are updated differently. Davé's FCS uses the exact distance from feature vector $\mathbf{x}_k$ to the spherical shell of cluster i,

$$D_{ik}^2 = \left(\left\|\mathbf{x}_k - \mathbf{v}_i\right\| - r_i\right)^2 \quad . \tag{2.35}$$

This distance, illustrated in Figure 2.8 for p=2, is the (squared) Euclidean distance between data point $\mathbf{x}_k$ and (the tangent to) the prototypical circle S_i that lies along a radius directed towards the datum.

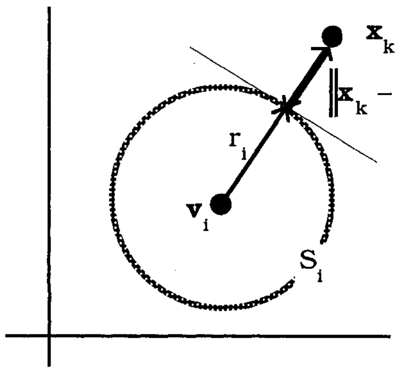


Figure 2.8 The distance basis for Davé's FCS

Minimization of (2.24a) in the non-possibilistic case when all distances between data and point prototypes $\{\mathbf{v}_i\}$ are non-zero and distance (2.35) is used for (2.24c) yields the usual necessary conditions for U (namely, FCM equation (2.7a)). However, differentiation with respect to $\mathbf{v}_i$ and r_i when (2.35) is used yields the necessary conditions

$$\sum_{k=1}^{n} u_{ik}^m \left[1 - \frac{r_i}{\left\|\mathbf{x}_k - \mathbf{v}_i\right\|}\right](\mathbf{x}_k - \mathbf{v}_i) = \mathbf{0} \quad \text{, and} \tag{2.36a}$$

$$\sum_{k=1}^{n} u_{ik}^m \left(\left\|\mathbf{x}_k - \mathbf{v}_i\right\| - r_i\right) = 0 \quad . \tag{2.36b}$$

These equations are not explicit in r_i and $\mathbf{v}_i$. Therefore a technique such as Newton's method that solves a set of coupled nonlinear equations must be used at each half iterate to estimate these parameters. This makes FCS computationally expensive. Bezdek and Hathaway (1992) showed that an exact solution of (2.36) is not required. Instead, only one step of Newton's method is needed at each half iterate. Man and Gath (1994) have suggested another variant of FCS in which the center and radius estimates are updated independently rather than found by simultaneous solution using (2.36). This avoids the need for numerical techniques, but may increase the overall number of iterations required for termination.

Krishnapuram et al.'s FCSS avoids the need for numerical solution of necessary conditions at each half iterate by using the algebraically defined proximity

$$D_{ik}^2 = \left(\|\mathbf{x}_k - \mathbf{v}_i\|^2 - r_i^2\right)^2 \quad . \tag{2.37}$$

Defining

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{x}_k \\ 1 \end{bmatrix} \text{ and } \mathbf{p}_i(\mathbf{v}_i, r_i) = \begin{bmatrix} -2\mathbf{v}_i \\ \mathbf{v}_i^T\mathbf{v}_i - r_i^2 \end{bmatrix} \quad , \tag{2.38}$$

it is easy to show that this minor modification of (2.35) makes the parameter update equations explicit:

$$\mathbf{p}_i = -\frac{1}{2} H_i^{-1} \mathbf{w}_i \quad \text{, where} \tag{2.39}$$

$$H_i = \sum_{k=1}^{n} u_{ik}^m \mathbf{y}_k \mathbf{y}_k^T \text{, and } \mathbf{w}_i = 2\sum_{k=1}^{n} u_{ik}^m (\mathbf{x}_k^T \mathbf{x}_k)\mathbf{y}_k \quad . \tag{2.40}$$

In theory the exact geometric distance used in FCS gives more accurate results than the algebraically motivated distance used in FCSS, but in most practical applications the difference may not justify the higher computational cost of FCS. As a compromise, FCS is typically applied to the data after FCSS terminates (that is, FCS is often initialized with the terminal outputs from FCSS). There are quite a few early papers on these two algorithms, but both have been subsumed by the more general case of elliptical prototypes, so an example of spherical prototypes is deferred to a later subsection.

D. Elliptical Prototypes

Davé and Bhaswan (1992) proposed the *adaptive fuzzy c-shells* (AFCS) model for elliptical shells. This model uses a hyperellipse for the i-th prototype,

$$E_i(\mathbf{x}; \mathbf{v}_i, A_i) = \left\{ \mathbf{x} \in \Re^p : \|\mathbf{x} - \mathbf{v}_i\|_{A_i}^2 = 1 \right\} \quad , \tag{2.41}$$

where A_i is a positive definite symmetric matrix which determines the major and minor axes lengths as well as the orientation of the hyperellipse, and $\mathbf{v}_i$ is its center. Consider the distance D^2_{ik} defined by

$$D^2_{ik} = \left[\|\mathbf{x} - \mathbf{v}_i\|_{A_i} - 1 \right]^2 \quad . \tag{2.42}$$

Davé and Bhaswan showed that minimization of (2.24a) when all distances between data and point prototypes $\{\mathbf{v}_i\}$ are non-zero with D^2_{ik} as in (2.42) results in the following equations for updating the parameters $\beta_i = (\mathbf{v}_i, A_i)$ of ellipse E_i:

$$\sum_{k=1}^{n} u^m_{ik} \left(\frac{D_{ik}}{\|\mathbf{x} - \mathbf{v}_i\|_{A_i}} \right) (\mathbf{x}_k - \mathbf{v}_i) = \mathbf{0} \quad \text{, and} \tag{2.43a}$$

$$\sum_{k=1}^{n} u^m_{ik} \left(\frac{D_{ik}}{\|\mathbf{x} - \mathbf{v}_i\|_{A_i}} \right) (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^T = \mathbf{0} \quad . \tag{2.43b}$$

Like (2.36), system (2.43) must be solved numerically at each half step in the iteration. The usual necessary conditions for U hold for AFCS. The evolution of AFCS is traced through Davé and Bhaswan (1991a,b), Davé (1992) and Davé and Bhaswan (1992).

In their *fuzzy c-ellipsoidal shells* (FCES) model, Frigui and Krishnapuram (1996a) use the "radial distance" defined by

$$D^2_{R_{ik}} = \frac{\left[\|(\mathbf{x}_k - \mathbf{v}_i)\|_{A_i} - 1 \right]^2 \|(\mathbf{x}_k - \mathbf{v}_i)\|^2}{\|(\mathbf{x}_k - \mathbf{v}_i)\|^2_{A_i}} \quad . \tag{2.44}$$

$D^2_{R_{ik}}$ is a good approximation to the exact (perpendicular) distance between the ellipse E_i and points located close to its major and minor axes. If $\mathbf{z}_k$ denotes the point of intersection of the line joining $\mathbf{v}_i$ to $\mathbf{x}_k$ and E_i, then $D^2_{R_{ik}} = \|\mathbf{x}_k - \mathbf{z}_k\|^2$. We also see from (2.42) and (2.44) that $D^2_{ik} = D^2_{R_{ik}} \left(\|\mathbf{x}_k - \mathbf{v}_i\|^2_{A_i} \Big/ \|\mathbf{x}_k - \mathbf{v}_i\|^2 \right)$. Thus, D^2_{ik} is a normalized version of $D^2_{R_{ik}}$, with the normalization a function of the position of $\mathbf{x}_k$. $D^2_{R_{ik}}$ has the advantage of being simpler to compute when compared with the exact distance to E_i (see next subsection). Minimization of (2.24a) when all distances between data and point prototypes $\{\mathbf{v}_i\}$ are non-zero with $D^2_{R_{ik}}$ in (2.24c) results in the following update equations for $\beta_i = (\mathbf{v}_i, A_i)$:

$$\sum_{k=1}^{n} u_{ik}^m \left(\frac{D_{R_{ik}}}{\|\mathbf{x}_k - \mathbf{v}_i\|^3_{A_i}} \right) \|\mathbf{x}_k - \mathbf{v}_i\| (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^T = \mathbf{0} \quad \text{, and} \tag{2.45a}$$

$$\sum_{k=1}^{n} u_{ik}^m \left[\frac{D_{R_{ik}} \|\mathbf{x}_k - \mathbf{v}_i\|}{\|\mathbf{x}_k - \mathbf{v}_i\|^3_{A_i}} A_i - \frac{D^2_{R_{ik}}}{\|\mathbf{x}_k - \mathbf{v}_i\|^2} I \right] (\mathbf{x}_k - \mathbf{v}_i) = \mathbf{0} . \tag{2.45b}$$

Equations (2.45) can be solved numerically with the Levenberg-Marquardt algorithm. Frigui and Krishnapuram (1996a) have shown that $D^2_{R_{ik}}$ performs better than D^2_{ik}, especially when the data are scattered and when the ellipses are of widely varying sizes.

E. Quadric Prototypes

(Krishnapuram et al., 1991) first generalized shell clustering to the quadric case. The general hyperquadric shell in $\Re^p$ with coordinate axes $x_1,\ldots,x_p$ can be written as

$$Q_i(\mathbf{x}; \mathbf{p}_i) = \left\{ \mathbf{x} \in \Re^p ; \left\langle \mathbf{p}_i, \mathbf{q} \right\rangle = 0 \right\} \quad \text{, where} \tag{2.46a}$$

$$\mathbf{p}_i^T = \left[p_{1i}, \ldots, p_{pi}, p_{(p+1)i}, \ldots, p_{ri}, p_{(r+1)i}, \ldots, p_{(r+p)i}, p_{(r+p+1)i} \right], \tag{2.46b}$$

$$\mathbf{q}^T = \left[x_1^2, \ldots, x_p^2, x_1x_2, \ldots, x_{p-1}x_p, x_1, \ldots, x_p, 1 \right] \quad \text{, and} \tag{2.46c}$$

$$r = p(p+1)/2 \quad . \tag{2.46d}$$

Define the algebraic (or residual) distance from a point $\mathbf{x}_k$ to prototype Q_i with parameters $\boldsymbol{\beta}_i = \mathbf{p}_i$ as

$$D^2_{Q_{ik}} = \mathbf{p}_i^T \mathbf{q}_k \mathbf{q}_k^T \mathbf{p}_i \qquad \text{, where} \qquad (2.47a)$$

$$\mathbf{q}_k^T = \left[x_{1k}^2, \ldots, x_{pk}^2, x_{1k}x_{2k}, \ldots, x_{(p-1)k}x_{pk}, x_{1k}, \ldots, x_{pk}, 1 \right]. \qquad (2.47b)$$

In order to obtain a fuzzy c-partition of the data, Krisnapuram et al. minimize the non-possibilistic form of (2.24a) when all distances between data and point prototypes $\{\mathbf{v}_i\}$ are non-zero with $D^2_{Q_{ik}}$ as the underlying distance measure. However, since the objective function is homogeneous with respect to $\mathbf{p}_i$, we need to constrain the problem in order to avoid the trivial solution. In their *fuzzy c-quadrics* (FCQ) model Davé and Bhaswan (1992) use the constraint $p_{1i} = 1$. However, the resulting proximity is not invariant to rotation. Moreover, it precludes linear prototypes and certain paraboloids. Another possibility is (Krishnapuram et al., 1991)

$$p_{1i}^2 + \ldots + p_{pi}^2 + \tfrac{1}{2} p_{(p+1)i}^2 + \ldots + \tfrac{1}{2} p_{ri}^2 = 1 \qquad . \qquad (2.48)$$

This constraint was used by Bookstein (1979) for fitting quadrics and has the advantage that the resulting distance measure is invariant to rigid transformations of the prototype. However, it does not allow the solution to be linear or planar. Many other constraints are also possible. Krishnapuram et al. (1995a) have shown that the above constraint is the best compromise between computational complexity and performance in the 2-D case. If for the i-th prototype we define

$$a_{ji} = \begin{Bmatrix} p_{ji} & ; \ 1 \le j \le p \\ \dfrac{p_{ji}}{\sqrt{2}} & ; \ p+1 \le j \le r \end{Bmatrix} \qquad \text{, and} \qquad (2.49a)$$

$$b_{ji} = p_{ji} \quad \text{for } j = r+1, r+2, \ldots, r+p+1 \qquad , \qquad (2.49b)$$

then the constraint in (2.48) simplifies to $\|\mathbf{a}_i\|^2 = 1$. It is easy to show that the necessary conditions under this constraint are

$$\mathbf{a}_i = \text{eigenvector of } (F_i - G_i^T H_i^{-1} G_i) \text{ for its smallest eigenvalue;} \qquad (2.50a)$$

$$\mathbf{b}_i = -H_i^{-1} G_i \mathbf{a}_i \qquad \text{; where} \qquad (2.50b)$$

$$F_i = \sum_{k=1}^{n} u_{ik}^m \mathbf{r}_k \mathbf{r}_k^T, \quad G_i = \sum_{k=1}^{n} u_{ik}^m \mathbf{t}_k \mathbf{r}_k^T, \quad H_i = \sum_{k=1}^{n} u_{ik}^m \mathbf{t}_k \mathbf{t}_k^T, \qquad (2.50c)$$

$$\mathbf{r}_k^T = \left[x_{1k}^2, \ldots, x_{pk}^2, \sqrt{2}x_{1k}x_{2k}, \ldots, \sqrt{2}x_{(p-1)k}x_{pk}\right] \quad \text{, and} \qquad (2.50d)$$

$$\text{and } \mathbf{t}_k^T = \left[x_{1k}, \ldots, x_{pk}, 1\right] \quad . \qquad (2.50e)$$

The algebraic distance in (2.47a) is highly nonlinear in nature. When there are curves (surfaces) of highly-varying sizes, the algebraic distance is biased toward smaller curves (surfaces), and for a particular curve (surface) the distance measure is biased towards points inside the curve (surface) as opposed to points outside. This can lead to undesirable fits (Davé and Bhaswan, 1992, Krishnapuram et al. 1995a). To alleviate this problem, use the exact (perpendicular) distance denoted by $D_{P_{ik}}^2$ between the point $\mathbf{x}_k$ and the shell Q_i. To compute $D_{P_{ik}}^2$, (2.46a) is first rewritten as

$$\mathbf{x}^T A_i \mathbf{x} + \mathbf{x}^T \mathbf{b}_i + c_i = 0 \quad . \qquad (2.51)$$

In (2.51), it is assumed that the coordinate system has been rotated to make the matrix A_i diagonal. The closest point $\mathbf{z}$ on the hyperquadric to point $\mathbf{x}_k$ can be obtained by minimizing $\left\|\mathbf{x}_k - \mathbf{z}\right\|^2$ subject to

$$\mathbf{z}^T A_i \mathbf{z} + \mathbf{z}^T \mathbf{b}_i + c_i = 0 \quad . \qquad (2.52)$$

By using a LaGrange multiplier λ, the solution is found to be

$$\mathbf{z} = \tfrac{1}{2}\left(I - \lambda A_i\right)^{-1}\left(\lambda \mathbf{b}_i + 2\mathbf{x}_k\right) \quad . \qquad (2.53)$$

In the 2-D case (i. e., p=2), substituting (2.53) into (2.52) yields a quartic equation in λ, which has at most four real roots λ_j, j =1,...,4. The four roots can be computed using the standard closed-form solution. For each real root λ_j, the corresponding $\mathbf{z}$ vector $\mathbf{z}_j$ can be computed with (2.53), and $D_{P_{ik}}^2$ is finally computed using

$$D_{P_{ik}}^2 = \min_j \left\|\mathbf{x}_k - \mathbf{z}_j\right\|^2 \quad . \qquad (2.54)$$

Minimization of the non-possibilistic form of (2.24a) with respect to $\mathbf{p}_i$ when all distances between data and non-point prototypes $\{\boldsymbol{\beta}_i\}$ are non-zero (with $D_{P_{ik}}^2$ as the underlying proximity measure) can again be achieved only by numerical techniques. To reduce the computational burden, we assume we can obtain approximately the same values for $\mathbf{p}_i$ by using (2.49) and (2.50), which will be true if all the feature points lie reasonably close to the hyperquadric shells. The resulting algorithm, in which the memberships are updated using $D^2_{P_{ik}}$ of (2.54) in (2.7a), but the prototypes are updated using $D^2_{Q_{ik}}$, is known as the *fuzzy c -quadric shells* (FCQS) algorithm. The 2D case leads to a quartic equation whose roots can be found in closed form; for higher dimensions, we must resort to numerical solutions.

Krishnapuram et al. (1995a) have shown that FCQS is adequate for some boundary description applications, and we return to this application in Chapter 5. The procedure described above to solve for the exact distance is practical only in the 2-D case. In higher dimensions, one needs to solve for the roots of a sixth (or higher) degree polynomial. To overcome this, Krishnapuram et al. (1995a) developed an alternative algorithm that uses an approximate distance (Taubin, 1991). Roughly speaking, this approximate distance corresponds to the first-order approximation of the exact distance. It is given by

$$D^2_{A_{ik}} = \frac{D^2_{Q_{ik}}}{\left|\nabla D_{Q_{ik}}\right|^2} = \frac{\mathbf{p}_i^T \mathbf{q}_k \mathbf{q}_k^T \mathbf{p}_i}{\mathbf{p}_i^T D_k D_k^T \mathbf{p}_i} \quad , \tag{2.55}$$

where $\nabla D_{Q_{ik}}$ is the gradient of $\mathbf{p}_i^T\mathbf{q}$ evaluated at $\mathbf{x}_k$. In (2.55) the matrix D_k is the Jacobian of $\mathbf{q}$ evaluated at $\mathbf{x}_k$. The minimization with respect to $\mathbf{p}_i$ of the non-possibilistic form of (2.24a) with $D^2_{A_{ik}}$ as the underlying proximity measure leads to coupled nonlinear equations which can be solved only iteratively. To avoid this problem, Krishnapuram et al. (1995) choose the constraint

$$\mathbf{p}_i^T G_i \mathbf{p}_i = n_i, \quad i = 1, \ldots, c \qquad \text{, where} \tag{2.56}$$

$$G_i = \sum_{k=1}^{n} u_{ik}^m D_k D_k^T \text{ and } n_i = \sum_{k=1}^{n} u_{ik}^m \quad . \tag{2.57}$$

This constraint is a generalization of the constraint used by Taubin (1991) for the (crisp) single curve case. Minimization of (2.24a) subject to (2.56) yields complicated equations that cannot be solved explicitly for $\mathbf{p}_i$. To avoid iterative solutions we assume that most of the data points are close to the prototypes, so the memberships $\{u_{ik}\}$ will be relatively crisp (i. e., close to 0 or 1). This assumption is also valid if we use possibilistic memberships.

The magnitude of the gradient at all points with high memberships in cluster i is approximately constant, i.e., $\mathbf{p}_i^T D_k D_k^T \mathbf{p}_i \approx 1$. In fact, the condition $\mathbf{p}_i^T D_k D_k^T \mathbf{p}_i \approx 1$ holds exactly for the case of lines/planes and certain quadrics such as circles and cylinders. Since $D^2_{Q_{ik}}$ and $D^2_{A_{ik}}$ differ only in the denominator which is ≈ 1, we will obtain approximately the same solution if we minimize (2.24a) with $D^2_{Q_{ik}}$ (rather than $D^2_{A_{ik}}$) as the distance measure subject to the constraint in (2.56). This leads to the generalized eigenvector solution for the prototype update:

$$F_i \mathbf{p}_i = \lambda_i G_i \mathbf{p}_i \quad \text{, where} \tag{2.58}$$

$$F_i = \sum_{k=1}^{n} u_{ik}^m \mathbf{q}_k \mathbf{q}_k^T \quad . \tag{2.59}$$

Unfortunately, since G_i is rank-deficient, (2.58) cannot be converted to the standard eigenvector problem. (The last row of $D_k = [0,...,0]$.) However, (2.58) can still be solved using other techniques that use the modified Cholesky decomposition (Taubin, 1991), and the solution is computationally inexpensive when p=2 or 3.

The assumption that $\mathbf{p}_i^T D_k D_k^T \mathbf{p}_i \approx 1$ is not valid for many geometric shapes when $p \geq 3$. One solution is to treat $\mathbf{p}_i^T D_k D_k^T \mathbf{p}_i$ as a weighting factor which is treated as a constant while deriving the update equation for $\mathbf{p}_i$. If we assume that the value of $\mathbf{p}_i$ does not change drastically from iteration to iteration, the weighting factor can be computed using the parameter values from the previous iteration. In this case, the update equation for $\mathbf{p}_i$ will remain the same as (2.58), except that

$$F_{i(t)} = \sum_{k=1}^{n} u_{ik}^m w_{(t)k} \mathbf{q}_k \mathbf{q}_k^T \text{, where } w_{(t)k} = \left[\mathbf{p}_{(t-1)i}^T D_k D_k^T \mathbf{p}_{(t-1)i}\right]^{-1} . \tag{2.60}$$

In (2.60), the subscripts in parentheses indicate iteration numbers. Since this reweight procedure is heuristic, it is not guaranteed that the fit obtained after reweighting will always be better than the one without reweighting. Therefore, it is necessary to compute the parameter vector $\mathbf{p}_i$ both with and without the weights and accept the $\mathbf{p}_i$ resulting from the reweight procedure only when the error of fit decreases. The sum of exact or approximate distances for each individual cluster may be used as a measure of the error of fit. The reweight procedure is highly recommended when $p \geq 3$.

Since constraint (2.56) allows lines and planes in addition to quadrics, the algorithm that uses (2.55) to update memberships and (2.58) (with or without reweighting) to update prototype parameters is known as the *fuzzy c - plano-quadric shells* (FCPQS) algorithm. Krishnapuram et al. (1995a) have also generalized the FCPQS algorithm to the case of hypersurfaces defined by sets of higher-order polynomials.

Shell clustering algorithms are very sensitive to initialization. Initializing FCQS and FCPQS randomly works well only in the simplest cases. When the data contain highly intermixed shell clusters, reliable initialization can sometimes be obtained with another clustering algorithm (or several of them), as illustrated in Example 2.8. We will return to the issue of sensitivity to initialization for shell clustering algorithms in Section 2.4.F.

Example 2.8 Figure 2.9 shows a typical example of the results obtained by the FCQS algorithm on a synthetic data set containing about 200 points. Fig. 2.9(a) shows the original data set, a pair of randomized ellipses. Noise uniformly distributed over [-1.5, 1.5] was added to the x and y coordinates of data points generated by sampling functional representations of the three curves so that the points do not lie on ideal curves. Figure 2.9(b) shows the resulting prototype curves superimposed on the original data set when c=3 was used.

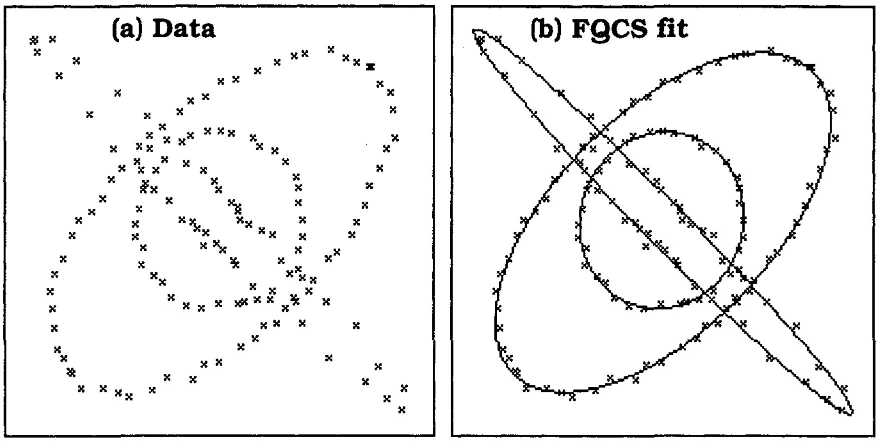


Figure 2.9 Two randomized ellipses and a circle

The results in Figure 2.9 were obtained by the sequential application of four algorithms, viz., FCQS ∘ FCSS ∘ GK ∘ FCM. First, FCM with m = 3 is applied to the data for 10 iterations. This locates initial cluster centers and memberships for the GK method. Then, 2 iterations of the GK algorithm are made with m = 2, resulting in longer, thinner clusters than are produced by FCM. The GK outputs are then used to initialize FCSS, which is again run for 5 iterations. This converts the long thin clusters to circular arcs. Finally, the FCSS outputs are used as inputs to the shell clustering method FCQS, which is run to termination with the outputs shown in Figure 2.9(b). The termination criterion for this example was to stop when the maximum change in the membership of any point in any cluster was less than 0.01. This hybrid FCQS model typically terminates in about 20 iterations and the CPU time on a Sun Sparc 1 workstation is less than 10 seconds.

Although we have only discussed the fuzzy cases in detail, the non point-prototype algorithms discussed in this section can all be used either in the hard, fuzzy or possibilistic modes. The possibilistic mode, with an initialization provided by the fuzzy mode, may be useful in noisy situations. This is because the constraint $\sum_{i=1}^{c} u_{ik} = 1$ will cause noise points to have relatively high memberships in the fuzzy clusters, which can lead to unacceptably high errors in prototype parameter estimates. However, the possibilistic mode requires that we estimate the scale parameter w_i for each cluster. In most shell clustering models, w_i may be set equal to the square of the expected thickness of the shells (Krishnapuram et al. 1995a,b).

Example 2.9 The top left view of Fig. 2.10 shows two visually apparent ellipses imbedded in a very noisy environment. Crisp, fuzzy and possibilistic quadric c-shells were all applied to this data set. All parameters for these runs were as in Example 2.9 except that c=2 and the initializations varied. Specifically, the initialization schemes were hybrid sequences of different algorithms that were applied to the data sequentially. The crisp case was the sequence of algorithms $\mathrm{HCQS} \circ \mathrm{HCSS} \circ \mathrm{ADDC} \circ \mathrm{HCM}$. The fuzzy case was the sequence $\mathrm{FCQS} \circ \mathrm{FCSS} \circ \mathrm{GK} \circ \mathrm{FCM}$. The possibilistic case was initialized by the output of FCQS, so that the bottom right view in Figure 2.10 is the result of a five algorithm sequence, $\mathrm{PCQS} \circ \mathrm{FCQS} \circ \mathrm{FCSS} \circ \mathrm{GK} \circ \mathrm{FCM}$.

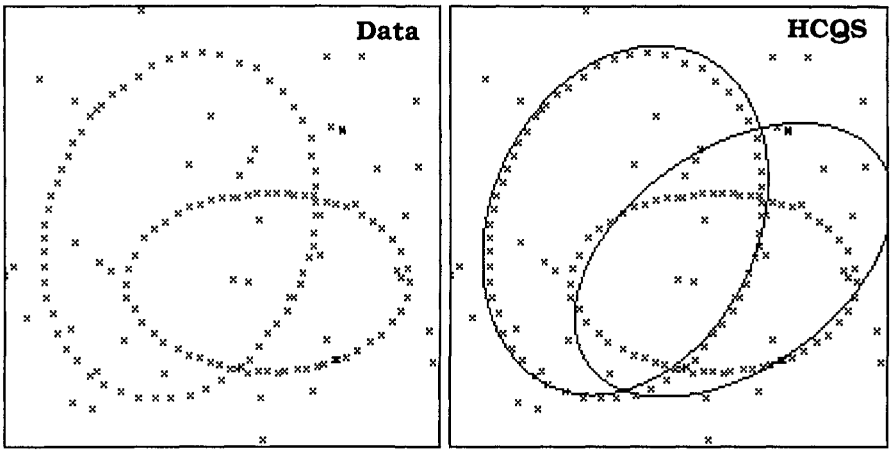


Figure 2.10 Three outputs for two ellipses in noise

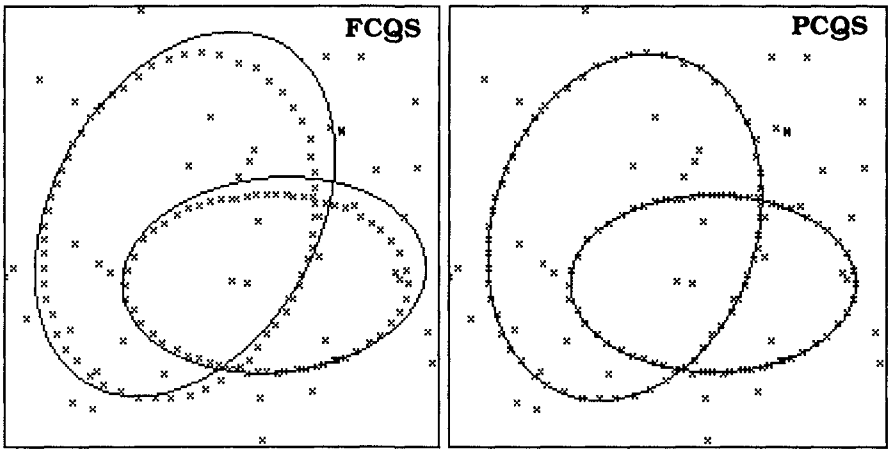


Figure 2.10 (con't.) Three outputs for two ellipses in noise

The top right view of Figure 2.10 depicts the result of a HCQS with the prototypical ellipses superimposed on the data. The fits are very poor. The bottom left in Figure 2.10 shows the result of the FCQS algorithm. This is an improvement over the HCQS result, but the effect of the noise points on the fits is still fairly significant. The bottom right view in Figure 2.10 displays the result of PCQS. The fit to the underlying pair of ellipses is quite good. This is a nice illustration of the power of hybrid models.

F. Norm induced shell prototypes

Bezdek et al. (1995) introduced a method that generates shell prototypes corresponding to level sets of *any norm* on $\Re^p$. To describe this model we need to define norm-induced shell-prototypes. The *level set* of any norm function for constant $\lambda \ge 0$ is $L_{\|x\|,\lambda} = \{\mathbf{x} \in \Re^p : \|\mathbf{x}\| = \lambda\}$. Another common notation for $L_{\|x\|,\lambda}$ emphasizes that this is also the *closed (boundary) ball* centered at **0** (the zero vector in $\Re^p$) of radius λ in the norm $\|*\|$, i.e., $L_{\|x\|,\lambda} = \partial B_{\|x\|}(\mathbf{0},\lambda) = \{\mathbf{x} \in \Re^p : \|\mathbf{x} - \mathbf{0}\| = \|\mathbf{x}\| = \lambda\}$.

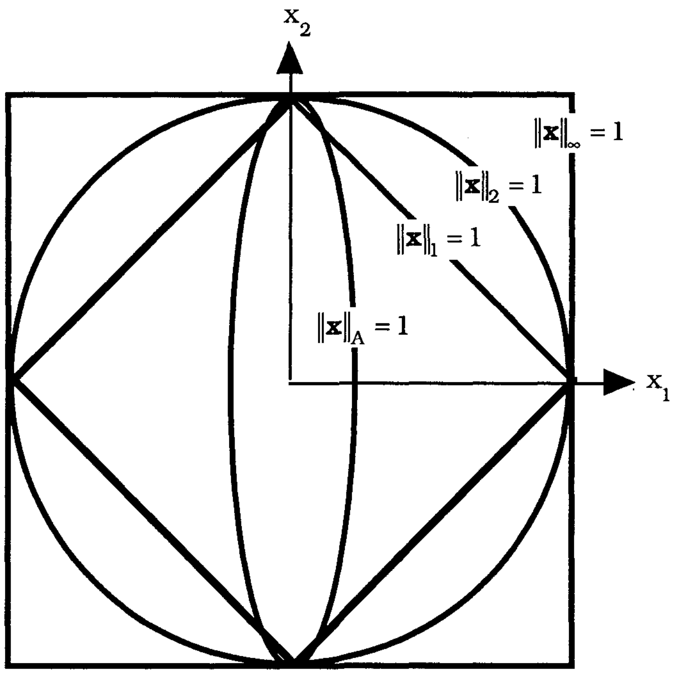


Figure 2.11 NISPs for various norms on $\Re^2$

For many, the word *ball* connotes roundness, and this is indeed the case for $\|*\|_2$, the Euclidean norm. More generally, the shape of the ball is determined by level sets of the norm. Figure 2.11 depicts some level sets (they are the boundaries shown) of various norm functions on $\Re^2$. For $\lambda = 1$, the boundaries are just the unit vectors for the norm in question. Along the boundary of the unit circle, for example, the Euclidean is 1, $\|\mathbf{x}\|_2 = 1$. All of the level sets of $\|\mathbf{x}\|_2$ are hyperspherical. If A is any positive-definite $p \times p$ matrix, the inner product norm $\|\mathbf{x}\|_A = \sqrt{\mathbf{x}^T A\mathbf{x}}$ has hyperelliptical level sets, such as the one depicted in Figure 2.11 where $\|\mathbf{x}\|_A = 1$. In other words, the ellipse where $\|\mathbf{x}\|_A = 1$ is the set of points that are equidistant from the origin of $\Re^2$ when $\|\mathbf{x}\|_A$ is the measure of distance. *Inner product norm-induced shells*, are sets generated this way.

More generally, *any norm* on $\Re^p$ has such level sets, and these are responsible for the shape of open and closed balls in their particular topology. Shown in Figure 2.11, for example, are the unit vectors (closed balls of unit radius) in the 1 and sup or ∞ norms, $\|\mathbf{x}\|_1 = \left(\sum_{j=1}^{p}|x_j|\right)$ and $\|\mathbf{x}\|_\infty = \underbrace{\max}_{1\le j\le p}\{|x_j|\}$. These two norms are special cases of the infinite family of Minkowski q-norms in (1.10). These norms cannot be induced by an inner product (except at q=2, the Euclidean norm), but they generate norm-induced shell prototypes just the same. Of particular importance for the example to follow is the shell induced by $\|*\|_1$, which is the "diamond" shown in Figure 2.11 for $\|\mathbf{x}\|_1 = 1$, $\mathbf{x}$ in $\Re^2$. The points on the diamond are equidistant from the origin in the 1-norm.

Another important fact about norms is that the square of any inner product norm is everywhere differentiable, while the squares of almost all non-inner product norms are not. This causes a great shift in the importance of using AO for approximate minimization of functionals that use norms to define the measure $D_{ik}^2 = S(\mathbf{x}_k, \boldsymbol{\beta}_i)$ in (2.24c), because most easily obtainable AO algorithms depend on solving necessary conditions obtained through differentiation. This has impeded the development of norm-induced shell prototypes that use non-inner product norms.

Recall that the FCS model of Davé is based on AO of the fuzzy version of (2.24a) with distance (2.35). Bezdek et al. (1995) proved that Davé's formula (2.35) was much more generally applicable. The main result is stated here as

Theorem NISP. Let $\mathbf{x}$ and $\mathbf{v} \in \Re^p$, $r > 0$, $\|*\|$ be a given norm on $\Re^p$ and $\partial B_{\|*\|}(\mathbf{v}, r) = \{\mathbf{y} \in \Re^p \mid \|\mathbf{y}-\mathbf{v}\| = r\}$ be the closed ball of radius r centered at $\mathbf{v}$. Then the shortest distance, as measured by $\|\cdot\|$, from any point in $\partial B_{\|*\|}(\mathbf{v}, r)$ to $\mathbf{x}$ is $\left|\,\|\mathbf{x}-\mathbf{v}\| - r\,\right|$.

This result is the basis of the NISP-AO algorithms which iteratively optimize (2.24a) when the distance in (2.24c) is defined by any norm on $\Re^p$. For example, this means that any Minkowski norm can be used in (2.24c), and theorem NISP tells us how to achieve the minimization of (2.24a) with respect to the parameters $\boldsymbol{\beta}_i = (\mathbf{v}_i, r_i)$ of the i-th shell, whose equation is $\delta B_{\|*\|}(\mathbf{v}_i, r_i) = \{\mathbf{y} \in \Re^p \,|\, \|\mathbf{y} - \mathbf{v}_i\| = r_i\}$.

Theorem NISP enables us to use other families of norms in (2.24c), by redefining $\boldsymbol{\beta}_i$ to include the shell center $\mathbf{v}_i$, radius r_i, and all other parameters needed to specify a particular member of the family of norms. As an example, suppose we seek a framework whereby it is possible to specify any *rectangle* in the plane as the i-th cluster shell. One possibility is to define a family of norms using the two real parameters a_i and θ_i as

$$\|\mathbf{x}\|_{a_i,\theta_i} = \max\left\{a_i\left|x_1\cos(\theta_i) + x_2\sin(\theta_i)\right|, \left|-x_1\sin(\theta_i) + x_2\cos(\theta_i)\right|\right\}, \quad (2.61)$$

where $0 < a_i \le 1$ and $0 \le \theta_i < \pi$. The NISP corresponding to the i-th shell is just the closed ball centered at $\mathbf{v}_i$, with radius r_i for which, *in this norm*, $\|\mathbf{x} - \mathbf{v}_i\|_{a_i,\theta_i} = r_i$ as shown in Figure 2.12.

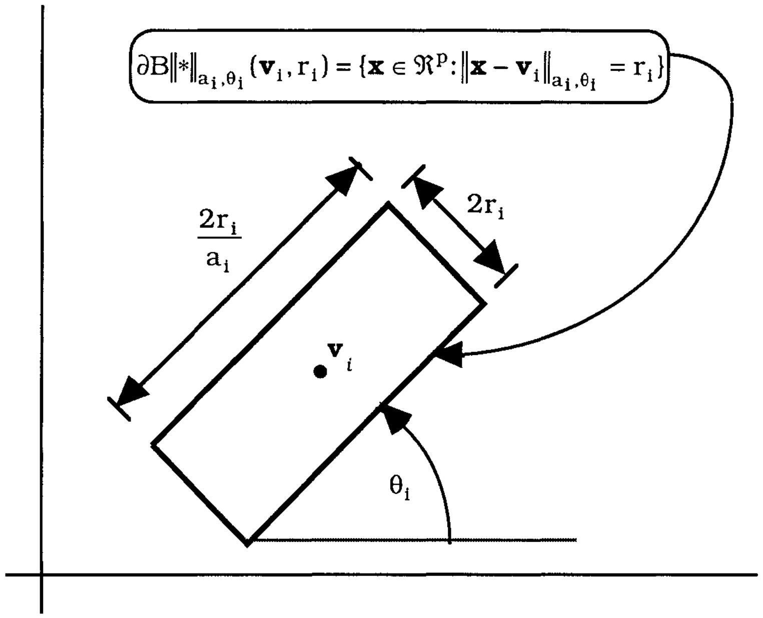

$\partial B_{\|*\|_{a_i,\theta_i}}(\mathbf{v}_i, r_i) = \{\mathbf{x} \in \Re^p : \|\mathbf{x} - \mathbf{v}_i\|_{a_i,\theta_i} = r_i\}$
$\frac{2r_i}{a_i}$
$2r_i$
$\mathbf{v}_i$
θ_i

Figure 2.12 Rectangular NISP corresponding to $\|\mathbf{x} - \mathbf{v}_i\|_{a_i,\theta_i} = r_i$

To verify that (2.61) is a vector norm on $\Re^p$, note that $\|\mathbf{x}\|_{a_i,\theta_i} = \|AQ\mathbf{x}\|_\infty$, where A and Q are nonsingular matrices, so it is just a weighted version (weighted by nonsingular matrix AQ) of another norm. The nonsingular weighting matrix is AQ, where $A = \begin{bmatrix} a_i & 0 \\ 0 & 1 \end{bmatrix}$ and $Q = \begin{bmatrix} \cos(\theta_i) & \sin(\theta_i) \\ -\sin(\theta_i) & \cos(\theta_i) \end{bmatrix}$. Nonsingularity is crucial to insure that the norm property $\|\mathbf{x}\| = 0$ implies that $\mathbf{x} = 0$ holds. The two matrices correspond to the operations that are required to turn the square into the rotated rectangle, namely: a rotation through θ_i (represented by Q), and a stretch (represented by A). We then let $\boldsymbol{\beta}_i=(\mathbf{v}_i, r_i, a_i, \theta_i)$ and use $D_{ik}(\mathbf{x}_k, \boldsymbol{\beta}_i) = \left| \|\mathbf{x}_k - \mathbf{v}_i\|_{a_i,\theta_i} - r_i \right|^2$ in (2.24c). Optimization of (2.24a) in all three cases (hard, fuzzy and possibilistic) can be done using AO directly or after reformulation as in (2.23) via the reformulation theorem. Alternatively, optimization can be done using, say, a genetic algorithm approach. In example 2.10 from Bezdek et al. (1995), a hybrid algorithm composed of FCM followed by reformulation optimization is used.

Example 2.10 The data for this example are a pair of diamond shaped shells, shown as hollow circles in Figures 2.13(a) and 2.13(b). The first stage in this example uses the FCM point-prototypes algorithm to find shell parameters that fit the data reasonably well. FCM estimates for the shell parameters in this problem correspond to shell centers (the terminal cluster centers $\mathbf{v}_1$ and $\mathbf{v}_2$ found by FCM); and shell radii computed as $r_i = \sqrt{\sum_{k=1}^{n} u_{ik}^m D_{ik} \Big/ \sum_{k=1}^{n} u_{ik}^m}$ for $1 \le i \le 2$ and the terminal FCM partition U. Here D is the 1-norm on $\Re^2$ - i.e., the NISP norm of choice for this problem. We initialized FCM with a partition $U_0 \in M_{hcn}$. The choice of U_0 did not matter in the cases examined, and the standard choice was to simply alternate 1's and 0's in each row of U. This choice is a poor initialization since every other point in each diamond starts out belonging to the wrong cluster.

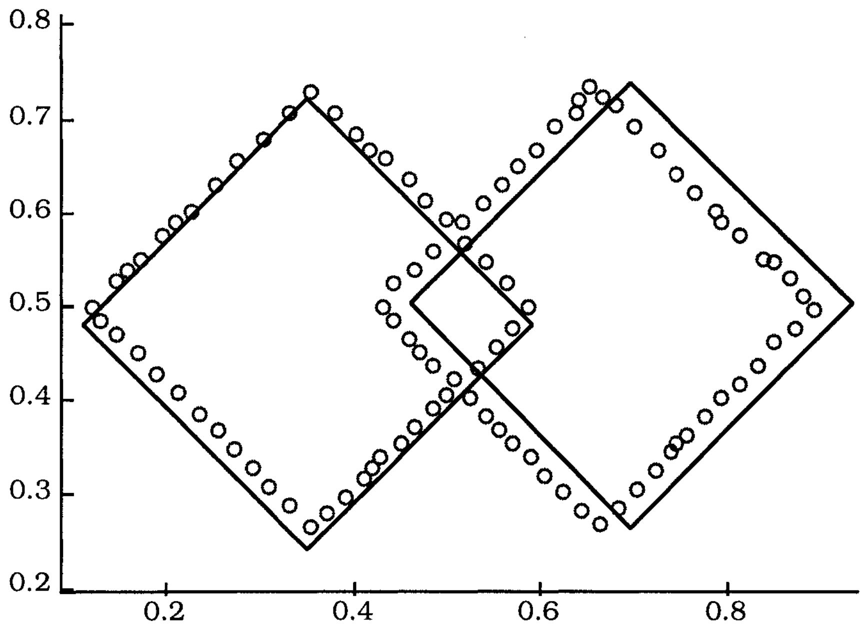


Figure 2.13(a) Stage 1 NISP shells obtained using FCM

In 18 iterations FCM with c = m = 2 and the Euclidean norm for J_2 terminated with $\|U_t - U_{t-1}\|_1 \le 0.001$, approximate cluster centers $\mathbf{v}_1 = \mathbf{v}_{1,t}$ and $\mathbf{v}_2 = \mathbf{v}_{2,t}$ and a fuzzy partition $U = U_t$. The terminal cluster centers were used to calculate the squared distances $D_{ij} = \|\mathbf{x}_j - \mathbf{v}_i\|_1^2$, for i=1,2 and j=1,...,n, which were then used with U to calculate the initial shell radii r_1 and r_2. The stage one shell estimates are shown in Figure 2.13(a). They fit the overlapping diamonds pretty well, but further processing with NISP-AO will improve the fit.

In stage 2 the fuzzy c-means shell estimates from stage 1 are used to initialize an optimization routine (we used the function "fmins" from the MATLAB optimization toolbox) that is then applied to the fuzzy reformulation $R_m(\beta) = \sum_{k=1}^{n}\left(\sum_{i=1}^{c}\left|\|\mathbf{x}_k - \mathbf{v}_i\|_1 - r_i\right|^{2/(1-m)}\right)^{1-m}$ of (2.24a) using the 1-norm as the shell inducing norm with m= 2. The final results produced using this two stage approach is shown in Figure 2.13(b). The 1-norm induced shell prototypes (the two diamonds) shown in Figure 2.13(b) fit the data quite well.

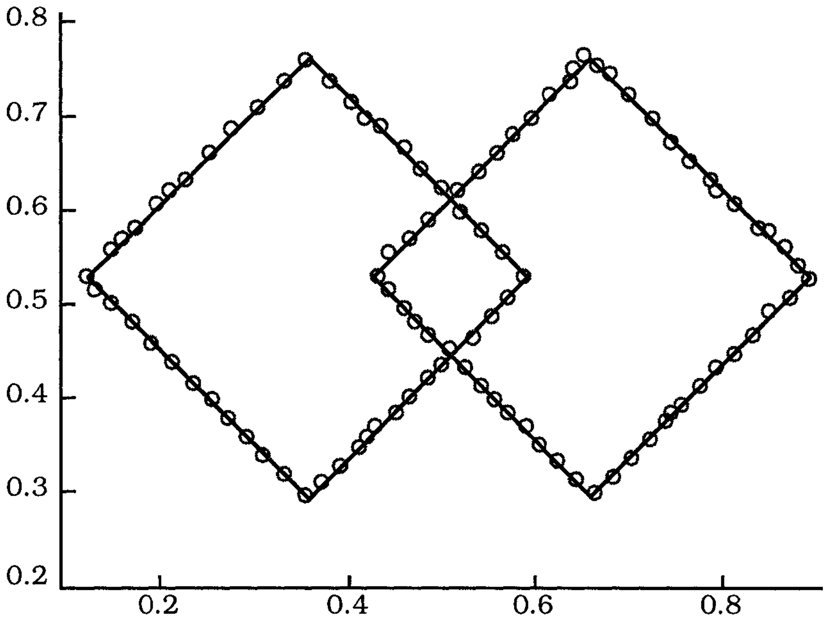


Figure 2.13(b) Stage 2 NISP shells obtained by fmins on R_m

If the matrix A in equation (1.6) or the power q in the Minkowski norm in equation (1.11) are considered part of the prototype along with $\mathbf{v}_i$ and r_i, it can be shown that the shapes generated by the NISP model using these two families of norms are *superquadrics* (Solina and Bajczy, 1990). We will discuss a recent model due to Hoeppner (1997) in chapter 5 - the *fuzzy c-rectangular shells* model - that is very similar to and in some ways slightly more general than the NISP model. To appreciate how similar the two models are, peek ahead to Figure 5.39, and compare it to Figure 2.12.

G. Regression models as prototypes

Another family of objective functions that use non-point prototypes was introduced in Hathaway and Bezdek (1993). They called this family *fuzzy c-regression models* (FCRM). Minimization of particular objective functions in the family yields *simultaneous* estimates for the parameters of c regression models; and a fuzzy c-partitioning of the data.

Let $S = \{(\mathbf{x}_1, \mathbf{y}_1),\ldots,(\mathbf{x}_n, \mathbf{y}_n)\}$ be a set of data where each independent observation $\mathbf{x}_k \in \Re^s$ has a corresponding dependent observation $\mathbf{y}_k \in$

$\Re^t$. In the simplest case we assume that a single functional relationship between **x** and **y** holds for all the data in S. In many cases a statistical framework is imposed on this problem to account for measurement errors in the data, and a corresponding optimal solution is sought. Usually, the search for a "best" function is partially constrained by choosing the functional form of **f** in the assumed relationship

$$\mathbf{y} = \mathbf{f}(\mathbf{x}; \mathbf{B}) + \boldsymbol{\varepsilon} \quad , \tag{2.61}$$

where $\mathbf{B} \in \Omega \subset \Re^k$ is the vector of parameters that define **f** to be determined, and $\boldsymbol{\varepsilon}$ is a random vector with mean vector $\boldsymbol{\mu} = \mathbf{0} \in \Re^t$ and covariance matrix Σ. The definition of an optimal estimate of **B** depends on distributional assumptions made about $\boldsymbol{\varepsilon}$, and the set Ω of feasible values of **B**. This type of model is well known and can be found in most texts on multivariate statistics.

The model considered by Hathaway and Bezdek (1993) is known as a *switching regression* model (Hosmer, 1974, Kiefer, 1978, Quandt and Ramsey, 1978, De Veaux, 1989). We assume S to be drawn from c models

$$\mathbf{y} = \mathbf{f}_i(\mathbf{x}; \boldsymbol{\beta}_i) + \boldsymbol{\varepsilon}_i \quad , 1 \le i \le c, \tag{2.62}$$

where $\boldsymbol{\beta}_i \in \Omega_i \subset \Re^{k_i}$, and $\boldsymbol{\varepsilon}_i$ is a random vector with mean vector $\boldsymbol{\mu}_i = \mathbf{0} \in \Re^t$ and covariance matrix Σ_i. Good estimates for the parameters $\mathbf{B} = \{\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c\}$ are desired as in the single model case. Here, as in (2.24), $\boldsymbol{\beta}_i$ is the set of parameters for the i-th prototype, which in this case is the regression function $\mathbf{f}_i$. However, we have the added difficulty that S is unlabeled, That is, for a given datum $(\mathbf{x}_k,\mathbf{y}_k)$, it is not known which model from (2.62) applies.

One approach for estimating the parameters $\{\beta_{ij}\}$ is to use the GMD-AO algorithm (Table 2.4).The approach taken here is more akin to fuzzy cluster analysis than statistics. The main problem is that the data in S are unlabeled, so numerical methods for estimation of the parameters almost always lead to equations which are coupled across classes. If S were partitioned into c *crisp* subsets corresponding to the regimes represented by the models in (2.62), then estimates for $\{\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c\}$ could be obtained by simpler methods. One alternative to using GMD-AO is to first find a crisp c-partition of S using an algorithm such as HCM; and then solve c separate single-model problems using S_i with (2.61). This is usually not done because it may not explain the data structure properly. The effectiveness using of (2.61) for each crisp cluster depends on how accurate the crisp clusters are.

Hathaway and Bezdek formulated the two problems (partitioning S and estimating $\{\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c\}$, the parameters of the prototype functions $\{\mathbf{f}_i(\mathbf{x};\ \boldsymbol{\beta}_i)\}$) so that a simultaneous solution could be attempted. A clustering criterion is needed that explicitly accounts for both the form of the regression models as well as the need to partition the unlabeled data so that each cluster of S is well-fit by a single model from (2.62). For the switching regression problem we interpret u_{ik} as the importance or weight attached to the extent to which the model value $\mathbf{f}_i(\mathbf{x}_k;\ \boldsymbol{\beta}_i)$ matches $\mathbf{y}_k$. Crisp memberships (0's and 1's) in this context would place all of the weight in the approximation of $\mathbf{y}_k$ by $\mathbf{f}_i(\mathbf{x}_k;\ \boldsymbol{\beta}_i)$ on one class for each k. But fuzzy partitions enable us to represent situations where a data point fits several models equally well, or more generally, may fit all c models to varying degrees.

The measure of similarity in (2.24c) for the FCRM models is some measure of the quality of the approximation of $\mathbf{y}_k$ by each $\mathbf{f}_i$: for $1 \le i \le c$; $1 \le k \le n$, define

$$E_{ik}(\boldsymbol{\beta}_i) \equiv \text{measure of error in } f_i(\mathbf{x}_k;\boldsymbol{\beta}_i) \approx \mathbf{y}_k \quad . \tag{2.63}$$

The most common example for such a measure is the vector norm $E_{ik}(\boldsymbol{\beta}_i) = \|\ \mathbf{f}_i(\mathbf{x}_k;\ \boldsymbol{\beta}_i) - \mathbf{y}_k\ \|$. The precise nature of (2.63) can be left unspecified to allow a very general framework. However, all choices for E_{ik} are required to satisfy the following *minimizer property*. Let $\mathbf{a} = (a_1,a_2,\ldots,a_n)^T$ with $a_i \ge 0\ \forall\ i$, and $\mathbf{E}_i(\boldsymbol{\beta}_i) = (E_{i1}(\boldsymbol{\beta}_i),\ldots,E_{in}(\boldsymbol{\beta}_i))^T$, $1 \le i \le c$. We require that each of the c Euclidean dot products

$$\langle \mathbf{a}, \mathbf{E}_i(\boldsymbol{\beta}_i) \rangle\ ;\ 1 \le i \le c \tag{2.64}$$

have a global minimum over Ω_i, the set of feasible values of $\boldsymbol{\beta}_i$. The general family of FCRM objective functions is defined, for $U \in M_{fcn}$ and $(\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c) \in \Omega_1 \times \Omega_2 \times \cdots \times \Omega_c \in \Re^{k_1} \times \Re^{k_2} \times \cdots \times \Re^{k_c}$, by the fuzzy instance of (2.24a) with (2.63) that satisfy (2.64) inserted into (2.24c) - that is, $D_{ik}^2 = E_{ik}(\boldsymbol{\beta}_i)$. The basis for this approach is the belief that minimizers (U, **B**) of J_m(U, **B**) are such that U is a reasonable fuzzy partitioning of S and that $\{\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c\}$ determine a good switching regression model.

Minimization of (2.24a) under the assumptions of FCRM can be done with the usual AO approach whenever grouped coordinate minimization with analytic formulae is possible. Specifically, given data S, set m > 1, choose c parametric regression models $\{\mathbf{f}_i(\mathbf{x};\boldsymbol{\beta}_i)\}$, and choose a measure of error $\mathbf{E} = \{E_{ik}\}$ so that $E_{ik}(\boldsymbol{\beta}_i) \geq 0$ for i and k, and for which the minimizer property defined by (2.64) holds. Pick a termination threshold $t_\varepsilon > 0$ and an initial partition $U_0 \in M_{fcn}$. Then for $r = 0,1,2,\ldots$: calculate values for the c model parameters $\boldsymbol{\beta}_i^{(r)}$ that globally minimize (over $\Omega_1 \times \Omega_2 \times \times \Omega_c$) the restricted objective function $J_m(U_r, \boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c)$. Update $U_r \rightarrow U_{r+1} \in M_{fcn}$ with the usual FCM update formula (2.7a). Finally, compare either $\|U_{r+1} - U_r\|$ or $\|\mathbf{B}_{r+1} - \mathbf{B}_r\|$ in some convenient matrix norm to a termination threshold ε. If successive estimates are less than ε, stop; otherwise set r = r+1 and continue.

Solution of the switching regression problem with mixture decomposition using the GMD-AO algorithm can be regarded as the same optimization approach applied to the objective function $L(U,\gamma_1,\ldots,\gamma_c) = \sum_{k=1}^{n}\sum_{i=1}^{c} u_{ik}(E_{ik}(\gamma_c) + \ln(u_{ik}))$, see equation (11) of Bezdek et al. (1987a). In this case, the $\{\gamma_i\}$ are the regression model parameters (the $\{\beta_i\}$), plus additional parameters such as means, covariance matrices and mixing proportions associated with the c components of the mixture. Minimization with respect to **B** is possible since the measure of error satisfies the minimizer property and J_m can be rewritten to look like a sum of functions of the form in (2.64).

For a specific example, suppose that t=1, and for $1 \leq i \leq c$: $k_i = s$, $\Omega_i = \Re^s$, $f_i(\mathbf{x}_k;\boldsymbol{\beta}_i) = (\mathbf{x}_k)^T\boldsymbol{\beta}_i$, and $E_{ik}(\boldsymbol{\beta}_i) = (y_k - (\mathbf{x}_k)^T\boldsymbol{\beta}_i)^2$. Then $J_m(U, \mathbf{B})$ is a fuzzy, multi-model extension of the least squares criterion for model-fitting, and any existing software for solving weighted least squares problems can be used to accomplish the minimization. The explicit formulae for the new iterates $\boldsymbol{\beta}_i^{(r)}$, $1 \leq i \leq c$, can be easily derived using calculus. Let X denote the matrix in $\Re^{ns}$ having $\mathbf{x}_k$ as its k-th row; Y denote the vector in $\Re^n$ having y_k as its k-th component; and D_i denote the diagonal matrix in $\Re^{nn}$ having $(u_{ik,r})^m$ as its k-th diagonal element. If the columns of X are linearly independent and $u_{ik,r} > 0$ for $1 \leq k \leq n$, then

$$\boldsymbol{\beta}_i^{(r)} = [X^T D_i X]^{-1} X^T D_i Y \quad . \qquad (2.65)$$

If the columns of X are not linearly independent, $\boldsymbol{\beta}_i^{(r)}$ can still be calculated directly, but techniques based on orthogonal factorizations of X should be used. Though it rarely occurs in practice, $u_{ik}^{(r)}$ can equal 0 for some values of k, but this will cause singularity of $[X^TD_iX]$ only in degenerate (and extremely unusual) cases. As a practical matter, $\boldsymbol{\beta}_i^{(r)}$ in (2.65) will be defined throughout the iteration if the columns of X are linearly independent.

Global convergence theory from Zangwill (1969) can be applied for reasonable choices of $E_{ik}(\boldsymbol{\beta}_i)$ to show that any limit point of an iteration sequence will be a minimizer, or at worst a saddle point, of $J_m(U,\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_c)$. The local convergence result in Bezdek et al. (1987a) states that if the error measures $\{E_{ik}(\boldsymbol{\beta}_i)\}$ are sufficiently smooth and a standard convexity property holds at a minimizer (U, **B**) of J_m, then any iteration sequence started with U_0 sufficiently close to U will converge to (U, **B**). Furthermore, the *rate* of convergence of the sequence will be q-linear.

The level of computational difficulty in minimization of J_m with respect to **B** is a major consideration in choosing the particular measure of error $E_{ik}(\boldsymbol{\beta}_i)$. The best situation is when a closed form solution for the new iterate $\boldsymbol{\beta}_i^{(r)}$ exists such as in the example at (2.65). Fortunately, in cases where the minimization must be done iteratively, the convergence theory in Hathaway and Bezdek (1991) shows that a single step of Newton's method, rather than exact minimization, is sufficient to preserve the local convergence results. The case of inexact minimization in each half step is further discussed and exemplified in Bezdek and Hathaway (1992) in connection with the FCS algorithm of Davé.

Example 2.11 This example illustrates the use of FCRM to fit c = 2 quadratic regression models. The quadratic models are of the form

$$y = \beta_{11} + \beta_{12}x + \beta_{13}x^2 \quad \text{, and} \qquad (2.66a)$$

$$y = \beta_{21} + \beta_{22}x + \beta_{23}x^2 \quad . \qquad (2.66b)$$

The four data sets A, B, C and D specified in Table 2.6 were generated by computing y from 2.66(a) or 2.66(b) at n/2 fixed, equally spaced x-values across the interval given in Column 3 of Table 2.6. This resulted in sets of n points (which we pretend are unlabeled), half of which were generated from each of the two quadratics specified by the parameters in Columns 4 and 5 of Table 2.6. These four data sets are scatterplotted in Figure 2.14.

Table 2.6 Data from the quadratic models $y = \beta_{i1} + \beta_{i2}x + \beta_{i3}x^2$

	n	x-interval	$\boldsymbol{\beta}_1$	$\boldsymbol{\beta}_2$
A	46	[5, 27.5]	$\boldsymbol{\beta}_{1A} = (21, -2, 0.0625)$	$\boldsymbol{\beta}_{2A} = (-5, 2, -0.0625)$
B	28	[9, 22.5]	$\boldsymbol{\beta}_{1B} = (21, -2, 0.0625)$	$\boldsymbol{\beta}_{2B} = (-5,2,-0.0625)$
C	30	[9, 23.5]	$\boldsymbol{\beta}_{1C} = (18, -1, 0.03125)$	$\boldsymbol{\beta}_{2C} = (-2, 1, -0.03125)$
D	46	[10.5, 21.75]	$\boldsymbol{\beta}_{1D} = (172, -26, 1)$	$\boldsymbol{\beta}_{2D} = (364, -38, 1)$

FCRM iterations seeking two quadratic models were initialized at a pair of quadratics with parameters $\boldsymbol{\beta}_{1,0} = (-19, 2, 0)$; $\boldsymbol{\beta}_{2,0} = (-31, 2, 0)$. Since the coefficients of the x^2 terms are zero, the initializing models are the dashed lines shown in Figure 2.14. FCRM run parameters were c = m = 2 and $E_{ik}(\boldsymbol{\beta}_i) = (y_k - \beta_{i1} - \beta_{i2}x_k - \beta_{i3}x_k^2)^2$. Iteration was stopped as soon as the maximum change in the absolute value of successive pairs of estimates of the six parameter values for that model was found to be less than or equal to $\varepsilon = .00001$, that is, $\|\mathbf{B}_{r+1} - \mathbf{B}_r\|_\infty \le 0.00001$.

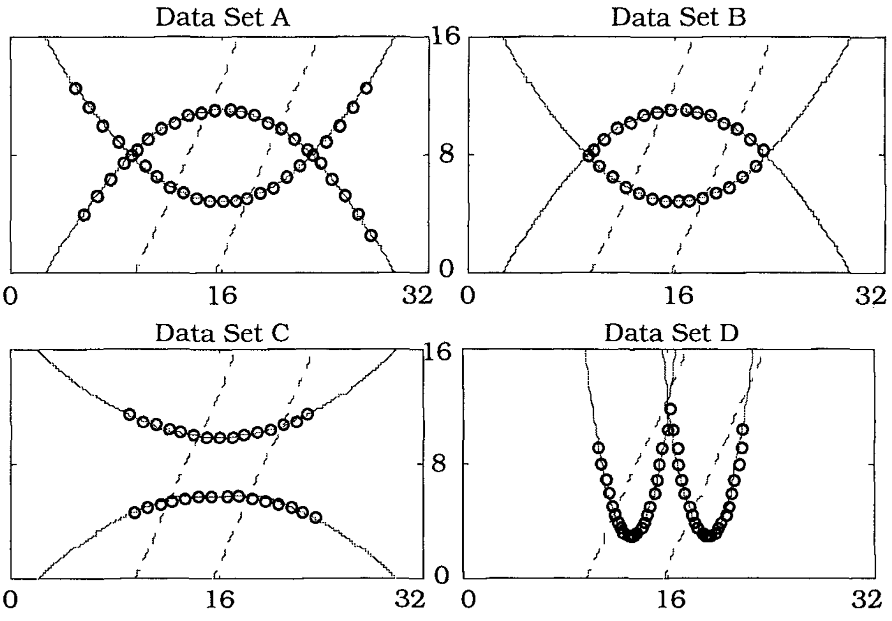


Figure 2.14 Initial (dashed) and terminal (solid) models

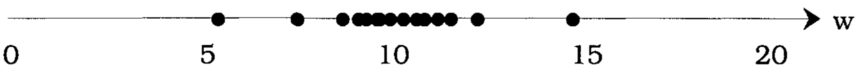
Figure 2.14 shows the initial (dashed lines) and terminal regression models FCRM found when started at the given initialization. The initializing lines were neither horizontal nor vertical - they were inclined to the axes of symmetry of the data in every case. This initialization led to successful termination at the true values of the generating quadratics very rapidly (6-10 iterations) for all four data sets. The terminal fits to the data are in these four cases good (accurate to essentially machine precision). In the source paper for this example FCRM detected and characterized the quadratic models generating these four data sets correctly in 9 of 12 attempts over three different pairs of initializing lines.

FCRM differs from quadric c-shells most importantly in the sense that the regression functions - which are the FCRM prototypes - need not be recognizable geometric entities. Thus, data whose functional dependency is much more complicated than hyperquadric can (in principle at least) be accommodated by FCRM. Finally, FCRM explicitly recognizes functional dependency between grouped subsets of independent and dependent variables in the data, whereas none of the previous methods do. These are the major differences between FCRM and all the other non-point prototype clustering methods discussed in this section. In the terminology of Section 4.6, FCRM is really more aptly described as a "system identification" method, the system being the mixed c-regression models.

H. Clustering for robust parametric estimation

The term "robust clustering", sometimes used to describe the algorithms in this subsection, is somewhat of a misnomer, since it seems to promise a clustering method that is somehow "more robust" than, for example, the c-means models. However, the algorithms in this subsection do not look for clusters in the same circumstances as our previous models. Here, we develop methods that can be used as tools to make (more) robust estimates of statistical parameters than, say, GMD-AO could, when certain assumptions are made about the data. Consequently, this topic fits equally well into the framework of subsection 4.6.G, where we discuss the use of clustering as a tool for estimating parameters of two kinds of fuzzy systems that are used for function approximation. This is really the aim of robust clustering too - estimation of model parameters that provide good approximations to unknown parameters of the assumed model.

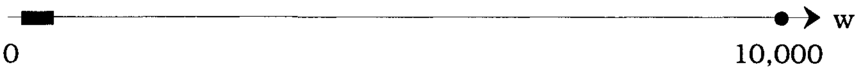
To understand the intent of robust statistics, imagine that you are measuring electronically the weights (w) of Chinook Salmon as they are being taken out of a fishing net. The weights of all Chinook Salmon will almost certainly resemble a normal distribution $n(\mu,\sigma^2)$, and you hope to estimate the parameters of this distribution using the collected samples. Suppose the population mean weight of all the fish of this species (excluding fish less than 6 inches long) is 10 pounds, with a standard deviation in the population of 1 pound. Then your expectation is that about 95% of all the measured weights will fall in the weight interval [8, 12], accounting for two standard deviations on either side of the mean. You will have no trouble visualizing the scatterplot of the first 10,000 samples of this process along the real line - it should look much like Sketch A.



Sketch A 10,000 samples of $n(10,1)$

The probability of seeing even one observation close to 5 or 15 in this situation is so small that the observations shown in sketch A are already far-fetched. If you ran the HCM algorithm with the Euclidean norm on the data in Sketch A with c = 1, all the points would be put unequivocally into one cluster. What estimate would you get for the cluster center? Since p = 1, $\|x_k - v\|^2 = (x_k - v)^2$ and $u_{1k} = 1 \forall k$, the cluster center estimated with (2.6b) would be $v = \sum_{k=1}^{10,000} x_k / 10,000$, the arithmetic mean of the 10,000 points. This is exactly what you want, and the estimate would be very close to 10. Of course, you can compute this statistic without clustering, but this illustrates how clustering can be used in statistical estimation.

Now suppose the voltage to the electronic scale that is measuring the w's suddenly jumps, causing the sensor to record just one measurement of, say, w = 10,000 (this is a fish the authors would like to catch!). Most estimates we might make for μ and σ from the data collected would be effected dramatically by this single mistake, since now the situation in sketch A becomes that of sketch B.



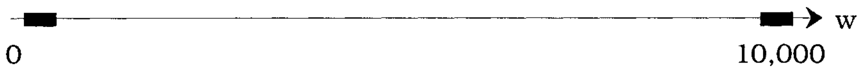
Sketch B 9,999 samples of $n(10,1)$ **+ one sample with value 10,000**

If you ran HCM with c = 1 on the data in Sketch B, the estimate of the mean would be pulled far to the right, as it would if you simply computed the new arithmetic mean of the data. This sensitivity to "noise", or "outliers", or whatever you prefer to call unusual perturbations of the data, is termed lack of robustness. In this example, we say that the statistic used (here the arithmetic mean) has a breakdown point of 1/n = 1/10,000 - that is, 1 bad point in n samples can give an estimate that is arbitrarily far from the true value.

According to Huber (1981), a robust procedure for statistical parametric estimation can be characterized by the following: (1) it should have a reasonably good efficiency (statistically) at the assumed model, (2) small deviations from the model assumptions should impair the performance only by a small amount, and (3) larger deviations from the model assumptions should not cause a catastrophe.

Statistics that can overcome sensitivity to outliers (to various extents) are called *robust estimators*. For example, if you use the median instead of the mean to estimate μ for the data in Sketch B, you will still obtain a very reasonable estimate, because all but one of the points to the right of the median is very close to μ relative to the one outlier. This estimate can also be obtained by clustering the Sketch B data with HCM if you replace the Euclidean norm in J_1 by the 1-norm. In this case the necessary conditions (2.6) do not apply, and there are a number of alternative methods that find estimates of extreme points of J_1. In particular, the median of the data is known to minimize J_1 in the situation of Sketch B (Kersten, 1995), so again, we can obtain a reasonable estimate of the mean μ, using a clustering algorithm that is robust in this well defined statistical sense. Two things to note: first, we still run the clustering algorithm at $c = 1$, presumably because the physical process here tells us it must be 1 (unless there is a large school of giant Chinooks somewhere, feeding on sperm whales); and second, although we know (or suspect) that the collected samples are contaminated by noise, we don't know which ones are the bad ones.

The question is not "how many clusters are there in sketches A and B" - there are two; rather, the question posed in robust statistics is "how badly will the estimators of the mean and variance of the single distribution we assume produced these samples be affected by the addition of the "noise point" whose value is 10,000. To underscore this more dramatically, suppose 45% of the 10,000 points were "accidentally" recorded at values near 10,000. This would result in the situation shown in Sketch C.



Sketch C 5,500 samples of $n(10,1)$ + 4,500 samples with values near 10,000

From the point of view of clustering, the data in Sketch C have - without question - two visual clusters, and any clustering algorithm we have discussed so far would find these two clusters in short order - *provided we ran it at* $c = 2$. But from the point of view of parametric estimation, if we *knew* (or assumed, anyway) that the data must come from a *single* normal distribution, we would want a method that somehow still produced reasonable estimates for the parameters of $n(10,1)$. The corrupted observations may or may not form a "cluster", but are still perfidious to statistical estimators.

Ordinary statistics such as the average of the 10,000 samples, which in this case would produce an estimate of about 4,500 for the mean, would be unreliable. In fact, the mean can be made arbitrarily far from the "correct" estimate by increasing the values of the "corrupted" observations. On the other hand, in the overdramatized situation depicted in Sketch C, the median will do much better, since the estimate produced by it will not be arbitrarily far from the actual (population) value, no matter how high the values of the corrupted observations are. The median will break down only when the fraction of corrupted samples exceeds 50% - i.e., the breakdown point of the median *is* 50%.

So, this is the problem set out for "robust clustering" : to find reasonable estimates for the model parameters under the assumptions that: (i) the model is known, and (ii) there are (unknown) samples in the data that are aberrant. Fuzzy clustering algorithms have been used to estimate parameters of normal mixtures for quite a while (Bezdek and Dunn, 1975, Bezdek et al., 1985, Gath and Geva, 1989b), but the methods used are "intolerant" to the problem of robust estimation. Non-point prototype clustering algorithms such as fuzzy c-lines (FCL) and fuzzy c-shells (FCS) can be used to estimate lines and curves in unlabeled data, and these algorithms may suffer from the same intolerance to aberrant data. The aim of the techniques discussed in this subsection is to design clustering models (albeit not quite unsupervised) that overcome or at least obviate sensitivity to noise under the specific assumptions just stated.

In robust statistics, the *breakdown point* of an estimator is defined to be the smallest fraction of noise or outlier points that *can* result in arbitrarily large errors in the estimate (Hampel, 1975). (Outliers are misrecorded observations or points otherwise included in data whose values can be arbitrarily distant from the correct ones.) Prototype-based clustering algorithms may be viewed as estimators of prototypes. Therefore, when the prototype estimates corresponding to the global minimum of the objective function *can* have arbitrarily large errors, we may say that the (formulation of the) clustering algorithm breaks down.

The *breakdown point of a clustering algorithm* can be used as a measure its robustness. When there is only one cluster in the data, theoretically the best breakdown point one can be achieve is 0.5 (or

50%). This is because if the noise points "conspire" to form a cluster that is equal in size to the good cluster, and if the noise cluster is arbitrarily far away, then there is no way to *guarantee* that any clustering algorithm will pick the right cluster instead of the spurious one. (If the algorithm picks the wrong one, the estimate will be arbitrarily off.) Similarly, when we have a known number of clusters in the data set, the best breakdown point any clustering algorithm can achieve is n_{min}/n, where n_{min} is the number of points in the smallest "good" cluster (Davé and Krishnapuram, 1997).

Bobrowski and Bezdek (1991) first investigated the use of the 1-norm in the FCM model. Kaufman and Rousseeuw showed that the c-means algorithm can be made more robust by using the 1-norm (see Kaufmann and Rousseeuw, 1990). Kersten (1995) later showed that when the 1-norm is used, the update equation for the cluster centers is the fuzzy median. Davé (1991a) proposed the idea of a *noise cluster* (NC) to deal with noisy data. In this approach, noise is considered to be a separate class, and is represented by a *fictitious* prototype that has a constant distance $\bar{\delta}$ from all the data points. The membership u_{*k} of point $\mathbf{x}_k$ in the noise cluster is defined to be

$$u_{*k} = 1 - \sum_{j=1}^{c} u_{jk} \quad . \tag{2.66}$$

Thus, the membership constraint for the good clusters is effectively relaxed to $\sum_{i=1}^{c} u_{ij} < 1$, a strategy that is very similar to the use of slack variables in other optimization domains. This allows noise points to have arbitrarily small membership values in good clusters. The objective function for the *fuzzy noise clustering* (FNC) model is

$$J_{NC}(U,\mathbf{V};\mathbf{O}) = \underbrace{\sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^m D_{ik}^2}_{J_m(U,\mathbf{V};\mathbf{O})} + \sum_{k=1}^{n} u_{*k}^m \bar{\delta}^2 \quad . \tag{2.67}$$

The second term on the right side of (2.67) corresponds to the weighted sum of distances to the *noise* cluster. The membership update equation in Davé's FNC modification of FCM-AO that replaces necessary condition (2.7a) is, for m > 1 and all i, k

$$u_{ik} = \frac{\left(1/D_{ik}^2\right)^{1/m-1}}{\sum_{j=1}^{c}\left(1/D_{jk}^2\right)^{1/m-1} + \left(1/\bar{\delta}^2\right)^{1/m-1}} \quad . \tag{2.68}$$

Together with necessary condition (2.7b), (2.68) forms an AO pair for the *fuzzy robust clustering* (FRC) algorithm. When the initial prototypes are reasonably close to the actual ones, D_{ik}, i = 1,..., c, in (2.68) will be large for outliers, so the numerator and the first term in its denominator will be small relative to the second term in the denominator. This results in small membership values in the good clusters for the outliers. Davé and Krishnapuram (1997) have shown that the FNC approach is essentially the same as Ohashi's (1984) method. (To our knowledge, Ohashi's work was never published, but a brief description of this method can be found in DeGruijter and McBratney, 1988.)

In the FNC approach, $\tilde{\delta}$ plays a role similar to that of w_i in PCM (see (2.5)). PCM and FNC can be shown to be equivalent to the M-estimator technique of robust statistics (Huber, 1981). As a result, their asymptotic breakdown point is limited to $1/n_p$, where n_p is the number of parameters to be estimated. Davé and Krishnapuram (1997) discuss the connection between several robust techniques, including the mountain method (Yager and Filev, 1994a); the *generalized minimum volume ellipsoid* (GMVE) algorithm (Jolion et al. 1991), and a method that seeks to *minimize the probability of randomness* (MINPRAN, Stewart, 1995).

The approach in (Frigui and Krishnapuram, 1995, 1996a), discussed later in this section, was the earliest attempt to incorporate a robust statistical loss function into fuzzy clustering. There have been several other methods studied to make FCM more robust (Frigui and Krishnapuram, 1995; Kim et al., 1995; Choi and Krishnapuram, 1996; Nasraoui and Krishnapuram, 1997). The methods in last three papers just mentioned are based on the reformulation theorem, equations (2.23), of Hathaway and Bezdek (1995). All of these algorithms have the potential to achieve the theoretical breakdown point of n_{min}/n.

Recall that the reformulated objective function for FCM is (2.23b):

$$R_m(\mathbf{V},\mathbf{O}) = \sum_{k=1}^{n}\left(\sum_{i=1}^{c} D_{ik}^{1/1-m}\right)^{1-m} = \sum_{k=1}^{n} H_k \quad , \tag{2.69}$$

where $H_k = \left(\sum_{j=1}^{c} D_{jk}^{1/(1-m)}\right)^{1-m}$. H_k is 1/c times the harmonic mean of the distances $\{D_{jk} : j = 1,\ldots,c\}$ when m=2. Since the H_k values (measured from true prototypes) corresponding to outliers are large, the idea is to design the objective function so that its global minimum is achieved when large H_k are discounted or even completely ignored. The objective function of Choi and Krishnapuram (1996) that defines the *robust FCM* (RoFCM) model and whose gradient supplies necessary conditions for the corresponding AO algorithm is

$$R_{RoFCM}(\mathbf{V},\mathbf{O}) = \sum_{k=1}^{n} \rho(H_k) \quad . \tag{2.70}$$

This objective function applies a loss function $\rho(.)$ to each of the H_k's to reduce the effect of outliers (Huber, 1981). The loss function is typically chosen to be linear for small distances and then it saturates for larger distances. The membership update equation for this formulation remains the same as that of the original FCM, i.e., u_{ik} is computed with (2.7a). However, update equation (2.7b) for the cluster centers is replaced by, for m > 1,

$$\mathbf{v}_i = \frac{\sum_{k=1}^{n} \omega_k u_{ik}^m \mathbf{x}_k}{\sum_{k=1}^{n} \omega_k u_{ik}^m}, i = 1,\ldots,c \quad , \tag{2.71}$$

where $\omega_k = \omega(H_k) = d\rho(H_k) / dH_k$ can be interpreted as the degree of "goodness" of point $\mathbf{x}_k$. The RoFCM algorithm is AO of the pair (2.71) and (2.7a). Ideally, for noise points, ω_k should be as low as possible. In robust statistics, the function ω_k is typically chosen as

$$\omega_k = \omega(H_k) = \begin{Bmatrix} 1; & H_k < \gamma \cdot \underset{s}{\text{med}}\{H_s\} \\ 0; & \text{otherwise} \end{Bmatrix} \quad . \tag{2.72}$$

In (2.72) γ is called the *tuning constant*, and is typically between 2 and 8. Note that ω_k must be updated at every iteration because the $\{H_k\}$ change whenever the $\{\mathbf{v}_i\}$ do. Moreover, there is no guarantee that AO achieves the global minimum of (2.70), and other optimization methods may be more effective for some problems.

The objective function of the *fuzzy trimmed c-prototypes* (FTCP) model of Kim et al. (1995, 1996) is

$$R_{FTCP}(\mathbf{V},\mathbf{O}) = \sum_{k=1}^{q} H_{[k]} \quad , \tag{2.73}$$

where $H_{[k]}$ is the k-th item when the quantities H_i, i=1,...,n are arranged in ascending order, and q is a value less than n. The idea here is to place the c prototypes in such a way that the sum of the smallest q H_k's is minimized. If the value of q is set equal to n-n_{min}+1, FTCP will achieve the theoretical breakdown point.

The *fuzzy c-least median of squares* (FCLMS) algorithm (Nasraoui and Krishnapuram, 1997) replaces the summation that appears on the right side of (2.69) with the median. The objective function of FCLMS is

$$R_{FCLMS}(\mathbf{V},\mathbf{O}) = \underset{k}{\text{med}}\{H_k\} \quad . \qquad (2.74)$$

The crisp version of this algorithm minimizes the median of the distances from the points to their closet prototypes. The median can be replaced by the q-th quantile (e. g. $q=n-n_{min}+1$). AO algorithms that heuristically minimize the FTCP and RoFCM functionals can (but are not guaranteed to) achieve a high breakdown point with relatively low computational complexity. However, the AO technique cannot be applied in these two cases, which both require a random (or exhaustive) search procedure. Kim et al. (1996) give a heuristic AO technique to minimize (2.73). A genetic search is used for minimizing the FCLMS functional at (2.74) in (Nasraoui and Krishnapuram, 1997).

Recently, Frigui and Krishnapuram (1996b, 1997) have introduced an algorithm based on *competitive agglomeration* (CA). This algorithm tries to determine the number of clusters in a data set automatically, without the use of an explicit validity measure. (See Section 2.4 for a detailed discussion on cluster validity.) CA combines the advantages of agglomerative and partitional clustering and achieves relative insensitivity to initialization by initially approximating the data set by a large number of small clusters. Agglomerative (hierarchical) clustering (see Section 3.3) has the advantage that it is insensitive to initialization and local minima, and that the number of clusters need not be specified. However, one cannot incorporate *a priori* information about the shape and size of clusters, as can be done in partitional prototype-based clustering. Agglomerative algorithms produce a nested sequence of partitions (dendrograms), and they are static in the sense that data points that are committed to a cluster in early stages cannot move to another cluster. In contrast, partitional prototype-based clustering is dynamic. The fuzzy CA model uses the following objective function, which seems to combine the advantages of both paradigms

$$J_{CA}(U,\mathbf{V};\alpha) = \underbrace{\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ik}^2 D_{ik}^2}_{J_2(U,\mathbf{V};\mathbf{0})} - \alpha\sum_{i=1}^{c}\left[\sum_{k=1}^{n} u_{ik}\right]^2 \quad . \qquad (2.75)$$

This objective function is minimized subject to $\sum_{i=1}^{c} u_{ik} = 1$, and α is a user defined constant. The first term in (2.75) is J_m at (2.5) with m=2 and $\mathbf{w} = \mathbf{0}$. It represents the sum of fuzzy intracluster distances, allows us to obtain compact clusters and is minimized when c=n. The second term (including the minus sign) is minimized when all good data points are lumped into one cluster. Thus, conceptually (2.75) tries to find a balance between c=n and c=1, and thereby attempts to partition the data set into the smallest possible number of compact clusters. Using LaGrange multipliers, it can be shown that the membership update equation for AO of the function at (2.75) is given by

$$u_{ik} = u_{ik}^{FCM} + u_{ik}^{Bias} \quad , \qquad (2.76)$$

where u_{ik}^{FCM} is the FCM membership with (2.7a) at m = 2, i.e.,

$$u_{ik}^{FCM} = \left[\sum_{j=1}^{c} \left(\frac{D_{ik}^2}{D_{jk}^2} \right) \right]^{-1} \quad , \qquad (2.77)$$

and u_{ik}^{Bias} is the bias membership given by

$$u_{ik}^{Bias} = \frac{\alpha}{D_{ik}^2}(N_i - \overline{N}_k) \quad . \qquad (2.78)$$

In (2.78) $N_i = \sum_{k=1}^{n} u_{ik}$ is the cardinality of cluster i and $\overline{N}_k$ is the weighted average of cardinalities of all clusters (from the point of view of $\mathbf{x}_k$),

$$\overline{N}_k = \frac{\sum_{i=1}^{c} \left(1 / D_{ik}^2\right) N_i}{\sum_{i=1}^{c} \left(1 / D_{ik}^2\right)} \quad . \qquad (2.79)$$

The second term in (2.76) can be either positive or negative, and it allows strong clusters to agglomerate and weak clusters to disintegrate. CA is usually initialized by applying FCM to X with a large value of c to find an initial U and **V**. The value of c is continually updated in CA as clusters become extinct. After the memberships are updated, if the cardinality of a cluster falls below a specified threshold, the prototype corresponding to that cluster and the corresponding row in U are discarded. When this happens, the memberships are redistributed amongst the remaining clusters according to (2.77). The value of α needs to be initially increased slowly, beginning from α = 0, to encourage agglomeration, and is then gradually reduced. The following "annealing schedule" is recommended for the control of α:

$$\alpha_t = \eta_t \left(\frac{\sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik,t-1}^2 D_{ik,t-1}^2}{\sum_{i=1}^{c} \left[\sum_{k=1}^{n} u_{ik,t-1}^2 \right]^2} \right) \text{, where} \tag{2.80}$$

$$\eta_t = \begin{Bmatrix} \eta_0 e^{-|t_0 - t|/\tau} ; & t \geq 1 \\ 0 \qquad ; & t = 0 \end{Bmatrix} . \tag{2.81}$$

In (2.80) and (2.81) α, η, u_{ik}, and D_{ik} are shown as functions of iteration number t. Values for η_0, t_0 and τ are typically 1, 5 and 10 respectively. Equation (2.81) shows that η_t increases until $t = t_0$, and then decays towards zero.

The CA technique can potentially find clusters of various types if we use appropriate prototypes and distance measures in the first term of (2.75). Since the second term in (2.75) does not involve prototypes, the update equations for the prototype parameters are the same as those in the corresponding fuzzy clustering algorithms that do not use the second term.

Frigui and Krishnapuram (1995) present a robust clustering algorithm called the *robust c-prototypes* (RCP) based on the M-estimator. This algorithm uses the objective function $J_{RCP}(U, \mathbf{B}) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik}^2 \rho_i(D_{ik}^2)$, where ρ_i is the loss function for cluster i. Each cluster in RCP has its own loss function, as opposed to RoFCM in (2.70), which has only one loss function for all c clusters. Davé and Sen (1998) have shown that with suitable modifications, FNC (see equation (2.67)) can be made to behave like RCP.

CA can be made robust (Frigui and Krishnapuram, 1996b) by incorporating the RCP approach into (2.75), resulting in the objective function

$$J_{RCA}(U, \mathbf{B}; \mathbf{w}, \alpha) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik}^2 \rho_i(D_{ik}^2) - \alpha \sum_{i=1}^{c} \left[\sum_{k=1}^{n} w_{ik} u_{ik} \right]^2 . \tag{2.82}$$

Thus, the objective function for *robust CA* (RCA) applies a loss function $\rho_i(*)$ to the squared distances to reduce the effect of outliers (Huber, 1981). However, unlike RoFCM, which associates only one weight with each point, RCA uses c robust (possibilistic) weights with each point, where $w_{ij} \in [0, 1]$ is the typicality of $\mathbf{x}_k$ with respect to cluster i. As is customary in robust statistics, the robust weights are related to the loss function via $w_{ik} = w_i(D_{ik}^2) = d\rho_i(D_{ik}^2) / dD_{ik}^2$.

If a point $\mathbf{x}_j$ is an outlier, the weights $\{w_{ij}\}$ will be low for the proper choice of $\rho_i(*)$, and the second term in (2.82) will effectively ignore the contribution of such points. Thus, the second term in (2.82) can be interpreted as the sum of squares of robust cardinalities. The memberships u_{ik}^{FCM} and u_{ik}^{Bias} are now given by

$$u_{ik} = \left[\sum_{j=1}^{c} \left(\frac{\rho_i(D_{ik}^2)}{\rho_j(D_{jk}^2)} \right)^{1/(m-1)} \right]^{-1} \quad ; \text{and} \tag{2.83}$$

$$u_{ik}^{Bias} = \frac{\alpha(N_i - \overline{N}_k)}{\rho_i(D_{ik}^2)} \quad . \tag{2.84}$$

where $N_i = \sum_{k=1}^{n} w_{ik}u_{ik}$ is the robust cardinality of cluster i, and $\overline{N}_k$ is the weighted average of robust cardinalities of all clusters given by

$$\overline{N}_k = \frac{\sum_{i=1}^{c} \left(1/\rho_i(D_{ik}^2)\right) N_i}{\sum_{i=1}^{c} \left(1/\rho_i(D_{ik}^2)\right)} \quad . \tag{2.85}$$

The prototype update equation for prototype $\boldsymbol{\beta}_i$ of cluster i (which could be a scalar, vector or a matrix), can be obtained from the following necessary condition:

$$\frac{dJ_{RCA}(U,\mathbf{B};\mathbf{w},\alpha)}{d\boldsymbol{\beta}_i} = \sum_{i=1}^{n} u_{ik}^2 \frac{d\rho_i}{dD_{ik}^2} \frac{dD_{ik}^2}{d\boldsymbol{\beta}_i} = \sum_{i=1}^{n} u_{ik}^2 w_{ik} \frac{dD_{ik}^2}{d\boldsymbol{\beta}_i} = \mathbf{0}. \tag{2.86}$$

A proper loss function $\rho_i(*)$ is needed for this algorithm to get good results. An alternative to simply guessing $\rho_i(*)$ is to estimate $w_i(*)$ from the data at each iteration and then compute $\rho_i(*)$ as the integral of $w_i(*)$. Example 2.12 illustrates this approach.

Example 2.12 Figure 2.15(a) shows a synthetic data set consisting of six Gaussian clusters of varied sizes and orientations. Uniformly distributed noise was added to the data set so that the noise points constitute about 40% of the total points. The distance measure used in this example, $D_{ik}^2 = |\mathbf{C}_i|^{1/p}(\mathbf{x}_k - \mathbf{v}_i)^T \mathbf{C}_i^{-1}(\mathbf{x}_k - \mathbf{v}_i)$, is due to Gustafson and Kessel (see (2.28)). The initial value for c was overspecified as c = 20. RCA-AO was initialized by running 5 iterations of GK-AO on the data; GK-AO was initialized by randomly choosing 20 points in the data for $\mathbf{V}_0$. When $\left\|\mathbf{v}_{i,t} - \mathbf{v}_{i,t-1}\right\| \le 0.001 \, \forall i$, termination occurred.

For this distance measure the update equations for the center and the covariance matrix of cluster i can be shown to be:

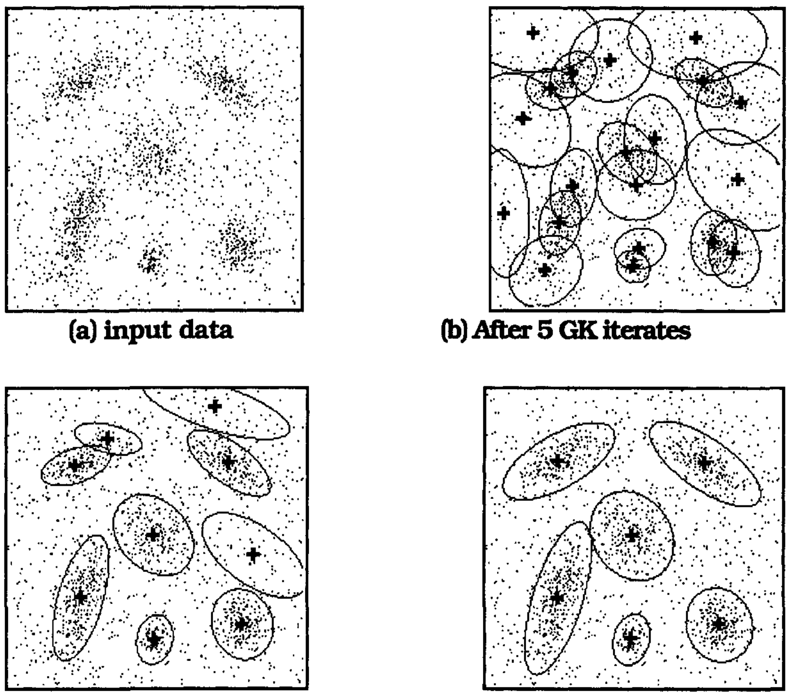
$$\mathbf{v}_i = \frac{\sum_{k=1}^{n} w_{ik} u_{ik}^2 \mathbf{x}_k}{\sum_{k=1}^{n} w_{ik} u_{ik}^2}, \; 1 \le i \le c \qquad ; \text{and} \tag{2.87}$$

$$\mathbf{C}_i = \frac{\sum_{k=1}^{n} w_{ik} u_{ik}^2 (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^T}{\sum_{k=1}^{n} w_{ik} u_{ik}^2}, \; 1 \le i \le c \qquad . \tag{2.88}$$

The weight function is estimated as follows. In each iteration, the fuzzy partition is hardened. Let X_i denote the i-th cluster of the hardened partition, let T_i denote the median of the distances D_{ik}^2 such that $\mathbf{x}_k \in X_i$, and let S_i denote the *median of absolute deviations* (MAD) of D_{ik}^2 for $\mathbf{x}_k \in X_i$. The weight function is chosen such that points within T_i of the prototype have a weight > 0.5, points within $T_i+\gamma S_i$ of the prototype have a weight < 0.5, and points beyond $T_i+\gamma S_i$ have a weight of 0:

$$w_{ik} = w_i(D_{ik}^2) = \begin{cases} 1 - \dfrac{D_{ik}^4}{2T_i^2} & ; D_{ik}^2 \in [0, T_i] \\[2ex] \dfrac{\left[D_{ik}^2 - (T_i + \gamma S_i)\right]^2}{2\gamma^2 S_i^2} & ; D_{ik}^2 \in [T_i, T_i + \gamma S_i] \\[2ex] 0 & ; D_{ik}^2 > T_i + \gamma S_i \end{cases} . \tag{2.89}$$

This weight function (softly) rejects 50% of the points within each component while updating the prototype parameters. Thus, it can tolerate up to 50% outliers in each component. The loss function $\rho_i(*)$ which is needed to update u_{ik} is obtained by integrating the weight function. Figure 2.15(a) shows the input data, which has six clusters which are visually apparent due to higher local densities than the data distribution over the rest of the square. The initial prototypes, obtained by running the GK algorithm for 5 iterations with c=20, are shown in Figure 2.15(b), where the ellipses enclose points with a Mahalanobis distance less than 9. After 6 iterations of RCA-AO the robust cardinalities of the remaining clusters have dropped below the threshold (=3), so the number of clusters is reduced to 9 as shown in Figure 2.15(c). The final result, after 10 iterations of RCA-AO, is shown in Figure 2.15(d).


(a) input data **(b) After 5 GK iterates**

(c) After 6 RCA iterates **(d) Final RCA result**

Figure 2.15 The robust competitive agglomeration technique

2.4 Cluster Validity

Now that we have some ways to *get* clusters, we turn to the problem of how to validate them. Figure 2.3(a) shows that the criterion driving a clustering algorithm towards an optimal partition sometimes produces a result that is disagreeable at best, and wrong at worst. This illustrates the need for approaches to the problem of cluster validity.

Clustering algorithms $\{\mathcal{C}_i\}$ will produce as many partitions as you have time to generate. Let $\mathcal{P} = \{\mathcal{C}_j(X) = U_j \in M_{pcn} : 1 \le j \le N\}$, where index (j) indicates: (i) clustering X with one $\mathcal{C}$ at various values of c; (ii) clustering X over algorithmic parameters of a particular $\mathcal{C}_i$; or

(iii) applying different $\mathcal{C}$'s to X. *Cluster validity* (problem $\boxed{3}$, Figure 2.1) is an assessment of the relative attractiveness of different U's in $\wp$. The usual approach is computational, and is based on one or more *validity functionals* $\mathcal{V}: D_{\mathcal{V}} \mapsto \Re$, $D_{\mathcal{V}}$ denoting the domain of $\mathcal{V}$, to rank each $U_i \in \wp$.

You may wonder: if the global minimum of, say J_1, cannot produce the clusters you want, then why not directly optimize a validity functional $\mathcal{V}$? First, no model can capture all the properties that "good' clusters might possess, and this of course includes any particular $\mathcal{V}$ we might propose. For example, we seek, from data set to data set, clusters with: compactness, isolation, maximal crispness, density gradients, particular distributions, etc. And more importantly, many of the validity indices that will be discussed do not fit naturally into a well behaved framework for mathematical optimization. So, we use validity measures as an "after the fact" way to gain further confidence in a particular clustering solution.

There are two ways to view clustering algorithms. First, it is possible to regard $\mathcal{C}$ as a parametric estimation method - U and any additional parameters such as **B** in the c-means and c-shells models are being estimated by $\mathcal{C}$ using X. In this case $\mathcal{V}$ is regarded as a measure of goodness of fit of the estimated parameters (to a true but *unknown* set!). This interpretation is usually (but not exclusively) made for validity measures in the context of probabilistic clustering.

The second interpretation of $\mathcal{C}$ is in the sense of exploratory data analysis. When $\mathcal{V}$ assesses U alone (even if the measure involves other parameters such as **B**), $\mathcal{V}$ is interpreted as a measure of the quality of U in the sense of partitioning for substructure. This is the rationale underlying most of the methods discussed in this section.

When $D_{\mathcal{V}} = M_{hcn}$, we call $\mathcal{V}$ a *direct* measure; because it assesses properties of crisp (real) clusters or subsets in X; otherwise, it is *indirect*. When $D_{\mathcal{V}} = M_{hcn} \times \underbrace{\text{other parameters}}_{\text{e.g. prototypes } \mathbf{B}}$, the test $\mathcal{V}$ performs is still direct, but addition of the other parameters is an important change, because these parameters often contain valuable information about cluster geometry (for example, measures that assess how well the prototypes **B** fit the cluster shapes). We call indices that fall into this category *direct parametric* indices.

When U is not crisp, validity measures are applied to an *algorithmic derivative* of X so they are called *indirect* measures of cluster validity. There are both indirect and *indirect parametric* measures of partition quality.

Finally, many validity measures also use X. This is a third important aspect of validity functionals: do they use the vectors in X during the calculation of $\mathcal{V}$? We indicate explicit dependence of $\mathcal{V}$ on X by adding the word *data* when this is the case. Let Ω represent the parameter space for **B**. Table 2.7 shows a classification of validity functionals into six types based on their arguments (domains).

Table 2.7 One classification of validity measures

Type of Index	Variables	Domain $D_{\mathcal{V}}$ of $\mathcal{V}$
Direct	U	M_{hcn}
Direct Parametric	(U, **B**)	$M_{hcn} \times \Omega$
Direct Parametric Data	(U, **B**, X)	$M_{hcn} \times \Omega \times \Re^p$
Indirect	U	$(M_{pcn} - M_{hcn})$
Indirect Parametric	(U, **B**)	$(M_{pcn} - M_{hcn}) \times \Omega$
Indirect Parametric Data	(U, **B**, X)	$(M_{pcn} - M_{hcn}) \times \Omega \times \Re^p$

Choosing c=1 or c=n constitutes rejection of the hypothesis that X contains cluster substructure. Most validity functionals are not equipped to deal with these two special cases. Instead, they concentrate on $2 \le c < n$, implicitly ignoring the important question of whether X has clusters in it at all.

Notation It is hard to choose a notation for validity indices that is both comprehensive and comprehensible. Ordinarily, validation means "find the best c", so the logical choice is to show $\mathcal{V}$ as $\mathcal{V}$(c). But in many cases, c doesn't even appear on the right side of an equation that defines $\mathcal{V}$. X in Table 2.7 is fixed, but U and **B** are functions of c through the algorithm that produces them, so any index that uses either of these variables is implicitly a function of c as well. A notation that indicates functional dependency in a precise way would be truly formidable. For example, the Xie and Beni (1991) index (which can be used to validate the number of clusters found) depends on (U, **B**, X), U and **B** depend on $\mathcal{C}$, the clustering algorithm that produces them, and $\mathcal{C}$ either determines or uses c, the number of clusters represented in U. How would you write the independent variables for this function? Well, we don't know a best way, so we will vacillate between two or three forms that make sense to us and that, we hope, will not confuse you. Dunn's index (Dunn, 1974a), for example, will be written as $\mathcal{V}_D(U;X)$ when we feel it important to show the variables it depends upon, but when the emphasis is on its use in its application context, recognizing the fact that U is a function of c, we will write $\mathcal{V}_D(c)$. The partition entropy defined below depends on both U (and hence c) as well as (a), the base of the logarithmic function chosen: thus, we may use $\mathcal{V}_{PE}(U,c,a)$, $\mathcal{V}_{PE}(U)$ or $\mathcal{V}_{PE}(c)$.

A. Direct Measures

If $U \in M_{hcn}$ is *crisp*, it defines *nonfuzzy* subsets in X, and there are many validity functionals that can be used to assess U. Most direct validity indices are based on measuring some statistical or geometric property that seems plausible as a definition of good clusters in X. Statistical indices tend to be estimators of the goodness of fit of the clusters to an assumed distribution. Usually, cluster free data are assumed to be uniformly or randomly distributed over some sampling window, and statistical indices measure the departure of a proposed set of clusters from this assumption. Geometric indices are based on properties such as cluster volume, cluster density and separation between clusters (or their centroids).

B. Davies-Bouldin Index

Davies and Bouldin (1979) proposed an index that is a function of the ratio of the sum of within-cluster scatter to between-cluster separation. Let $U = \{X_1, \ldots, X_c\}$ be a c-partition of X, so that $\bigcup_i X_i = X$; $X_i \cap X_j = \emptyset$ if $i \neq j$; and $X_i \neq \emptyset\ \forall i$. Since scatter matrices depend on the geometry of the clusters, this index has both statistical and geometric rationales, and is designed to recognize good volumetric clusters.

$$\mathcal{V}_{DB,qt}(c) = \left(\frac{1}{c}\right) \sum_{i=1}^{c} \left[\underbrace{\max}_{j,j \neq i} \left\{ (\alpha_{i,t} + \alpha_{j,t}) \Big/ \left(\left\| \bar{\mathbf{v}}_i - \bar{\mathbf{v}}_j \right\|_q \right) \right\} \right] ;\ t, q \geq 1, \tag{2.90a}$$

$$\alpha_{i,t} = \left(\sum_{\mathbf{x} \in X_i} \left\| \mathbf{x} - \bar{\mathbf{v}}_i \right\|^t \Big/ |X_i| \right)^{1/t} ,\ i=1, \ldots, c,\ t \geq 1 \qquad \text{, and} \tag{2.90b}$$

$$\bar{\mathbf{v}}_i = \sum_{\mathbf{x} \in X_i} \mathbf{x} \,/\, |X_i| ,\ i=1, \ldots, c \qquad . \tag{2.90c}$$

Integers q and t can be selected independently. In (2.90a) $\|*\|_q$ is the Minkowski q- norm. In (2.90b) $\|*\|^t$ is the t-th power of the *Euclidean* norm. For p = q = 2, Davies and Bouldin state that the term $(\alpha_{i,2} + \alpha_{j,2}) \Big/ \left(\left\| \bar{\mathbf{v}}_i - \bar{\mathbf{v}}_j \right\|_2 \right)$ is the reciprocal of Fisher's classical measure of separation between clusters X_i and X_j (Duda and Hart, 1973, p.116). However, it differs from Fisher's criterion by having square roots on each term in the numerator, and by using cardinalities of the crisp clusters in the denominator. In any case, these two criteria share similar geometric rationales.

$V_{DB,qt}(\mathbf{1}_{1\times n})$ is undefined, and $V_{DB,qt}(I_n) = 0$. Since minimum within-cluster dispersion and maximum between-class separation are both desirable, low values of $V_{DB,qt}$ are taken as indicants of good cluster structure. In our classification of validity indices in Table 2.7, $V_{DB,qt}$ is a *direct parametric data* index. As a reminder, this would be formally indicated by writing $V_{DB,qt}$ as a function of U, **V** and X, $V_{DB,qt}(U, \mathbf{V}; X)$. We avoid this cumbersome notation when discussing its use by writing $V_{DB,qt}(c)$.

Araki et al. (1993) proposed a fuzzy generalization of $V_{DB,qt}$ that is explicitly tied to the FCM clustering algorithm. For $U_{FCM} \in M_{fcn}$ and point prototypes **V** generated from X at some value of m>1, they define

$$\hat{\alpha}_{i,t} = \left(\frac{\sum_{k=1}^{n} u_{ik}^{m} \|\mathbf{x}_k - \mathbf{v}_i\|^2}{\sum_{k=1}^{n} u_{ik}^{m}} \right) \quad \text{where } \mathbf{v}_i = \left(\frac{\sum_{k=1}^{n} u_{ik}^{m} \mathbf{x}_k}{\sum_{k=1}^{n} u_{ik}^{m}} \right), i = 1, \ldots c .$$

Notice that the square root is not taken, as it would be in (2.90) for t = 2. Moreover, Araki et al. also use q = 2 in (2.90) without taking the square root.

Substituting $\{\hat{\alpha}_{i,t}\}$and $\{\mathbf{v}_i\}$ for $\{\alpha_{i,t}\}$ and $\{\bar{\mathbf{v}}_i\}$ respectively into (2.90), Araki et al. arrive at a well defined *indirect parametric data* index $V_{DB,22}^{ANW}$ for validation of fuzzy clusters in U. $V_{DB,22}^{ANW}$ is a fuzzy generalization of $V_{DB,qt}$, but cannot be called *the* fuzzy Davies-Bouldin index because of its explicit dependence on FCM. Furthermore, $V_{DB,22}^{ANW}$ does not reduce to $V_{DB,qt}$ when U is crisp.

Araki et al. incorporate $V_{DB,22}^{ANW}$ into FCM by adding an external loop for c = 2 to c = c_{max} to the iteration phase of FCM in Table 2.2. At termination, this outputs the (U, **V**) found by FCM that minimizes $V_{DB,22}^{ANW}$ over candidate pairs generated by FCM for $2 \le c \le c_{max}$. They report that this strategy leads to good segmentations of thermal images.

C. Dunn's index

Dunn (1974a) proposed an index based on geometric considerations that has the same basic rationale as $V_{DB,qt}$ in that both are designed to identify volumetric clusters that are compact and well separated. Let S and T be non empty subsets of $\Re^p$, and let $\delta: \Re^p \times \Re^p \mapsto \Re^+$ be any metric. The standard definitions of the *diameter* Δ_1 of S and the *set distance* $\hat{\delta}_1$ between S and T are

$$\Delta_1(S) = \underbrace{\max}_{\mathbf{x},\mathbf{y} \in S}\{\delta(\mathbf{x},\mathbf{y})\} \quad ; \text{and} \tag{2.91}$$

$$\hat{\delta}_1(S,T) = \underbrace{\min}_{\substack{\mathbf{x} \in S \\ \mathbf{y} \in T}}\{\delta(\mathbf{x},\mathbf{y})\} \quad . \tag{2.92}$$

Dunn defined the *separation index* for the crisp c-partition $U \leftrightarrow \{X_1, \ldots, X_c\}$ of X as

$$V_D(U;X) = \underbrace{\min}_{1 \le i \le c}\left\{ \underbrace{\min}_{\substack{1 \le j \le c \\ j \ne i}}\left\{ \frac{\hat{\delta}_1(X_i, X_j)}{\underbrace{\max}_{1 \le k \le c}\{\Delta_1(X_k)\}} \right\}\right\} \quad . \tag{2.93}$$

The quantity $\hat{\delta}_1(X_i, X_j)$ in the numerator of V_D is analogous to $\|\mathbf{v}_i - \mathbf{v}_j\|_q$ in the denominator of $V_{DB,qt}$; $\hat{\delta}_1(X_i, X_j)$ measures the distance between clusters directly on the points in the clusters, whereas $\|\mathbf{v}_i - \mathbf{v}_j\|_q$ uses the distance between their cluster centers for the same purpose. The use of $\Delta_1(X_k)$ in the denominator of V_D is analogous to $\alpha_{k,t}$ in the numerator of $V_{DB,qt}$; both are measures of scatter volume for cluster X_k. Thus, extrema of V_D and $V_{DB,qt}$ share roughly the same geometric objective: maximizing intercluster distances while simultaneously minimizing intracluster distances. Since the measures of separation and compactness in V_D occur inversely to their appearance in $V_{DB,qt}$, *large* values of V_D correspond to good clusters. Hence, the number of clusters that *maximizes* V_D is taken as the best solution. V_D is not defined on $\mathbf{1}_n$ when c=1 or on I_n when c=n.

Dunn called U *compact and separated* (CS) relative to the (point) metric δ if and only if: for all s, q and r with q≠r, any pair of points **x**, **y** ∈ X_s are closer together (with respect to δ) than any pair **u**,**v** with **u** ∈ X_q and **v** ∈ X_r. Dunn proved that X can be clustered into a compact and separated c-partition with respect to δ if and only if $\underbrace{\max}_{U \in M_{hcn}} \left\{ V_D(c) \right\} > 1$. Dunn's indexis a *direct data* index.

Example 2.13 Table 2.8 shows values of $V_{DB,22}$ and V_D for terminal partitions of $\mathbf{X}_{30}$ produced by HCM-AO using the same protocols as in Example 2.2. Table 2.8 reports values of each index for c=2 to 10. Each column of Table 2.8 is computed by applying the two indices to the same crisp c-partition of X. The highlighted **(bold and shaded)** entries correspond to optimal values of the indices.

Table 2.8 Direct cluster validity for HCM-AO partitions of $\mathbf{X}_{30}$

c	2	3	4	5	6	7	8	9	10
$V_{DB,22}$	0.35	**0.18**	0.48	0.63	0.79	0.87	0.82	0.88	0.82
V_D	0.96	**1.53**	0.52	0.12	0.04	0.04	0.04	0.04	0.04

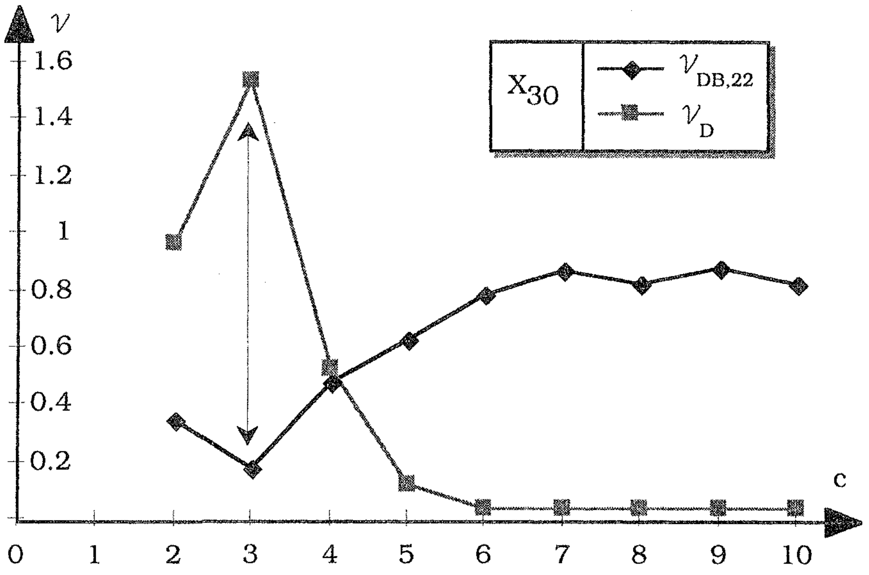


Figure 2.16 V_D and $V_{DB,22}$ from Table 2.8 for HCM-AO on $\mathbf{X}_{30}$

$V_{DB,22}$ indicates c = 3 by its strong minimum value of 0.18. The table shows only two significant digits so ties may appear to occur in it, but there are no ties if four digit accuracy is retained. For this very well separated data set, V_D, which is to be maximized, also gives a very strong indication that c = 3 is the best choice.

Figure 2.16 is a graph of the values in Table 2.8 that shows how strongly c = 3 is preferred by both of these direct indices. Don't expect these (or any other) indices to show such sharp, well-defined behavior on data that do not have such clear cluster structure. Another point: don't forget that the graphs in Figure 2.16 are explicit functions of HCM-AO, the clustering algorithm that produced the partitions being evaluated. You might get different graphs (and infer a different best value of c) simply by changing the initialization, or the norm, or the termination criterion ε, etc. of HCM.

Our next example illustrates the use of V_D and $V_{DB,22}$ on clusters found by HCM-AO in the ubiquitous Iris data (Anderson, 1935). Interestingly, Anderson was a botanist who collected the data, but did not publish their values. Fisher (1936) was apparently the first author to publish the data values, which he used to illustrate the method of linear discriminant analysis. Several scatterplots of Iris are shown in Section 4.3. And finally, please see our comments in the preface about the *real* Iris data.

Example 2.14 Iris has n = 150 points in p = 4 dimensions that represent 3 *physical* clusters with 50 points each. Iris contains observations for 50 plants from each of three different subspecies of Iris flowers, but in the numerical representation in $\Re^4$ of these objects, two of the three classes have substantial overlap, while the third is well separated from the others. Because of this, many authors argue that there are only c=2 geometric clusters in Iris, and so good clustering algorithms and validity functionals should indicate that c=2 is the best choice. Table 2.9 lists the values of V_D and $V_{DB,22}$ on terminal HCM-AO partitions of Iris. All parameters of the runs were as in Example 2.2 except that the initializing vectors were from Iris. Figure 2.17 shows graphs of the values of V_D and $V_{DB,22}$ in Table 2.9.

The Davies-Bouldin index clearly points to c = 2 (our first choice for the correct value), while Dunn's index seems to equally prefer c = 3 and c = 7. To four place accuracy (not shown here), c = 3 is slightly higher, so Dunn's index here would (weakly) indicate the partition corresponding to c = 3. The lesson here is not that one of these answers is right. What is important is that these two indices point to *different "right answers" on the same partitions* of the data.

Table 2.9 Direct cluster validity for HCM-AO partitions of Iris

c	2	3	4	5	6	7	8	9	10
$V_{DB,22}$	**0.47**	0.73	0.84	0.99	1.00	0.96	1.09	1.25	1.23
V_D	0.08	**0.10**	0.08	0.06	0.09	0.10	0.08	0.06	0.06

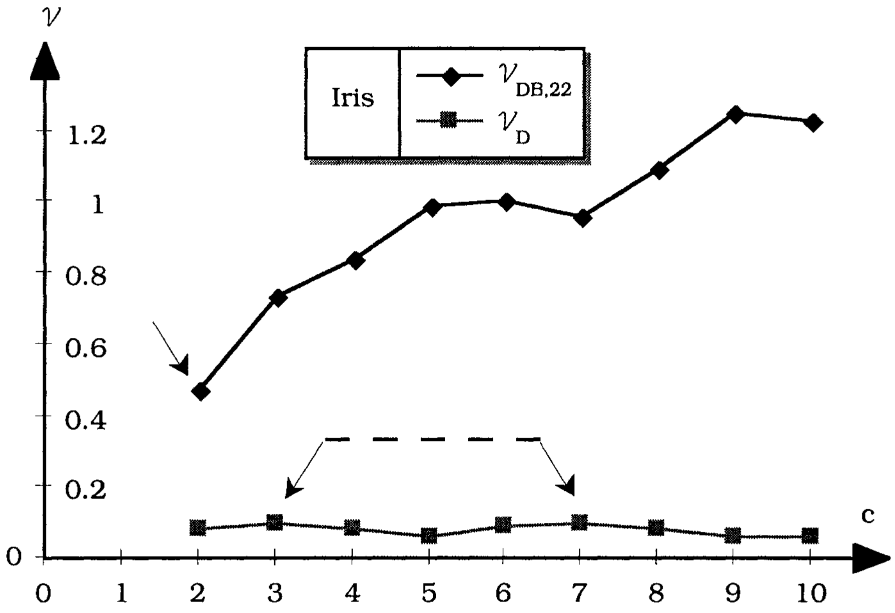


Figure 2.17 V_D and $V_{DB,22}$ from Table 2.9 for HCM-AO on Iris

The numerator and denominator of V_D are both overly sensitive to changes in cluster structure. $\hat{\delta}_1$ can be dramatically altered by the addition or deletion of a single point in either S or T. The denominator suffers from the same problem - for example, adding one point to S can easily scale $\Delta_1(S)$ by an order of magnitude. Consequently, V_D can be greatly influenced by a few noisy points (that is, outliers or inliers to the main cluster structure) in X, and is far too sensitive to what can be a very small minority in the data.

To ameliorate this Bezdek and Pal (1998) generalized V_D by using two other definitions for the diameter of a set and five other definitions for the distance between sets. Let $\hat{\Delta}$ be any positive semi-definite (*diameter*) function on $\wp(\Re^p)$, the *power set* of $\Re^p$. And let $\hat{\delta}$ denote any positive semi-definite, symmetric (*set distance*) function on $\wp(\Re^p) \times \wp(\Re^p)$. The general form of V_D using $\hat{\delta}$ and $\hat{\Delta}$ is

$$V_{\hat{\delta}\hat{\Delta}}(U;X) = V_{\hat{\delta}\hat{\Delta}}(c) = \underbrace{\min}_{1\le i\le c}\left\{\underbrace{\min}_{\substack{1\le j\le c \\ j\ne i}}\left\{\frac{\hat{\delta}(X_i, X_j)}{\underbrace{\max}_{1\le k\le c}\left\{\hat{\Delta}(X_k)\right\}}\right\}\right\} \quad . \tag{2.94}$$

Generally speaking indices from family (2.94) other than V_D show better performance than V_D. The classification of $V_{\hat{\delta}\hat{\Delta}}$ as in Table 2.7 depends on the choices of $\hat{\delta}$ and $\hat{\Delta}$. All of these indices are direct data indices (they all use U and X), and several also use the sample means $\overline{\mathbf{V}}$.

D. Indirect measures for fuzzy clusters

If $U \in (M_{pcn} - M_{hcn})$ is not crisp, there are two approaches to validity assessment. First, direct measures such as $V_{DB,qt}$ and V_D can be applied to any crisp partition derived from U. For example, we can harden U using (2.10) and then assess the resultant crisp partition as in Examples 2.13 and 2.14. Other defuzzifications of U (e.g., α-cuts at different levels) can produce different crisp partitions, and hence, different values for validity indices.

The alternative to hardening U followed by direct validation is validation using some function of the non-crisp partition, and possibly, X as well as other parameters found by $\mathcal{C}$. Almost all of the measures in this category have been developed for fuzzy partitions of X, so we concentrate on this type of index.

Indirect indices that do not involve **B** and X are nothing more than estimates of the fuzziness (or typicality if U is possibilistic) in U. As such, it is *not* possible for them to assess *any* geometric property of either the clusters or prototypes that some algorithm chooses to represent them. Given this, it may surprise you to discover how much effort has gone into the development of indirect measures.

A measure of fuzziness estimates the *average* ambiguity in a fuzzy set in some well-defined sense (Pal and Bezdek, 1994). (Measures of fuzziness and imprecision are covered extensively in Volume 1 of this handbook.) Our discussion is limited to the use of such measures as indicants of cluster validity.

The first measure of fuzziness was the degree of separation between two discrete fuzzy sets $\mathbf{U}_{(1)}$ and $\mathbf{U}_{(2)}$on n elements (Zadeh, 1965):

$$\rho(\mathbf{U}_{(1)},\mathbf{U}_{(2)}) = 1-\left[\bigvee_{k=1}^{n}(u_{1k}\wedge u_{2k})\right] \quad . \tag{2.95}$$

Zadeh used ρ to characterize separating hyperplanes; he did not impose the crisp or fuzzy partitioning constraint $(u_{1k}+u_{2k})=1$ on each pair of values in the vectors $\mathbf{U}_{(1)}$ and $\mathbf{U}_{(2)}$. That is, they were not necessarily fuzzy label vectors (ρ *is* applicable to possibilistic labels, however).

The first attempt to use a measure of fuzziness in the context of cluster validity was discussed by Bezdek (1973), who extended ρ to c fuzzy sets (the rows of U in M_{fcn}) by writing

$$\rho_c(U\in M_{fcn}) = 1-\left[\bigvee_{k=1}^{n}(\bigwedge_{i=1}^{c} u_{ik})\right] \quad . \tag{2.96}$$

ρ_c, which can be interpreted as (1- the "height" of the intersection of the c fuzzy sets), is inadequate for cluster validity. To see this, consider, for odd n,

$$\rho_2\begin{bmatrix}1 & 0.5 & \cdots & 0.5 & 0\\ 0 & 0.5 & \cdots & 0.5 & 1\end{bmatrix} = \rho_2\begin{bmatrix}1 & \cdots & 1 & 0.5 & 0 & \cdots & 0\\ 0 & \cdots & 0 & 0.5 & 1 & \cdots & 1\end{bmatrix} = 0.5. \tag{2.97}$$

In (2.97) the membership 0.5 occurs 2(n-2) times in the first fuzzy 2-partition, but only twice in the second one. The value ρ_c = 0.5 indicates that the two partitions at (2.97) are in some sense (*exactly*, in the sense of ρ_c!) equivalent, but the structure these two partitions portrays is certainly very different. The first partition has one point each in two clusters, and (n-2) shared equally between them, while the second has just one shared point and ((n-1)/2) points in each of two distinct sets.

The failure of ρ_c led to the first pair of (sometimes) useful *indirect* validity measures for U in M_{fcn}, viz., the *partition coefficient* and *partition entropy* of U (Bezdek, 1973).

$$\mathcal{V}_{PC}(U,c) = \frac{1}{n}\left(\sum_{k=1}^{n}\sum_{i=1}^{c}u_{ik}^2\right) = \frac{\|U\|^2}{n} = \frac{tr(UU^T)}{n} \quad ; \text{and} \tag{2.98}$$

$$\mathcal{V}_{PE}(U,c,a) = -\frac{1}{n}\left(\sum_{k=1}^{n}\sum_{i=1}^{c}[u_{ik}\ \ln_a(u_{ik})]\right) \quad . \tag{2.99}$$

In (2.99) a ∈ (1, ∞) is the logarithmic base, and is a direct extension to c-partitions of the fuzzy entropy of Deluca and Termini (1972). Properties of these two indices *as functions of U and c* were studied in Bezdek (1973, 1974b, 1975). For convenience, we drop dependency on c and a. Here are the main results:

$$\mathcal{V}_{PC}(U) = 1 \Leftrightarrow \mathcal{V}_{PE}(U) = 0 \Leftrightarrow U \in M_{hcn} \text{ is crisp; and} \tag{2.100a}$$

$$\mathcal{V}_{PC}(U) = \frac{1}{c} \Leftrightarrow \mathcal{V}_{PE}(U) = \mathrm{Ln}_a(c) \Leftrightarrow U = \left[\frac{1}{c}\right] \doteq \overline{U}. \tag{2.100b}$$

Equation (2.100) shows that $\mathcal{V}_{PC}$ maximizes (and $\mathcal{V}_{PE}$ minimizes) on every crisp c-partition of X. And at the other extreme, $\mathcal{V}_{PC}$ takes its unique minimum (and $\mathcal{V}_{PE}$ takes its unique maximum) at the centroid $U = [1/c] \doteq \overline{U}$ of M_{fcn}. $\overline{U}$ is the "fuzziest" partition you can get, since it assigns every point in X to all c classes with equal membership values 1/c. Observe that both of these indices take extremal values at the unique crisp partitions $\mathbf{1}_{1\times n} = [1\ 1\ \cdots\ 1]$ at c=1 and $I_{n\times n}$, the n×n identity matrix for c = n. Neither of these indices can be used to accept or reject the hypothesis that X contains cluster substructure (i.e., they cannot be used for tendency assessment) because they cannot discriminate between different hard partitions of the data.

The bounds in (2.100a) seem to justify the heuristic validation strategy of, for example, maximizing $\mathcal{V}_{PC}$ over candidate U's to pick the best one, where "best" means nearest to some crisp partition in the sense of the 2-norm of U. This is a weak strategy, however, for several reasons. First, there are an infinite number of different fuzzy partitions that produce any fixed value of $\mathcal{V}_{PC}$ in the open interval (1/c, 1), or of $\mathcal{V}_{PE}$ in the open interval (0, $\mathrm{Ln}_a c$), because a fixed value of either functional can be used to define a hypersphere in $\Re^{cn}$ centered at $[1/c] \doteq \overline{U}$ whose radius gives a surface upon which the fixed value is attained. Consequently, every crisp partition of X - a vertex of the convex hull of the degenerate partition set M_{fcn0} - is equidistant from the surface of the hypersphere! Thus, all these two indices really measure is fuzziness relative to partitions that yield other values of the indices. Second, there are roughly $(c^n/c!)$ crisp matrices in M_{hcn}, and $\mathcal{V}_{PC}$ is constantly 1 ($\mathcal{V}_{PE}$ is constantly 0) on all of them. For example, $\mathcal{V}_{PC}(U_i) = 1$ on:

$$U_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, U_2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, U_3 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, U_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}. \quad (2.101)$$

The first matrix in (2.101) has c = 3 singleton clusters. Each of the other three partitions has only c = 1 cluster. Since the last three matrices put all the data into class 1, 2 or 3, respectively, these are 4 very different partitions of the n=3 objects. But they are all equally valid in the eyes of $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$. Since $\mathcal{V}_{PC} = 1$ ($\mathcal{V}_{PE} = 0$) for *every* U in M_{hcn}, it is misleading to infer that just because $\mathcal{V}_{PC}$ is near 1 (or $\mathcal{V}_{PE}$ is near 0), U is a good clustering of X.

On the other hand, in the context of validation it is clear that when an algorithm produces a partition U that is close to $\overline{U}$, that algorithm is not finding distinct cluster structure. This may be the fault of the algorithm, or the data simply may lack substructure. Consequently, values near the unique minimum of $\mathcal{V}_{PC}$ (or maximum of $\mathcal{V}_{PE}$) *are* helpful in deciding when structure is <u>not</u> being found. It is less clear, as shown in (2.101), that when U approaches M_{hcn}, cluster structure has been found. Empirical studies vary: some show that maximizing $\mathcal{V}_{PC}$ (or minimizing $\mathcal{V}_{PE}$) leads to a good interpretation of the data; others have shown that different indirect indices such as the proportion exponent (Windham, 1981, 1982) and Rouben's indices (1978) are sometimes more effective. This simply confirms what we already know: no matter how good your index is, there's a data set out there waiting to trick it (and you).

$\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ essentially measure the distance U is from being crisp by measuring the fuzziness in the rows of U. Normalizations of both indices that scale their ranges so that it is fixed are discussed in the next subsection. A much more subtle point, the dependency of $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ on secondary parameters of the algorithm producing U (specifically, m in FCM-AO) are considered in Pal and Bezdek (1995).

The separation coefficient of Gunderson (1978) was the first indirect validity index that explicitly used the three components (U, **V**; X), where $U \in M_{fcn}$ and **V** is a vector of c prototypes that are associated with the clusters in U - in the language of Table 2.7, the first *indirect parametric data* index. More recent indices in this category include the functionals of Fukuyama-Sugeno (1989) and Xie-Beni (1991). The *Xie-Beni index* $\mathcal{V}_{XB}$ is defined as

$$\mathcal{V}_{XB}(U,\mathbf{V};X) = \frac{\sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^2 \|\mathbf{x}_k - \mathbf{v}_i\|^2}{n\left(\underbrace{\min}_{i \neq j}\left\{\|\mathbf{v}_i - \mathbf{v}_j\|^2\right\}\right)} = \left(\frac{\left(\frac{\sigma}{n}\right)}{\text{sep}(\mathbf{V})}\right) \quad . \tag{2.102}$$

Xie and Beni interpreted their index by writing it as the ratio of the *total variation* σ of (U, **V**) and *separation* sep(**V**) between the vectors in **V**:

$$\sigma(U,\mathbf{V};X) = \sum_{i=1}^{c}\left(\sum_{k=1}^{n} u_{ik}^2 \|\mathbf{x}_k - \mathbf{v}_i\|^2\right) \quad ; \tag{2.103}$$

$$\text{sep}(\mathbf{V}) = \underbrace{\min}_{i \neq j}\left\{\|\mathbf{v}_i - \mathbf{v}_j\|^2\right\} \quad . \tag{2.104}$$

If (U, **V**) is an extrema of the FCM functional J_2 then $\sigma = J_2$. A good (U, **V**) pair should produce a small value of σ because u_{ik} is expected to be high when $\|\mathbf{x}_k - \mathbf{v}_i\|$ is low, and well separated $\mathbf{v}_i$'s will produce a high value of sep(**V**). So, when $\mathcal{V}_{XB}(U_1, \mathbf{V}_1; X) < \mathcal{V}_{XB}(U_2, \mathbf{V}_2; X)$ for either of these reasons (or both), U_1 is presumably a better partition of X than U_2. Consequently, the *minimum* of $\mathcal{V}_{XB}$ over $\mathcal{P}$ is taken as the most desirable partition of X. This strategy makes sense, because the geometric and statistical flavor of $\mathcal{V}_{XB}$ is very similar to the Davies-Bouldin index: the numerators of both are functions of the Euclidean distances $\{\|\mathbf{x}_k - \mathbf{v}_i\|\}$ and the denominators both depend on measures of separation (distances $\{\|\mathbf{v}_i - \mathbf{v}_j\|\}$) between the cluster centers.

Example 2.15 Table 2.10 shows values for the five indices discussed in this section on terminal FCM-AO partitions of X_{30}. Processing parameters were: m = 2, the Euclidean norm for both similarity and termination, ε = 0.001, and initialization by random selection of c distinct points in the data. Crisp partitions of the data for the direct indices V_D and $V_{DB,22}$ were obtained from terminal FCM-AO estimates by hardening with (2.10). The values for V_D and $V_{DB,22}$ in Table 2.10 are slightly different than those in Table 2.8 because the hardened partitions from FCM for c > 4 were slightly different. As expected, all five indices point to the visually correct partition of the data at the value c = 3.

Table 2.10 Validity for terminal FCM-AO partitions of X_{30}

c	$V_{DB,22}$	V_D	v_{PC}	v_{PE}	v_{XB}
2	0.35	0.96	0.91	0.18	0.70
3	**0.18**	**1.53**	**0.97**	**0.08**	**0.02**
4	0.48	0.52	0.92	0.15	0.05
5	0.65	0.13	0.86	0.25	0.41
6	0.77	0.13	0.83	0.77	0.13
7	0.70	0.10	0.80	0.38	0.53
8	0.65	0.18	0.79	0.41	0.23
9	0.54	0.18	0.79	0.41	0.21
10	0.54	0.18	0.77	0.46	0.21

Our next example replicates the experiments just described in Example 2.15 using the Iris data instead of X_{30}.

Example 2.16 Table 2.11 shows values for the five indices on terminal FCM-AO partitions of Iris obtained with the same protocols as in Example 2.15, including hardening of the fuzzy partitions before validation with V_D and $V_{DB,22}$. Please compare the first two columns of Table 2.11 with the corresponding values in the rows of Table 2.9 for c = 2 to 6 to see that only three of the 10 pairs of corresponding values are the same. This is because the hardened FCM partitions of Iris are somewhat different than the crisp partitions obtained directly from HCM except in these three cases. Four of the five indices in Table 2.11 agree that the best partition occurs for c = 2; only Dunn's index, applied to the hardened partition obtained by FCM-AO, points to c = 3.

Table 2.11 Validity for terminal FCM-AO partitions of Iris

c	$V_{DB,22}$	V_D	v_{PC}	v_{PE}	v_{XB}
2	**0.47**	0.08	**0.89**	**0.20**	**0.04**
3	0.76	**0.10**	0.78	0.39	0.09
4	1.03	0.04	0.68	0.58	0.57
5	1.07	0.05	0.62	0.71	0.30
6	1.07	0.06	0.59	0.80	0.27
7	1.15	0.08	0.55	0.91	0.50
8	1.21	0.08	0.52	1.05	0.38
9	1.37	0.08	0.48	1.11	0.33
10	1.41	0.08	0.45	1.18	0.63

In the four dimensional data space chosen by Anderson there are two geometrically well-defined clusters, so the best partition of Iris from (four of the five) indices' point of view, c = 2, is (perhaps) correct. Since the best solution from the modeler's point of view is (perhaps) c=3, this again illustrates the caveat about models and our expectations for them stated immediately after equation (2.5). And we again see disagreement among validity indices about the best value for c on the same partition of the data.

Several generalizations and relatives of the Xie-Beni index have been studied recently. See, for example, Pal and Bezdek (1995), who define and analyze limiting properties of the index $\mathcal{V}_{XB,m}^{FCM}$, which is the Xie-Beni index with memberships raised to the power m >1 that is explicitly tied to FCM.

Bensaid et al. (1996a) introduced another validity index similar to $\mathcal{V}_{XB}$, $\mathcal{V}_{SC}(U,\mathbf{V};X)=\sum_{i=1}^{c}\left[\sum_{k=1}^{n}u_{ik}^{m}\|\mathbf{x}_k-\mathbf{v}_i\|_A^2 \Big/ \left(\left(\sum_{k=1}^{n}u_{ik}\right)\left(\sum_{j=1}^{c}\|\mathbf{v}_i-\mathbf{v}_j\|_A^2\right)\right)\right]$, and call the ratio inside square brackets the *compactness to separation* ratio of cluster i. They illustrate the use of this index for progressive clustering (adjustments to individual clusters during processing) for different tissue types in magnetic resonance images of the brain.

The last indices covered in this subsection are due to Gath and Geva (1989a), who introduced three very useful indirect parametric data indices. These indices involve one more set of clustering outputs than any of the previous measures that are constructed by algorithms such as GK, FCV, GMD and FMLE which produce point prototypes **V**, (fuzzy or probabilistic) partitions U and covariance matrices $\{C_i\}$. Chronological order would place our discussion of these indices before the Xie-Beni index, but we prefer to discuss them here, just before validation of shell clusters, because these three indices involve one more set of parameters, and because they have played an important role in generalizations for shell validity. Gath and Geva (1989a) defined the *fuzzy hypervolume* of $U \in M_{fcn}$ as

$$\mathcal{V}_{HV}(\mathbf{C})=\sum_{i=1}^{c}\sqrt{\det(C_i)} \quad , \qquad (2.105)$$

where $\mathbf{C}=(C_1,\ldots,C_c)\in\Re^{c(p\times p)}$ is the set of fuzzy covariance matrices given by (2.27) with m=1 (this amounts to using the covariance matrices at (2.21c)). **C** is a function of (X, U, **V**), but only **C** appears on the left side of (2.105). To be consistent with the notation in Table 2.7, we call $\mathcal{V}_{HV}$ an *indirect* index. This index should be small when clusters are compact, so good clusters are identified by minima of V_{HV}. For consistency with the next two indices, users often calculate $1/V_{HV}$ and search for the maximum.

Gath and Geva (1989a) also discussed an indirect parametric data measure of dispersion they called the *average partition density* $\overline{V}_{PD}$ of $U \in M_{fcn}$:

$$\overline{V}_{PD}(U, \mathbf{C}) = \frac{1}{c}\sum_{i=1}^{c}\left(\frac{\sum\limits_{\mathbf{x}_k \in \omega_i} u_{ik}}{\sqrt{\det(C_i)}}\right) \quad , \qquad (2.106)$$

where $\omega_i = \left\{\mathbf{x} \in \Re^p : \|\mathbf{x} - \mathbf{v}_i\|^2_{C_i^{-1}} < 1\right\}, i = 1, \ldots, c$ is the open ball centered at $\mathbf{v}_i$ of radius 1 with respect to the fuzzy Mahalanobis norm $\|\mathbf{x} - \mathbf{v}_i\|^2_{C_i^{-1}}$. This index measures the compactness of points in each cluster that have a strong central tendency - the more points within ω_i, the larger will be $\overline{V}_{PD}$, so this index should be maximized. Lastly, they defined the *partition density* V_{PD} of $U \in M_{fcn}$ as

$$V_{PD}(U, \mathbf{C}) = \frac{\sum\limits_{i=1}^{c}\left(\sum\limits_{\mathbf{x}_k \in \omega_i} u_{ik}\right)}{V_{HV}(\mathbf{C})} \quad . \qquad (2.107)$$

We classify (2.106) and (2.107) as *indirect parametric* indices. V_{PD} should maximize when clusters which are geometrically desirable are submitted to it, achieved either by a large numerator (dense clusters), or a small denominator (compact clusters), or both. This index has a geometric rationale that is quite similar to Dunn's index (and the inverse of the Davies-Bouldin index). Gath and Geva illustrate the use of these measures on clusters in various data sets. For example, V_{HV} and V_{PD} both select c = 3, the physically correct choice, for the Iris data when tested in a situation analogous to the experiments described in Tables 2.9 and 2.11.

Example 2.17 This example is a combined illustration of the Gath and Geva (1989a) clustering algorithm called FMLE and cluster validation with their three indices at (2.105)-(2.107). Recent papers of Geva and his coauthors call the combination of FMLE with the use of these three validity indices the *unsupervised optimal fuzzy clustering* UOFC) algorithm. As pointed out in Section 2.3, this method is essentially GMD-AO. The data we chose for this example is the unbalanced data set called X_{43} shown in Figure 2.3(a) that has 40 points in the left cluster and 3 points in an isolated cluster on the right. Although the sample size is quite small, the data can be viewed as having been drawn from a mixture of c = 2 (roughly) circular bivariate Gaussian distributions. Computing protocols for this example: m = 2 in both the FCM and FMLE clustering stages (don't forget that m = 1 when using (2.27) with FMLE).

The algorithm was initialized at c =1. No clusters are computed for this value, but the GG validity indices do take meaningful values. Subsequently, c was incremented from 2 to 5, and for each value of c, FMLE was executed to termination. Initialization at each new value of c is done as explained in Gath and Geva's 1989a paper, by adding one more cluster prototype to the set found at the previous value. The new prototype is placed very far away from every point in X. Specifically, the distance from all data points to the new center are set to 10 times the sum of the variances of all 43 data points.

Table 2.12 lists the values of $1/V_{HV}$, $\overline{V}_{PD}$ and V_{PD} obtained on the terminal outputs of FMLE for each c from 2 to 5. All three indices are to be maximized. First note that the fuzzy hypervolume points to c = 3, which is clearly wrong. This index is felt by Gath and Geva to be least reliable of the three in this group, but it is needed to compute the other two in any case. The average partition density points to c = 4, which is also wrong, and the partition density points to c = 2. We conclude from this (again) the same thing that we learn from previous examples: using just one index of validity is a very dangerous strategy, for there is great inconsistency in the recommendations that various indices can make.

Table 2.12 Validity measures for FMLE partitions of X_{43}

$c \rightarrow$	2	3	4	5
$1/V_{HV}$	0.00045	**0.00049**	0.00040	0.00044
$\overline{V}_{PD}$	0.0025	0.0015	**0.0036**	0.0027
V_{PD}	**0.0050**	0.0044	0.0047	0.0043

(Right cluster) terminal FMLE memberships for c = 2

data pt.	left: u_{1k}	right: u_{2k}
$\mathbf{x}_{41}$	0.000310	0.999690
$\mathbf{x}_{42}$	0.000221	0.999729
$\mathbf{x}_{43}$	0.000053	0.999947

At c = 2 the terminal FMLE partition of X_{43} has cluster memberships that are crisp up to 3 or 4 places past the decimal for the 40 points in the left cluster. Membership columns in both clusters for the three points in the right cluster are shown in the bottom portion of Table 2.12. As you can see, the FMLE algorithm solves the problem illustrated in Example 2.3 without recourse to the trick of semi-supervision illustrated there. Hardening the terminal FMLE partition of X_{43} found at c = 2 produces the visually correct solution.

The terminal cluster centers for c = 2 were, to two decimal places, $\mathbf{v}_{\text{left}}^{\text{FLME}} = (44.8, 48.8)^T$ and $\mathbf{v}_{\text{right}}^{\text{FLME}} = (91, 49)^T$. The labeled sample means for the two visually apparent clusters in X_{43} are exactly these values!

Remark Processing X_{43} with the standard GMD-AO algorithm gives the same result as long as a solution is requested for c = 2. Thus, the added value of FMLE seems to be the three validity indices. On the other hand, all three of these indices fail to indicate c = 3 on data set X_{30} in Figure 2.2 (possibly because the small number of points in each of the three clusters in X_{30} do not follow the expected shapes of samples from 2D Gaussians very well). This illustrates the point we continue to emphasize: no validity index has proven very reliable across wide variations of data, clustering algorithms, or partitions.

E. Standardizing and normalizing indirect indices

Indirect indices such as $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ have at least four problems. First, they are at best indirectly connected to real substructure in X. Second, justification for using them often relies on heuristic rationales - e.g., U_1 is better than U_2 if U_1 is "crisper" than U_2. We have shown this to be a misleading heuristic. Third, many indirect indices can be shown to be, or have been experimentally observed to be, monotonic in c. And fourth, their range is sometimes itself a function of c. This last property makes their use for cluster validity problematical.

For example, equation (2.100) shows that $1/c \le \mathcal{V}_{PC}(U) \le 1$ for every $U \in M_{fcn}$. Thus, as c increases from 2 to n-1, the range of $\mathcal{V}_{PC}$ increases: $c = 2 \Rightarrow \mathcal{V}_{PC} \in \left[\frac{1}{2}, 1\right]$, $c = n-1 \Rightarrow \mathcal{V}_{PC} \in \left[\frac{1}{n-1}, 1\right]$. $\mathcal{V}_{PE}$ has the same problem. Moreover, the range of $\mathcal{V}_{PE}$ is also a function of logarithmic base a.

Variable ranges make interpretation of values of $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ difficult, since they are not referenced to a fixed scale. The significance, for example, of $\mathcal{V}_{PE}(U) = 0.04$ is not as high when $c \approx n - 1$ as it would be for $c << n$ because of the change in the range of

$\mathcal{V}_{PE}$. Moreover, $\mathcal{V}_{PE} = 0$ at $\mathbf{1}_{1\times n} = [1 \ 1 \ \cdots \ 1]$ at c=1 and at $I_{n\times n}$, for c = n. Hence, minimization of $\mathcal{V}_{PE}$ is confined to $c \in \{2, 3, \ldots, n-1\}$.

Many authors have attempted to address this problem through standardizations and normalizations of various indices. The first normalization of this kind was introduced by Bezdek (1974a), who transformed the partition entropy $\mathcal{V}_{PE}$ by a simple scaling so that the normalized index was valued in the unit interval,

$$\hat{\mathcal{V}}_{PE,B}(U) = \frac{\mathcal{V}_{PE}(U)}{\ln_a c}, \ U \in M_{fcn} \quad . \tag{2.108}$$

The limits for $\mathcal{V}_{PE}$ given at (2.100) immediately yield

$$\hat{\mathcal{V}}_{PE,B}(U) = 0 \Leftrightarrow \mathcal{V}_{PE}(U) = 0 \Leftrightarrow U \in M_{fcn} \text{ is crisp} \quad ; \text{and} \tag{2.109a}$$

$$\hat{\mathcal{V}}_{PE,B}(U) = 1 \Leftrightarrow \mathcal{V}_{PE}(U) = \ln_a c \Leftrightarrow U = \overline{U} \quad . \tag{2.109b}$$

This scaling fixes the range of $\hat{\mathcal{V}}_{PE,B}$ so that $\hat{\mathcal{V}}_{PE,B}(U) \in [0,1]$ is independent of c. This makes the comparison of values of $\hat{\mathcal{V}}_{PE,B}$ at different numbers of clusters more appealing than the direct use of $\mathcal{V}_{PE}$. Roubens (1978) gave a similar normalization of the partition coefficient,

$$\hat{\mathcal{V}}_{PC,R}(U) = \left(\frac{c\mathcal{V}_{PC}(U) - 1}{c-1}\right), \ U \in M_{fcn} \quad . \tag{2.110}$$

Comparing (2.100) with (2.110) establishes that

$$\hat{\mathcal{V}}_{PC,R}(U) = 1 \Leftrightarrow \mathcal{V}_{PC}(U) = 1 \Leftrightarrow U \in M_{fcn} \text{ is crisp} \quad ; \text{and} \tag{2.111a}$$

$$\hat{\mathcal{V}}_{PC,R}(U) = 0 \Leftrightarrow \mathcal{V}_{PC}(U) = \frac{1}{c} \Leftrightarrow U = \overline{U} \quad . \tag{2.111b}$$

Consequently, Roubens' normalization of $\mathcal{V}_{PC}$ scales its range so that $\hat{\mathcal{V}}_{PC,R}(U) \in [0,1]$ for any value of c. Backer (1978) discussed the related index $\hat{\mathcal{V}}_{PC,Ba}(U) = \left(\frac{c}{c-1}\right)(1 - \mathcal{V}_{PC}(U)) = 1 - \hat{\mathcal{V}}_{PC,R}$, but used his index as an objective function upon which a cluster seeking algorithm was based. Apparently unaware of Roubens work, Davé (1996) recently (re)introduced $\hat{\mathcal{V}}_{PC,R}$ with a new name, viz., the *modified partition coefficient* (MPC). In any case, $\hat{\mathcal{V}}_{PC,R}$ is, in the end, just like $\hat{\mathcal{V}}_{PE,B}$. Both of these indices are normalized measures of the fuzziness in U, and as such, really address just one of the four problems mentioned above - the problem of variable range.

Dunn (1977) first suggested that normalizations of indirect validity indices be referenced somehow to data substructure - i.e., that they be held accountable more directly for substructure in the data. Dunn proposed the index

$$\hat{\mathcal{V}}_{PE,D}(U) = \frac{\mathcal{V}_{PE}(U)}{\mathcal{V}_{PE,0}(U)} \approx \left(\frac{n\mathcal{V}_{PE}(U)}{n-c}\right), \quad U \in M_{fcn} \quad . \tag{2.112}$$

This quasi-statistical normalization was given as an approximation to the ratio of $\mathcal{V}_{PE}(U)$ to the null hypothesis value we call $\mathcal{V}_{PE,0}(U)$. Dunn used FCM on a reference set X_0 of n vectors uniformly distributed over a hypercube in $\Re^p$ to estimate $\mathcal{V}_{PE,0}(U)$, which he took to be approximately (n-c)/n. Dunn's idea was that if X contained c compact, well-separated clusters, the value $\mathcal{V}_{PE}(U)$ on any reasonable partition U of X should be low relative to the value $\mathcal{V}_{PE,0}(U)$ on X_0. Roughly speaking, $\hat{\mathcal{V}}_{PE,D}$ takes the form of an inverse likelihood ratio test, so minimizing $\hat{\mathcal{V}}_{PE,D}$ ostensibly corresponds to maximizing the approximate likelihood that X contains cluster substructure.

Dunn used FCM to generate the clusters that lead to his approximation of $\mathcal{V}_{PE,0}(U)$, so (2.112), like $\mathcal{V}_{DB,22}^{ARW}$ of Araki et al. (1993), is implicitly linked to the fuzzy c-means model. Whether the same approximation is useful for fuzzy clusters found by other algorithms is an open question.

Numerical experiments given by Dunn (1977) indicate that $\hat{\mathcal{V}}_{PE,D}$ is a useful modification of $\mathcal{V}_{PE}$. However, substituting $U = \overline{U}$ into (2.112) with the upper bound for $\mathcal{V}_{PE}(U)$ in (2.100b) gives the upper bound $\hat{\mathcal{V}}_{PE,D}(\overline{U}) = \left(\frac{n\mathcal{V}_{PE}(\overline{U})}{n-c}\right) = \left(\frac{n \log_a c}{n-c}\right)$ at the equi-membership partition of X, which is again a function of c. Thus, Dunn's normalization of $\mathcal{V}_{PE}$ does *not* solve the variable range problem, but it does (approximately) reference the indirect index $\mathcal{V}_{PE}$ to a property that seems desirable for characterizing data with clusters: namely, that data with cluster structure be non-uniformly distributed over a hypercube in $\Re^p$.

Bezdek et al. (1980) gave probabilistic standardizations for both $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ based on the well known fact that the linear transformation $Y = \frac{X - \mu_X}{\sigma_X}$ of the random variable X with mean and standard deviation (μ_X, σ_X) is a random variable whose mean and standard deviation are $(\mu_Y, \sigma_Y) = (0, 1)$. The assumption used in their analysis was that the validity indices in question were random variables uniformly distributed over the *degenerate* fuzzy c-partitions M_{fcn0} of X. This is necessary because the derivations are done on one column of U, and it is necessary to have independent (in the probabilistic sense) columns to aggregate the results across an entire partition of X. They derived the mean and standard deviation of $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ under this assumption. Specifically, they prove that for $U \in M_{fcn0}$, the expected value (E) and variance (var) of $\mathcal{V}_{PC}$ and $\mathcal{V}_{PE}$ are

$$E(\mathcal{V}_{PC}(U)) = \left(\frac{1}{c+1}\right) \quad ; \qquad (2.113a)$$

$$\mathrm{var}(\mathcal{V}_{PC}(U)) = \left(\frac{4(c-1)}{n(c+1)^2(c+2)(c+3)}\right) \quad ; \qquad (2.113b)$$

$$E(\mathcal{V}_{PE}(U)) = \sum_{k=2}^{c} \frac{1}{k} \quad ; \qquad (2.114a)$$

$$\mathrm{var}(\mathcal{V}_{PE}(U)) = \frac{1}{n}\left(\left(\sum_{k=2}^{c} \frac{1}{k^2}\right) - \left(\frac{(c-1)}{(c+1)}\right)\left(\frac{\pi^2}{6} - 1\right)\right) \quad . \qquad (2.114b)$$

Results (2.113) can be used with $Y = \frac{X - \mu_X}{\sigma_X}$ to standardize $\mathcal{V}_{PC}$, $\hat{\mathcal{V}}_{PC,R}$ and $\hat{\mathcal{V}}_{PC,Ba}$. And results (2.114) can be used to standardize $\mathcal{V}_{PE}$, $\hat{\mathcal{V}}_{PE,B}$ and $\hat{\mathcal{V}}_{PE,D}$ the same way.

For large enough n, the central limit theorem tells us that a standardized random variable is approximately *normal* (Gaussian) with mean 0 and variance 1, indicated as $n(0, 1)$. Thus, for example, when $\mathbf{u} \in N_{fc}$ is uniformly distributed over the fuzzy label vectors N_{fc}, and because $M_{fcn} \subset M_{fcn0}$, we have the standardizations

$$\mathcal{V}^*_{PC}(U) = \left(\frac{n(c+2)(c+3)}{c-1}\right)^{0.5}\left(\frac{(c+1)\mathcal{V}_{PC}(U)-2}{2}\right) \approx n(0,1). \tag{2.115a}$$

$$\mathcal{V}^*_{PE}(U) = \frac{\mathcal{V}_{PE}(U) - \left(\sum_{k=2}^{c}\frac{1}{k}\right)}{\left(\sum_{k=2}^{c}\left(\frac{1}{nk^2}\right) - \left(\frac{c-1}{c+1}\right)\left(\frac{\pi^2-6}{6n}\right)\right)^{0.5}} \approx n(0,1). \tag{2.115b}$$

Since X is always finite, the actual distribution of standardizations such as these can be far from normal. Nonetheless, they provide another way to link statistical tests to properties of substructure in X by providing a basis for significance tests for cluster validity. Like Dunn's normalization of $\mathcal{V}_{PE}$, these standardizations are attempts to characterize clusters statistically as a departure from uniformity in the feature space. And again, this happens at the expense of a fixed range for the validity measure in question.

F. Indirect measures for non-point prototype models

The validity criteria discussed so far were designed largely on the expectation that X contains volumetric or cloud type clusters. This is mirrored in the use of functions that measure central tendency (means) and dispersion (variances) about the means. All of the direct indices discussed above (Davies-Bouldin, Dunn's index and the generalized Dunn's indices) are designed for cloud type clusters, as are the indirect parametric indices of Gath and Geva and the indirect parametric data index of Xie and Beni.

In order to evaluate partitions that represent shell clusters, different validity measures are needed. To see this, let X be any finite set of points uniformly distributed on the surface of a hypersphere in $\Re^p$ with radius 1; and let 100X be the same data drawn from a hypersphere of radius 100. These data sets will have the same statistic, $\bar{\mathbf{v}}$, as their classical measure of central tendency of the points. The covariance structure of X will also be the same, even though X is more compact than 100X. The surface density of X is much greater than that of 100X. But when these points are regarded as shell clusters, the correct hyperspherical prototypes fit them equally well, so measures of validity should indicate this. The standard measures that assess them for central tendency and dispersion can be very misleading.

Several indirect validity measures that relate to the fitting prototypes have their roots in the work of Gath and Geva (1989a). Their three indices were *not* designed for shell clusters - they measure properties which are optimized by compact clouds. However, these indices paved the way towards similar measures that *are* customized for shell clusters. Man and Gath (1994) defined indices for shell clusters that are related to the measures of Gath and Geva (1989a), and Davé (1996) refined the hypervolume and partition density measures for the cases of spherical and ellipsoidal shells. Krishnapuram et al. (1995b) generalized these definitions to more general shell types; the development of these measures follows.

Let $\boldsymbol{\beta}_i$ be the parameters of (i.e., coefficients of the equation of) a shell prototype S_i which is a hyperquadric surface in $\Re^p$. For any $\mathbf{x}_k \in \Re^p$, define

$$\mathbf{t}_{ik} = \mathbf{x}_k - \mathbf{z}_k^i \quad . \tag{2.116}$$

where $\mathbf{z}_k^i$ is a closest point (measured in Euclidean distance) on the shell S_i to $\mathbf{x}_k$. When S_i is a hypersphere with center $\mathbf{v}_i$ and radius r_i the vector $\mathbf{z}_k^i$ in (2.116) takes the explicit form $\mathbf{z}_k^i = \mathbf{v}_i + r_i \frac{(\mathbf{x}_k - \mathbf{v}_i)}{\|\mathbf{x}_k - \mathbf{v}_i\|}$. For other hyperquadric surfaces $\mathbf{z}_k^i$ can be determined using (2.53). For more general types of shells $\mathbf{z}_k^i$ may be difficult to compute. In these cases we can use a convenient point on the shell or simply use the "approximate closest point" (Krishnapuram et al., 1995b) on the shell. For example, in the case of ellipsoidal shells, we can use the point of intersection of the radial line joining $\mathbf{x}_k$ and $\mathbf{v}_i$ with the ellipsoidal shell (cf. ellipsoidal prototypes, Section 2.3). The fuzzy shell covariance matrix for shell S_i is defined as

$$C_{S_i} = \frac{\sum_{k=1}^{n} u_{ik}^m \mathbf{t}_{ik} \mathbf{t}_{ik}^T}{\sum_{k=1}^{n} u_{ik}^m}, \; 1 \le i \le c \quad . \tag{2.117}$$

Let $\mathbf{C}_S = (C_{S_1}, \ldots, C_{S_c}) \in \Re^{c(p \times p)}$. The *shell hypervolume* of a fuzzy c-partition U of X with parameters $\mathbf{B} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_c)$ is defined as

$$\mathcal{V}_{SHV}(\mathbf{C}_S) = \sum_{i=1}^{c} \sqrt{\det(C_{S_i})} \quad . \tag{2.118}$$

Equation (2.118) is a direct generalization of (2.105) for hyperquadric shells. The extension of (2.106) and (2.107) to the non-point prototype case requires some terminology associated with shell clusters. We illustrate the basic ideas using $\Re^2$ as the feature space with circular prototypes, but many of these ideas can be extended to $\Re^p$ and other types of hyperquadric prototypes.

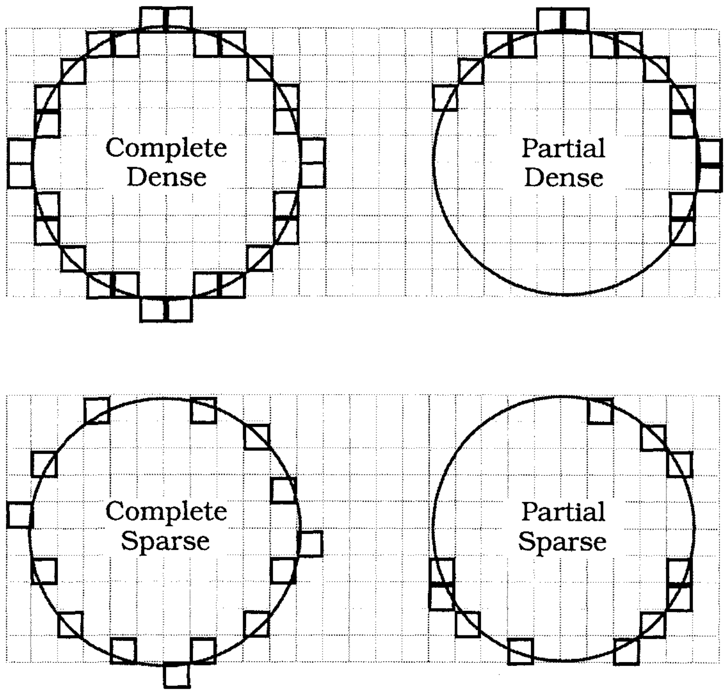


Figure 2.18 "Circular" clusters with different properties

First we point out that clusters are necessarily finite sets of points, while non-point prototypes (shells) that the clusters may be fitted to are continuous structures. So, for example, a circle in $\Re^2$ has infinitely many points, but a "circular cluster" has only finitely many. The following definitions assume a spatial grid of pixels with fixed resolution underlying the points in the clusters. We assume that each pixel can be represented by a square. See Chapter 5 for more on this terminology, which is derived from image processing considerations.

We will say that a "circular" cluster of pixels is *dense* (or not sparse) if and only if its points are either 4-connected or 8-connected in the plane. A circular cluster that is not dense will be called *sparse*. A circular cluster is *complete* if and only if each point on the prototype touches or falls within some (square) pixel belonging to the cluster of pixels. A circular cluster that is not complete will be called a *partial* circular cluster. The *thickness* of a circular cluster is the average distance from its points to the circular prototype. These definitions are illustrated in Figure 2.18. During optimization of functions designed to find good circular prototypes to represent clusters of pixels such as these, the distance used is almost always measured from the "center" of the pixel to the fitting prototype.

Extending (2.106), Krishnapuram et al. define the *average shell partition density* $\overline{V}_{SPD}$ for a fuzzy c-partition U of X with parameters $\mathbf{B} = (\mathbf{V}, \mathbf{C}_S)$ as

$$\overline{V}_{SPD}(U, \mathbf{C}_S) = \frac{1}{c}\sum_{i=1}^{c}\left(\frac{\sum_{\mathbf{x}_k \in \omega_{S_i}} u_{ik}}{\sqrt{\det(C_{S_i})}}\right) \quad , \qquad (2.119)$$

where $\omega_{S_i} = \left\{\mathbf{x} \in \Re^p : \|\mathbf{x} - \mathbf{v}_i\|^2_{C^{-1}_{S_i}} < 1\right\}, i = 1, \ldots, c$. Finally, the *shell partition density* for a c-shell partition U of X with parameters $\mathbf{B} = (\mathbf{V}, \mathbf{C}_S)$ is defined as (cf. (2.107))

$$V_{SPD}(U, \mathbf{C}_S) = \frac{\sum_{i=1}^{c}\left(\sum_{\mathbf{x}_k \in \omega_{S_i}} u_{ik}\right)}{V_{SHV}(\mathbf{C}_S)} \quad . \qquad (2.120)$$

Krishnapuram et al. also proposed an alternative definition for ω_{S_i}, the core or central members in X whose memberships are used in the calculation of the numerators of (2.119) and (2.120),

$$\hat{\omega}_{S_i} = \left\{\mathbf{x}_k \in \Re^p : \|\mathbf{t}_{ik}\| < \tau_{\max,i}; i = 1, \ldots, c\right\} \quad , \qquad (2.121)$$

where $\tau_{\max,i}$ is the expected thickness of the i-th shell. When possibilistic versions of the shell clustering algorithms are used, Krishnapuram et al. suggest $\tau_{\max,i} = \sqrt{w_i}$. Davé (1991b, 1996) proposed an *indirect parametric data* validity index V_{ST} for fuzzy partitions that consist of c hyperspherical shells with parameters **B**,

$$V_{ST}(U, \mathbf{B}; X) = \frac{\sum_{i=1}^{c}\left[\frac{\sum_{k=1}^{n} u_{ik}^m\left(\|\mathbf{x}_k - \mathbf{v}_i\| - r_i\right)^2}{\sum_{k=1}^{n} u_{ik}^m}\right]}{\left(\frac{1}{c}\right)\sum_{i=1}^{c} r_i} \quad . \qquad (2.122)$$

Recall $\|\mathbf{x}_k - \mathbf{v}_i\| - r_i$ as shown in Figure 2.8 when interpreting this equation. Each factor of the numerator of (2.122) is interpreted as the thickness of the i-th hyperspherical shell since it is a generalized average distance from the points in the i-th cluster to the i-th shell. The denominator in (2.122) is called the *average radius of the shells* and is used to normalize this index so that the shell thickness is measured relative to the size of the circles. The index V_{ST} can be extended to ellipsoidal shells, and is to be minimized for identification of the best partition of X.

Krishnapuram et al. (1995b) also define the total fuzzy *average shell thickness* $\overline{V}_{ST}$ for fuzzy partitions that consist of c hyperquadric shells with parameters $\mathbf{B} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_c)$ as

$$\overline{V}_{ST}(U, \mathbf{B}) = \sum_{i=1}^{c} \left[\frac{\sum_{k=1}^{n} u_{ik}^m \|\mathbf{t}_{ik}\|^2}{\sum_{k=1}^{n} u_{ik}^m} \right] \quad . \tag{2.123}$$

The validity measures in equations (2.118), (2.119), (2.120), (2.122) and (2.123) suffer from many drawbacks. Their biggest problem is large variability depending on the size, sparsity and incompleteness of shell clusters. They also lack normalized or standardized (theoretical) values to compare against the validity of a particular c-partition. Hypervolume and shell thickness may be misleadingly small when c is overspecified because there may be only a few points in each shell cluster. For example, if there are only three points in a circular shell cluster, the error of a perfect fit is zero, the volume is zero while the density is infinite, regardless of the relative placement of the three points.

With a view towards ameliorating these drawbacks, Krishnapuram et al. (1995b) introduced a surface density criterion for validation of hyperquadric shell clusters. In the two-dimensional case, the *shell surface density* $V_{SSD_{i2}}$ of the i-th cluster of a fuzzy partition U whose parameters are $\boldsymbol{\beta}_i = (\mathbf{v}_i, C_i)$ is defined as the number of points per unit of estimated surface density (along the fitting prototype), i. e.,

$$V_{SSD_{i2}}(U, C_i) = \frac{\sum_{i=1}^{c} \left(\sum_{\mathbf{x}_k \in \hat{\omega} S_i} u_{ik} \right)}{2\pi \sqrt{\mathrm{Tr}(C_i)}} \quad , \tag{2.124}$$

C_i in (2.124) is the fuzzy covariance matrix in (2.27), not the shell covariance matrix at (2.117), and it is not involved in the iterative calculations of the algorithm; rather, this matrix is computed once after the algorithm terminates. The quantity $\sqrt{Tr(C_i)}$ in the denominator of (2.124) is interpreted as the *effective radius*, $\sqrt{Tr(C_i)} = r_{eff,i}$, of the i-th shell because the "equivalent circle" with radius $\sqrt{Tr(C_i)}$ has the same second moment as the shell cluster under consideration. $2\pi\sqrt{Tr(C_i)}$ is an *estimate* of the arc length of the prototype that represents the (possibly partial) cluster, since the exact arc length cannot be computed easily for clusters that are sparse or partial. In the continuos case for a complete circle of radius r_i it can be shown that $r_i = \sqrt{Tr(C_i)}$.

In the three-dimensional case, the shell surface density $V_{SSD_{i3}}$ of the i-th cluster of a fuzzy partition U whose parameters are $\beta_i = (\mathbf{v}_i, C_i)$ is defined as

$$V_{SSD_{i3}}(U, C_i) = \frac{\sum_{i=1}^{c}\left(\sum_{\mathbf{x}_k \in \hat{\omega}_{S_i}} u_{ik}\right)}{4\pi(Tr(C_i))}, \ 1 \le i \le c \quad , \tag{2.125}$$

where $Tr(C_i) = r^2_{eff,i}$ is the square of the effective radius. In this case $V_{SSD_{i3}}$ measures the number of points per unit of *estimated* surface area of the i-th shell. The *average shell surface density* in the two $(\overline{V}_{SSD_2})$ or three $(\overline{V}_{SSD_3})$ dimensional cases is

$$\overline{V}_{SSD_2}(U, \mathbf{C}) = \frac{1}{c}\left(\sum_{i=1}^{c} V_{SSD_{i2}}(U, C_i)\right) \quad ; \tag{2.126b}$$

$$\overline{V}_{SSD_3}(U, \mathbf{C}) = \frac{1}{c}\left(\sum_{i=1}^{c} V_{SSD_{i3}}(U, C_i)\right) \quad . \tag{2.126b}$$

These measures are used to evaluate fuzzy hyperquadric c-shell partitions U of X characterized by the shell parameters **B**. After the algorithm terminates, if desired, the parameters (**V**, **C**) are computed non-iteratively.

Example 2.18 Figure 2.19 shows a 158-point data set X_{158} consisting of three shell clusters (a circle, an ellipse and a parabola). Initialization and termination criteria were the same as those used in Example 2.8. To decide on the central members of each cluster, (2.121) was used with $\tau_{\max,i} = 2 \, \forall i$.

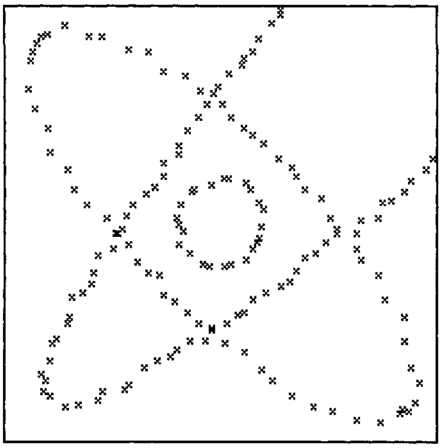


Figure 2.19 Three shell clusters X_{158}

The FCQS algorithm was applied to X_{158} for c=2,...,10, and the validity indices $\overline{V}_{ST}$, V_{SHV}, V_{PD}, and $\overline{V}_{SSD_2}$ were computed using terminal partitions and parameters from FCQS. Table 2.13 summarizes the validity values obtained.

Table 2.13 Validity measures for FCQS partitions of X_{158}

c	$\overline{V}_{ST}$	V_{SHV}	V_{PD}	$\overline{V}_{SSD_2}$
2	296.14	127.35	0.13	0.021
3	4.87	1.69	75.93	**0.138**
4	40.89	16.33	7.10	0.070
5	19.29	7.44	15.25	0.067
6	13.86	5.70	21.34	0.070
7	15.35	4.93	26.28	0.058
8	6.88	1.72	83.85	0.071
9	**4.35**	**1.64**	**89.65**	0.050
10	9.36	2.58	54.56	0.052

$\overline{V}_{SSD_2}$, which is to be maximized, indicates the visually correct value of c=3. The other three measures all indicate that the optimum number of clusters is 9. As mentioned just below equation (2.123), this is because many clusters are able to provide good (low error) fits to various pieces of the shell clusters.

To provide a better understanding of this problem, the left view in Figure 2.20 shows the prototypes obtained with c = 3 superimposed on X_{158}. The right view in Figure 2.20 shows the prototypes obtained with c=9 superimposed on X_{158}. This does not to imply that the validity functionals $\overline{V}_{ST}$, V_{SHV} and V_{PD} are without merit. When used in conjunction with $\overline{V}_{SSD_2}$ they can provide valuable information in boundary description applications (cf. Chapter 6).

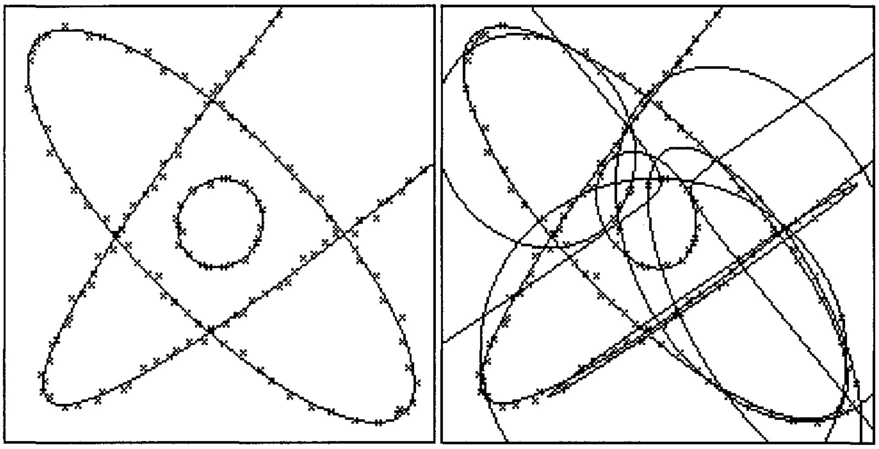


Figure 2.20 FCQS prototypes (left, c=3) and (right, c=9)

We have already mentioned the idea of progressive clustering in connection with the work of Bensaid et al. (1996a). The indirect validity measures for non point prototypes discussed in this section may be used to determine the optimal number of clusters in a c-shells partition of X. However, repetitively clustering the data for an entire range of c-values is very time consuming, and is never guaranteed to work due to local minima of the objective function, particularly for noisy or complex data sets. Progressive clustering based on the validity of individual clusters seems very appropriate for shell type algorithms, and will be discussed in more detail in Chapter 6.

Equation (2.94) provides a very general paradigm for defining cluster validity indices. Appropriate definitions of $\hat{\delta}$ and $\hat{\Delta}$ lead to validity indices suitable for different types (e.g., clouds or shells) of clusters. Pal and Biswas (1997), for example, used minimal spanning trees, relative neighborhood graphs and Gabriel graphs to define the denominator of (2.94). These modifications result in graph theoretic validity indices that are applicable to chain or shell type clusters. These authors also extended the Davies-Bouldin index for chain type clusters using graph theoretic concepts.

G. Fuzzification of statistical indices

Table 2.7 provides one classification of validity measures and procedures. Jain and Dubes (1988) offer another way to subdivide validation methods for crisp partitions of the data. Specifically, they discuss (i) external criteria; (ii) internal criteria; and (iii) relative criteria for validation of: (a) a particular clustering method; (b) nested hierarchies of clusters found by methods such as single linkage (see Chapter 3); (c) individual partitions; and (d) individual clusters. This provides 12 subgroups of methods, all for crisp partitions of X.

External criteria are those that match the structure of a partition U(X) computed with X to a partition U* of X that pertains to the data but which is independent of it. For example, every crisply labeled data set comes with a crisp partition U* of X. Or, an investigator may hypothesize a partitioning U* of X under some assumption (e.g., the random label hypothesis used by Hubert and Arabie (1985)). When a measure is a function of (U*, U(X)), it is called an external criterion. None of the criteria discussed in this section are external.

Internal criteria assess the goodness of fit between an algorithmically obtained crisp partition U(X) and the input data using only the data themselves, usually in the form of the distance matrix $D(X) = [\delta(\mathbf{x}_i, \mathbf{x}_j)]_{n \times n}$ of the data. This group of indices are thus functions of (U(X), D(X) or X), and it intersects (but is not equal to) the measures we call direct data indices in Table 2.7.

Relative indices are used to decide which of two crisp partitions, U(X) or V(X), is a "better choice", where better is defined by the measure that is being used. The Davies-Bouldin index discussed earlier, for example, is a member of this group. This group includes almost all internal indices, which are simply used differently for this different problem, and almost all of the non-crisp indices that have been discussed in this section, most of which apply to fuzzy partitions as well as (hardened) crisp ones derived from them. Most of these crisp validation methods are statistically oriented, and require assumptions about the distribution of the data that are often hard to justify. Nonetheless, several are widely used and have recently been fuzzified, so we provide a short discussion of two popular external indices for crisp partitions.

Let $U, S \in M_{hcn}$ be crisp partitions of X. Define four counts on pairs of objects $(\mathbf{x}_i, \mathbf{x}_j) \in X \times X$

A = # of pairs in the same cluster in U and S ; (2.127a)
B = # of pairs in the same cluster in U but not S ; (2.127b)
C= # of pairs in the same cluster in S but not U ; (2.127c)
D = # of pairs in different clusters in both U and S. (2.127d)

A convenient formulation of the indices we describe next is in terms of the entries of the so-called c×c *matching matrix* M of U and S,

$$M(U,S) = M = [m_{ij}] = US^T \quad . \quad (2.128)$$

Associated with M are three numbers that aggregate the counts in (2.127):

$$T = \left(\sum_{j=1}^{c} \sum_{i=1}^{c} m_{ij}^2 \right) - n \quad ; \quad (2.129a)$$

$$P = \sum_{i=1}^{c} \left(\sum_{j=1}^{c} m_{ij} \right)^2 - n \quad ; \quad (2.129b)$$

$$Q = \sum_{j=1}^{c} \left(\sum_{i=1}^{c} m_{ij} \right)^2 - n \quad ; \quad (2.129c)$$

For $U, S \in M_{hcn}$ Rand (1971) defined the crisp validity measure

$$V_R(U,S) = \left(\frac{A+D}{A+B+C+D} \right) = \left(\frac{2T - P - Q + n(n-1)}{n(n-1)} \right) \quad . \quad (2.130)$$

It is easy to show that

$$V_R = 1 \Rightarrow U = S \quad ; \quad (2.131a)$$

$$V_R = 0 \Rightarrow U \text{ and } S \text{ contain no similar pairs} \quad ; \quad (2.131b)$$

$$0 \le V_R \le 1 \,\forall\, (U,S) \in M_{hcn} \times M_{hcn} \quad . \quad (2.131c)$$

Consequently, high values of V_R are taken as indicants of a good match between U and S. Several corrections of V_R based on normalizations that are in spirit very similar to the ones shown in (2.115) have been proposed to offset its monotone increasing tendency with c. Jain and Dubes (1988) provide a nice discussion of such corrections.

A second external index to compare two crisp partitions of X that is often cited is the index of Fowlkes and Mallow (1983), which for $U, S \in M_{hcn}$ is defined as

$$V_{FM}(U,S) = \left(\frac{T}{\sqrt{PQ}} \right) \quad . \quad (2.132)$$

Just as in (2.131), we have

$$\mathcal{V}_{FM} = 1 \Leftrightarrow U = S \qquad ; \quad (2.133a)$$
$$\mathcal{V}_{FM} = 0 \Leftrightarrow U \text{ and } S \text{ are completely different} \qquad ; \quad (2.133b)$$
$$0 \le \mathcal{V}_{FM} \le 1 \, \forall (U, S) \in M_{hcn} \times M_{hcn} \qquad . \quad (2.133c)$$

High values of $\mathcal{V}_{FM}$ are again interpreted as indicating a good match between U and S. $\mathcal{V}_{FM}$ tends to decrease with increasing c. Milligan and Cooper (1986, e.g.) have studied at least 30 external indices of this type in a series of papers over several years, and they conclude that the adjusted Rand index and the Fowlkes - Mallow measure are probably more reliable than many others of this kind.

Back and Hussain (1995) proposed fuzzy generalizations of $\mathcal{V}_R$ and $\mathcal{V}_{FM}$. Given two *fuzzy* partitions $U, S \in M_{fcn}$, define, in direct analogy with (2.128), the $c \times c$ fuzzy matching matrix between U and S as

$$M_f(U, S) = M_f = [m_{f,ij}] = US^T \qquad . \quad (2.134)$$

Entries in M_f are no longer counts of matches and mismatches between pairs in X×X; rather, $m_{f,ij}$ is now interpreted as the similarity between the fuzzy cluster whose membership values are the i-th row of U and the fuzzy cluster whose membership values are the j-th row of S (which is the j-th column of S^T).

The numbers T, P and Q in (2.129) are well defined for M_f, and can be used to make direct extensions of $\mathcal{V}_R$ and $\mathcal{V}_{FM}$. For $U, S \in M_{fcn}$, Back and Hussain define the fuzzy Rand index and fuzzy Fowlkes-Mallow index as, respectively,

$$\mathcal{V}_{R,f}(U, S) = \left(\frac{2T - P - Q + n(n-1)}{n(n-1)} \right) \qquad ; \quad (2.135)$$

$$\mathcal{V}_{FM,f}(U, S) = \left(\frac{T}{\sqrt{PQ}} \right) \qquad . \quad (2.136)$$

The partition coefficient $\mathcal{V}_{PC}$ at (2.98) is a function of UU^T instead of US^T as in (2.134). Thus, one way to interpret (2.135) or (2.136) is that they are generalizations of $\mathcal{V}_{PC}$, which really compares U to itself, whereas $\mathcal{V}_{R,f}$ and $\mathcal{V}_{FM,f}$ compare U to a second fuzzy partition

S in different ways. Properties of these two indices are interesting. For example,

$$\mathcal{V}_{R,f} = 1 \Rightarrow U = S,\ U, S \in M_{fcn} \qquad ; \ (2.137a)$$

$$\mathcal{V}_{R,f} = 0 \Rightarrow U \text{ and } S \text{ are completely dissimilar} \qquad ; \ (2.137b)$$

$$0 \le \mathcal{V}_{R,f} \le 1 \, \forall (U,S) \in M_{fcn} \times M_{fcn} \qquad . \ (2.137c)$$

$\mathcal{V}_{R,f}$ cannot be used to compare two truly fuzzy partitions because the implication in (2.137a) is one way; in particular, $\mathcal{V}_{R,f}(U,U) \ne 1$ for $U \in M_{fcn} - M_{hcn}$, so its usefulness lies in validation of a fuzzy U against a crisp V. Results similar to (2.137) hold for $\mathcal{V}_{FM,f}$. Since neither $\mathcal{V}_{R,f}$ nor $\mathcal{V}_{FM,f}$ can be used to compare two truly fuzzy partitions of X to each other, Back and Hussain propose a measure that they call the *MC index* for this job. Let $U, S \in M_{fcn}$, and define

$$\mathcal{V}_{MC}(U,S) = 1 - \left(\frac{1}{2n}\right)\left(\sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik} - s_{ik})^2\right) = 1 - \left(\frac{\|U - S\|^2}{2n}\right); \qquad (2.138)$$

From (2.98) $\mathcal{V}_{PC}(U) = \|U\|^2/n$ so $\mathcal{V}_{PC}(U - S) = \|U - S\|^2/n$. Comparing this with (2.138), we have

$$\mathcal{V}_{MC}(U,S) = 1 - \left(\frac{\mathcal{V}_{PC}(U - S)}{2}\right) \qquad . \qquad (2.139)$$

Thus, the MC index is a relative of the partition coefficient that can be used for the comparison of U to S. If $\mathcal{V}_{MC} = 1$, U and S are identical. If $\mathcal{V}_{MC} = 0$, U and S are crisp and share no object in the same class. In this case $\mathcal{V}_{MC}$ becomes the ratio of equally labeled objects in U and S.

To summarize, (2.137a) means, for example, that $\mathcal{V}_{R,f}$ is really useful only for comparing a fuzzy U to a crisp S. When might this happen? Often. For example, Table 2.3 in Example 2.2 shows terminal FCM and PCM partitions of X_{30}. Our comparison of the three matrices in Table 2.2 was confined to hardening the FCM and PCM partition matrices and then noting that all three were identical. If we let S be U*, the visually correct 3-partition of X_{30}, we can compare it to U from FCM using either $\mathcal{V}_{R,f}$ and $\mathcal{V}_{FM,f}$. This extends the idea of external cluster validation for different terminal fuzzy partitions of X to the case where there is a known crisp partition of the data (usually just the labels of labeled data, such as we have for Iris) without using the defuzzification or hardening of U.

2.5 Feature Analysis

Methods that explore and improve raw data are broadly characterized as *feature analysis*. This includes scaling, normalization, filtering, and smoothing. Any transformation $\Phi: \Re^p \mapsto \Re^q$ does *feature extraction* when applied to X. Usually q << p, but there are cases where $q \geq p$. For example, transformations of data can sometimes make them linearly separable in higher dimensions (cf. functional link nets, Zurada, 1992). For a second example where q > p, in image processing each pixel is often associated with a vector of many variables (gray level at the pixel, gradients, texture measures, entropy, average, standard deviation, etc.) built from, for example, intensity values in a rectangular window about the pixel. Examples of feature extraction transformations include Fourier transforms, principal components, and feature vectors built from window intensities in images.

The goals of extraction and selection are: to improve the data for solving a particular problem; to compress feature space to reduce time and space complexity; and to eliminate redundant (dependent) and unimportant (for the problem at hand) features. When p is large, it is often desirable to reduce it to q<<p. *Feature selection* consists of choosing subsets of the original measured features. Features are selected by taking Φ to be a projection onto some coordinate subspace of $\Re^p$. If q << p, time and space complexity of algorithms that use the transformed data can be significantly reduced. Our next example uses a cartoon type illustration to convey the ideas of feature nomination, measurement, selection and the construction of object vector data.

Example 2.19 Three fruits are shown in Figure 2.21; an apple, an orange and a pear. In order to ask and attempt to answer questions about these objects by computational means, we need an object vector representation of each fruit. A human must <u>nominate</u> features that seem capable of representing properties of each object that will be useful in solving some problem. The choices shown in column two are ones that allow us to formulate and answer some (but not all!) questions that might be posed about these fruits.

Once the features (mass, shape, texture) are nominated, sensors measure their values for each object in the sample. The mass of each fruit is readily obtainable, but the shape and texture values require more thought, more time, more work, and probably will be more expensive to collect.

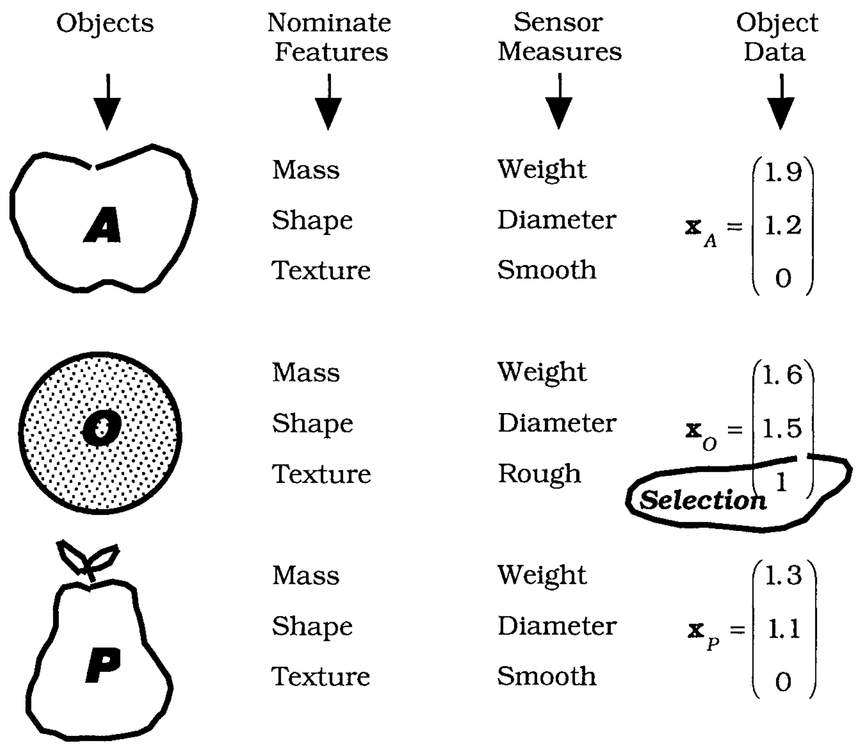

Objects
Nominate Features
Sensor Measures
Object Data
A
Mass
Shape
Texture
Weight
Diameter
Smooth
$\mathbf{x}_A = \begin{pmatrix} 1.9 \\ 1.2 \\ 0 \end{pmatrix}$
O
Mass
Shape
Texture
Weight
Diameter
Rough
$\mathbf{x}_O = \begin{pmatrix} 1.6 \\ 1.5 \\ 1 \end{pmatrix}$
Selection
P
Mass
Shape
Texture
Weight
Diameter
Smooth
$\mathbf{x}_P = \begin{pmatrix} 1.3 \\ 1.1 \\ 0 \end{pmatrix}$

Figure 2.21 Feature analysis and object data

A number of definitions could yield shape measures for the diameter. We might take the diameter as the maximum distance between any pair of points on the boundary of a silhouette of each fruit. This will be an expensive feature to measure, and it may not capture a property of the various classes that is useful for the purpose at hand. Finally, texture can be represented by a binary variable, say 0 = "smooth" and 1 = "rough". It may not be easy or cheap to automate the assignment of a texture value to each fruit, but it can be done. After setting up the measurement system, each fruit passes through it, generating an object vector of measurements. In Figure 2.21 each feature vector is in p=3 space, $\mathbf{x}_i \in \Re^3$.

Suppose the problem is to separate the citrus fruits from the non-citrus fruits, samples being restricted to apples, oranges and pears. Given this constraint, the only feature we need to inspect is the third one (texture). Oranges are the solution to the problem, and they will (almost) always have rough texture, whereas the apples and pears generally will not. Thus, as shown in Figure 2.21, we may select texture, and disregard the first and second features, when solving this problem. This reduces p from 3 to 1, and makes the computational solution simpler and possibly more accurate, since calculations involving all three features use measurements of the other variables that may make the data more mixed in higher dimensions. The feature selection function Φ that formally accomplishes this is the projection $\Phi(x_1, x_2, x_3) = x_3$. It is certainly possible for an aberrant sample to trick the system - that is, we cannot expect a 100% success rate, because real data exhibits noise (in this example noise corresponds to, say, a very rough apple).

Several further points. First, what if the data contained a pineapple? This fruit has a much rougher texture than oranges, but is not a citrus fruit, so in the first place, texture alone is insufficient. Moreover, the texture measurement would have to be modified to, perhaps, a ternary variable; 0 = smooth, 1=rough, and 2 = very rough. Although it is easy to say this verbally, remember that the system under design must convert the texture of each fruit into one of the numbers 0, 1 or 2. This is possible, but may be expensive.

Finally, the features selected depend on the question you are attempting to answer. For example, if the problem was to remove from a conveyor belt all fruits that were too small for a primary market, then texture would be useless. One or both of the first two variables would work better for this new problem. However, the diameter and weight of each fruit are probably correlated. Statistical analysis might yield a functional relationship between these two features. One of the primary uses of feature analysis is to remove redundancy in measured features. In this example, the physical meaning of the variables suggests a solution; in more subtle cases, computational analysis is often the only recourse.

It is often unclear which features will be good at separating the clusters in an object data set. Hence, a large number - perhaps hundreds - of features are often proposed. While some are intuitive, as those in Example 2.19, many useful features have no intuitive or physically plausible meaning. For example, the coefficients of the normalized Fourier descriptors of the outline of digitized shapes in the plane are often quite useful for shape analysis (Bezdek et al., 1981c), but these extracted features have no direct physical interpretation as properties of the objects whose shapes they represent. Even finding the "best" subset of selected features to use is computationally so expensive as to be prohibitive.

Any feature extraction method that produces $Y = \Phi[X] \subset \Re^q$ can be used to make visual displays by taking q = 1, 2, or 3 and plotting Y on a rectangular coordinate system. In this category, for example, are feature extraction functions such as the linear transformations defined by principal components matrices, and feature extraction algorithms such as Sammon's method (Sammon, 1969). A large class of transformations, however, produce *only* visual displays from X (and not data sets $Y \subset \Re$, $\Re^2$ or $\Re^3$) through devices other than scatterplots. In this category are functions such as trigonometric plots (Andrews, 1972) and pictogram algorithms such as Chernoff faces (Chernoff, 1973), and trees and castles (Kleiner and Hartigan, 1981).

The simplest and most natural method for selecting 1, 2 or 3 features from a large feature set is to visually examine each possible feature combination. Even this can be computationally challenging, since p features, for example, offer p(p-1) two dimensional planes upon which to project the data. Moreover, visual assessment of projected subsets can be very deceptive, as we now illustrate.

Example 2.20 The center of Figure 2.22 is a scatterplot of 30 points $X = \{(x_1, x_2)\}$ whose coordinates are listed in columns 2 and 3 of Table 2.14. The data are indexed so that points 1-10, 11-20 and 21-30 correspond to the three visually apparent clusters. Projection of X onto the first and second coordinate axes results in the one-dimensional data sets X_1 and X_2. This illustrates feature selection.

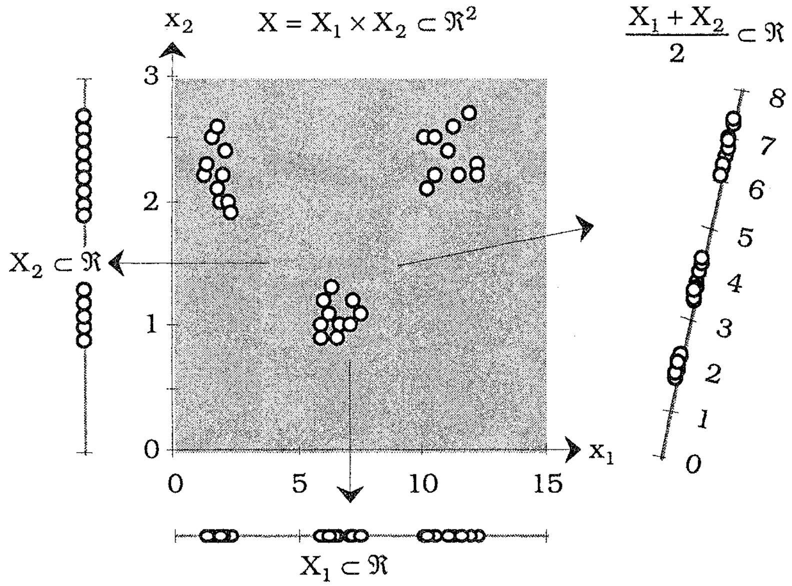


Figure 2.22 Feature selection and extraction

The one dimensional data $\frac{1}{2}(X_1 + X_2)$ in Figure 2.22 (plotted to the right of X, not to scale) is made by averaging the coordinates of each vector in X. Geometrically this amounts to orthogonal projection of X onto the line $x_1 = x_2$. This illustrates feature extraction.

Table 2.14 Data and terminal FCM cluster 1 for four data sets

	x_1	x_2	Init	Init	Init	X	X_1	$\frac{1}{2}(X_1+X_2)$	X_2
			$\mathbf{U}_{(10)}$	$\mathbf{U}_{(20)}$	$\mathbf{U}_{(30)}$	$\mathbf{U}_{(1)}$	$\mathbf{U}_{(1)}$	$\mathbf{U}_{(1)}$	$\mathbf{U}_{(1)}$
$\mathbf{x}_1$	1.5	2.5	1	0	0	0.99	1.00	1.00	0.00
$\mathbf{x}_2$	1.7	2.6	0	1	0	0.99	1.00	0.99	0.03
$\mathbf{x}_3$	1.2	2.2	0	0	1	0.99	0.99	0.98	0.96
$\mathbf{x}_4$	1.8	2.0	1	0	0	1.00	1.00	1.00	0.92
$\mathbf{x}_5$	1.7	2.1	0	1	0	1.00	1.00	1.00	0.99
$\mathbf{x}_6$	1.3	2.3	0	0	1	0.99	0.99	0.99	0.63
$\mathbf{x}_7$	2.1	2.0	1	0	0	0.99	0.99	1.00	0.92
$\mathbf{x}_8$	2.3	1.9	0	1	0	0.97	0.98	1.00	0.82
$\mathbf{x}_9$	2.0	2.4	0	0	1	0.99	1.00	0.98	0.17
$\mathbf{x}_{10}$	1.9	2.2	1	0	0	1.00	1.00	1.00	0.96
$\mathbf{x}_{11}$	6.0	1.2	0	1	0	0.01	0.01	0.01	0.02
$\mathbf{x}_{12}$	6.6	1.0	0	0	1	0.00	0.00	0.00	0.00
$\mathbf{x}_{13}$	5.9	0.9	1	0	0	0.02	0.02	0.07	0.02
$\mathbf{x}_{14}$	6.3	1.3	0	1	0	0.00	0.00	0.00	0.07
$\mathbf{x}_{15}$	5.9	1.0	0	0	1	0.02	0.02	0.05	0.00
$\mathbf{x}_{16}$	7.1	1.0	1	0	0	0.01	0.01	0.02	0.00
$\mathbf{x}_{17}$	6.5	0.9	0	1	0	0.00	0.00	0.00	0.02
$\mathbf{x}_{18}$	6.2	1.1	0	0	1	0.00	0.00	0.01	0.00
$\mathbf{x}_{19}$	7.2	1.2	1	0	0	0.02	0.02	0.03	0.02
$\mathbf{x}_{20}$	7.5	1.1	0	1	0	0.03	0.03	0.04	0.00
$\mathbf{x}_{21}$	10.1	2.5	0	0	1	0.01	0.01	0.01	0.00
$\mathbf{x}_{22}$	11.2	2.6	1	0	0	0.00	0.00	0.00	0.03
$\mathbf{x}_{23}$	10.5	2.5	0	1	0	0.01	0.01	0.00	0.00
$\mathbf{x}_{24}$	12.2	2.3	0	0	1	0.01	0.01	0.01	0.63
$\mathbf{x}_{25}$	10.5	2.2	1	0	0	0.01	0.01	0.01	0.96
$\mathbf{x}_{26}$	11.0	2.4	0	1	0	0.00	0.00	0.00	0.17
$\mathbf{x}_{27}$	12.2	2.2	0	0	1	0.01	0.01	0.01	0.96
$\mathbf{x}_{28}$	10.2	2.1	1	0	0	0.01	0.01	0.02	0.99
$\mathbf{x}_{29}$	11.9	2.7	0	1	0	0.01	0.01	0.01	0.09
$\mathbf{x}_{30}$	11.5	2.2	0	0	1	0.00	0.00	0.00	0.96

Visual inspection should convince you that the three clusters seen in X, X_1 and $\frac{1}{2}(X_1 + X_2)$ will be properly detected by most clustering algorithms. Projection of X onto its second axis, however, mixes the data in the two upper clusters and results in just two clusters in X_2. This illustrates that projections of high dimensional data into lower (often visual) dimensions cannot be relied upon to show much about cluster structure in the original data as explained next.

The results of applying FCM to these four data sets with c = 3, m = 2, ε = 0.01, and the Euclidean norm for both termination and J_m are shown in Table 2.14, which also shows the (poor) initialization used. Only memberships in the first cluster are shown. In Table 2.14 the three clusters are blocked into their visually apparent subsets of 10 points each. As expected, FCM discovers three very distinct fuzzy clusters in X, X_1 and $\frac{1}{2}(X_1 + X_2)$ (not shown in Table 2.14). For X, X_1 and $\frac{1}{2}(X_1 + X_2)$ all memberships for the first ten points are ≥ 0.97, and memberships of the remaining 20 points in this cluster are less than or equal to 0. 07. For X_2, however, this cluster has eight anomalies with respect to the original data.

When the columns of U_{FCM} for X_2 are hardened, this cluster contains the 12 points (underlined in Table 2.14) numbered 3, 4, 5, 6, 7, 8, 10, 24, 25, 27, 28 and 30; the last five of these belong to cluster 3 in X, and the points numbered 1, 2, and 9 should actually belong to this cluster, but do not.

Example 2.20 shows the effects of changing features and then clustering the transformed data. Conversely, clustering can sometimes be used to extract or select features. Bezdek and Castelaz(1977) illustrate how to use terminal point prototypes from FCM to select subsets of triples from a set of 11 (binary-valued) features for 300 stomach disease patients. Their experiments showed that the average error committed by a nearest prototype classifier (cf. Chapter 4) was nearly identical for the original data and the selected feature triples. We discuss this in more detail in Chapter 4, but mention it here simply to illustrate that fuzzy clustering can be used in the context of feature analysis.

Another possibility is to use the c distances from each point in the original data to the cluster centers as (c-dimensional) extracted features that replace the original p-dimensional features. We close chapter 2 with an example that shows how important feature selection can be in the context of a real data application - segmentation of a digital image.

Example 2.21 To show how feature selection can effect a real world pattern recognition problem, consider the segmentation of a 7 channel satellite image taken from (Barni et al., 1996, Krishnapuram and Keller, 1996). Figure 2.23(a) shows channel 1. Barni et al. applied FCM pixel-based clustering with c = 4 to this multispectral image, which had p = 7 bands with spatial dimensions 512×699. Thus, data set X contained n = 512×699 = 357,888 pixel vectors in p = 7-space. (pixel vector $\mathbf{x}_{ij} = (x_{1,ij}, \ldots, x_{7,ij})^T \in X$ is the vector of 7 intensities taken from the spatial location in the image with address (i,j), $1 \le i \le 512$; $1 \le j \le 699$.) In this example we processed the image for two sets of features with FCM using c = 4, m = 2, the Euclidean norm for both termination and J_m, and $\varepsilon = 0.1$. FCM was initialized with the first four pixel vectors from the image as $\mathbf{V}_0$.

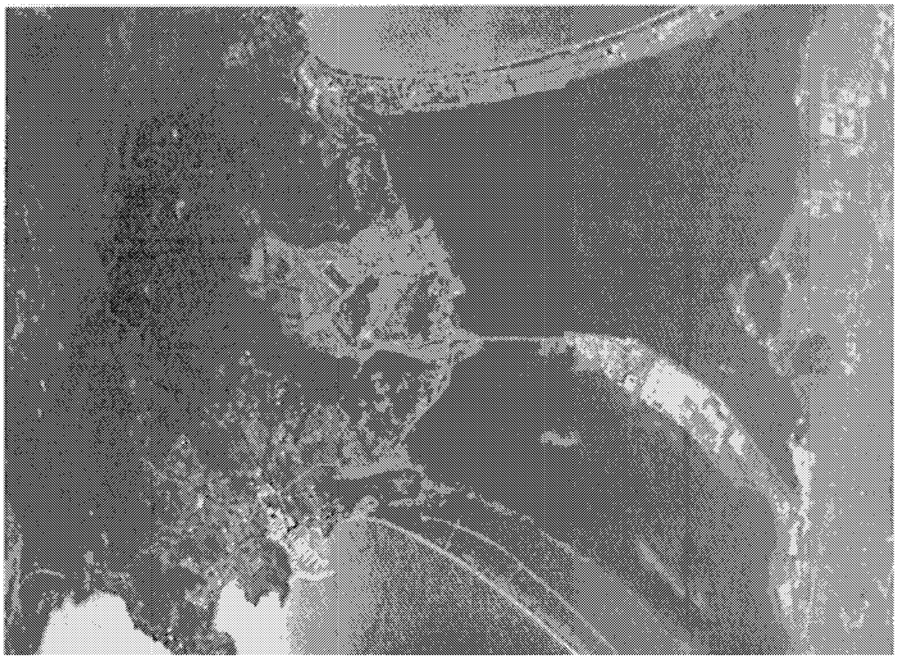


Figure 2.23 (a) Channel 1 of a 7 band satellite image

While this image has 4 main clusters (water, woods, agricultural land, and urban areas), when viewed in proper resolution there are many regions that do not fit into any of the four main categories. For example, what about beaches? Do we distinguish between roads, bridges and buildings or lump them all into the category of urban areas? In the latter case, do the features allow us to do that?

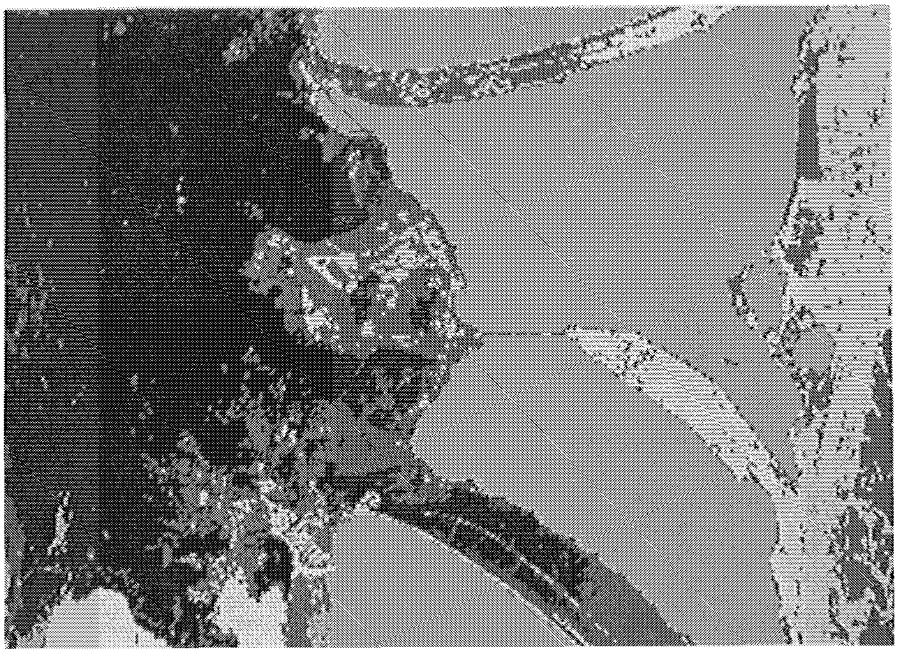


Figure 2.23 (b) FCM segmentation using all 7 features

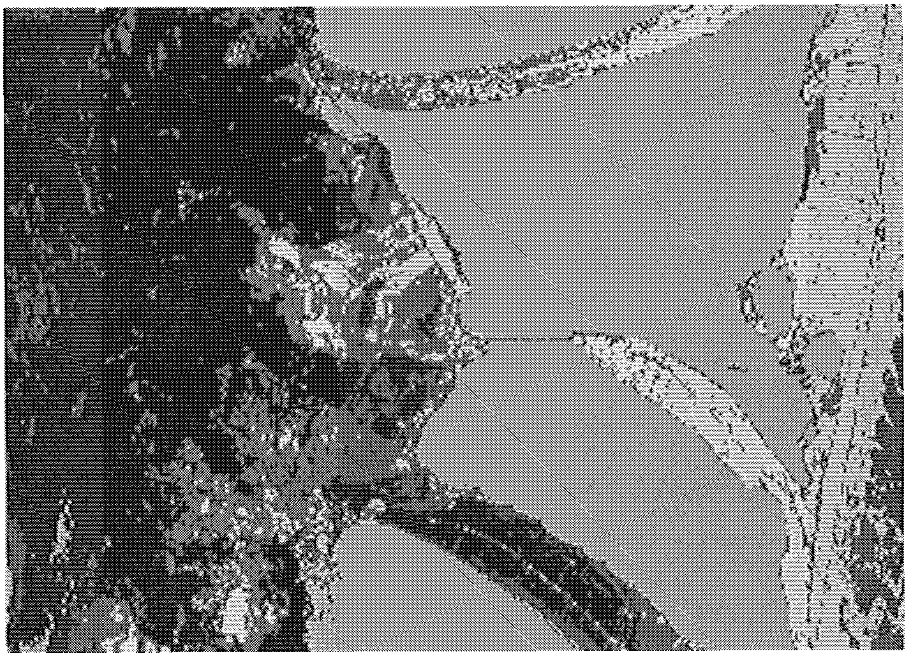


Figure 2.23(c) FCM segmentation using only 2 features

The seven channels in this image are highly correlated. To illustrate this, we show the FCM segmentation when all 7 channels are used (Figure 2.23(b)), and when only channels 5 and 6 are used (Figure 2.23(c)). Visually similar results imply that channels 1- 4 and 7 don't contribute much to the result in Figure 2.23(b). From Figure 2.23(b) it appears (visually) that the FCM misclassification rate is high. This is mainly due to the choice of these features, which are not sufficiently homogeneous within each class, and distinct between classes to provide good discrimination.

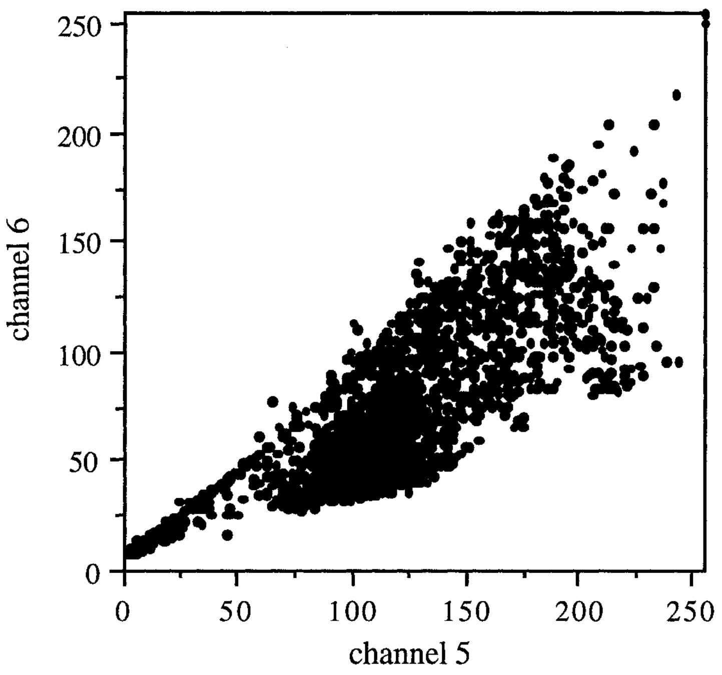


Figure 2.24 Scatter plot of channel 5 vs 6 of satellite image

Figure 2.24 is a scatterplot of the two features (channels 5 and 6) used for the segmentation shown in Figure 2.23(c). Since the number of data points is very large (512x699), to prevent clutter, only a subsample of the data set is shown, and in the subsample only two distinct clusters can be seen. The water region appears as the smaller and denser cluster, because in this region, there is relatively less variation in the intensity values in all 7 channels. The highly reflective areas that appear white in the image show up as outliers in this mapping.

The larger cluster includes samples from all the remaining regions, and it is hard if not impossible to distinguish the remaining three classes within this cluster. If this data were to be used for classifier design (instead of clustering) we could tell from the scatterplot that the features would not be sufficient to distinguish 4 classes. Other more complex features, such as texture, would be needed.

2.6 Comments and bibliography

Clustering algorithms

The crisp or hard c-means clustering model has its roots in the work of Gauss, who wrote to Olbers in 1802 about the method of least squared errors for parameter estimation (Bell, 1966). Duda and Hart (1973) credit Thorndike (1953) as the first explicit user of the HCM functional for clustering. There are now thousands of papers that report theoretical results or applications of the HCM model. There is also a very large body of non-fuzzy work in the signal and image processing literature that is a very close relative of (indeed, perhaps it *is*) HCM. The basic method for this community is the *Lloyd-Buzo-Gray* (LBG) algorithm. See Gersho and Gray (1992) for an excellent summary of this material. A new approach to signal processing based on clustering has been recently discussed by Geva and Pratt (1994).

The FCM model and FCM-AO were introduced by Dunn (1974a) for the special case m=2, and both were generalized by Bezdek (1973, 1974a) for any m > 1. Krishnapuram and Keller's (1993) PCM model and PCM-AO was published in 1993. The newest entrant to the c-means families is a mixed *fuzzy-possibilistic c-means* (FPCM) model and AO algorithm for optimizing it that simultaneously generates both a fuzzy partition of and typicality matrix for unlabeled data set X (Pal et al., 1997a). See Yang (1993) for a nice survey of many other generalizations of the basic FCM model, including some to the case of continous data processes, in which the double sum for J_m is replaced by integrals.

There are several points to be careful about when reading papers on c-means clustering. First, many writers use k instead of c for the integer that denotes the number of clusters. Our notation follows Duda and Hart (1973). Second, many papers and books refer to the *sequential* version of c or k-means more simply as, e.g., "k-means". The well-known sequential version is not an AO method and has many adherents. Its basic structure is that of a competitive learning model, which will be discussed in Chapter 4. Be very careful, when reading about c-means or k-means, to ascertain whether the author means the sequential (Section 4.3.C) or batch version (Section 2.2.A); their properties and performance can be wildly different.

The term ISODATA was used (incorrectly) for both HCM-AO and FCM-AO in early papers that followed the lead of Dunn (1974a) and Bezdek (1973). Conditions (2.6) were used by Ball and Hall (1967) in their crisp ISODATA (*iterative self-organizing data analysis*) algorithm, which is our HCM-AO combined with a number of heuristic procedures for (dynamic) determination of the correct number of clusters to seek. Early papers by Dunn and Bezdek called the FCM-AO algorithm "fuzzy ISODATA", even though there were no heuristics attached to it analogous to those proposed by Ball and Hall. Later papers replaced the term fuzzy ISODATA by fuzzy c-means, but the incorrect use of fuzzy ISODATA still occurs now and then. To our knowledge, a generalization of crisp ISODATA that could correctly bear the name fuzzy ISODATA has - surprisingly- yet to be studied. There is a crisp *iterative self-organizing entropy* (ISOETRP) clustering algorithm due to Wang and Suen (1984) that uses some of the same heuristics as ISODATA. ISOETRP is an interactive clustering model that builds classifier decision trees, and it attempts to optimize an entropy functional instead of J_1: we will discuss this method in Section 4.6.

Suppose you have T sets of unlabeled data, $X = \{X_1, \ldots, X_T\}$, where $X_j = \{x_{j1}, \ldots, x_{jn}\} \subset \Re^p$, $|X_j| = n$, for j = 1 to T. Sato et al. (1997) call data of this kind *3-way object data*, and refer to X_j as the j-th *situation* in the data. Data like these are common. For example, in estimates of brain tumor volume such as those made by Clark et al. (1998), X_j corresponds to the j-th slice in a set of T slices of 3D magnetic resonance images. In this example, the data are not collocated either spatially or temporally. For a second example, X_j might be the j-th image in a temporal sequence of frames collected by an imaging sensor such as a Ladar on a flying seeker platform that sweeps the scene below it. In this second case the data are not temporally collocated, but after suitable adjustments to register the images, they are spatially collocated.

In the first step of tumor volume estimation in Clark et al. (1998) each of the T magnetic resonance slices is independently segmented by unsupervised FCM, leading to a set of T computationally uncorrelated terminal pairs, say $\{(U_1, \mathbf{V}_1), \ldots, (U_T, \mathbf{V}_T)\}$ for the input data sets $X = \{X_1, \ldots, X_T\}$. In such a scheme the number of clusters could be - and in this application should be - a variable, changing from slice to slice as the number of tissue classes changes. In the seeker example, however, when images are collected at very high frame rates, only the locations of the targets (the $\mathbf{V}_k$'s) in the images should change. The number of clusters for each frame in a (short time window) of this temporal data should be fixed. You can cluster each image in such data independently with c fixed, of course, and the sequence $\{(U_1, \mathbf{V}_1), \ldots, (U_T, \mathbf{V}_T)\}$ might be a useful representation of unfolding events in the illuminated scene. However, Sato et al. (1997) discuss a different approach to this problem that involves a very minor change to the FCM functional that seems like a useful alternative.

Sato et al. extend the basic FCM function $J_m(U, \mathbf{V}) = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^m D_{ik}^2$ by adding together T terms (one for each X_j), and each term is weighted with a user specified weight ω_j, j = 1,...,T. Their *temporal fuzzy c-means* (TFCM) function is defined as $J_m^{TFCM}(U, \{\mathbf{V}_j\}) = \sum_{j=1}^{T} \omega_j J_m(U, \mathbf{V}_j)$, $\omega_j > 0$ for all j. (TFCM is our name for their model and algorithm; they don't really give it a name.) J_m^{TFCM} is a positive linear combination of T copies of the FCM functional, so an easy corollary that follows from the proofs for necessary conditions (Bezdek, 1981) yields necessary conditions for AO minimization of J_m^{TFCM} that are simple extensions of (2.6a) and (2.6b). The fuzzy partition common to all T terms of J_m^{TFCM} is calculated as

$$u_{ik} = \left[\sum_{s=1}^{c} \left\{ \frac{\sum_{j=1}^{T} \omega_j \delta^2(\mathbf{x}_{jk}, \mathbf{v}_{ji})}{\sum_{t=1}^{T} \omega_t \delta^2(\mathbf{x}_{tk}, \mathbf{v}_{ts})} \right\}^{\frac{1}{m-1}} \right]^{-1}, \ 1 \le i \le c, \ 1 \le k \le n. \quad (2.140a)$$

The c prototypes $\mathbf{V}_j$, one for each data set X_j, are updated with the common fuzzy c-partition in (2.140a) using the standard equation in (2.7b),

$$\mathbf{v}_{ji} = \left(\sum_{k=1}^{n} u_{ik}^m \mathbf{x}_{jk} \Big/ \sum_{k=1}^{n} u_{ik}^m \right), \ 1 \le j \le T, \ 1 \le i \le c \quad . \quad (2.140b)$$

In (2.140) the values $\{u_{ik}\}$ define a common fuzzy c-partition for all T data sets $X = \{X_1, \ldots, X_T\}$, and for each data set X_j, there is a set of c point prototypes, $\mathbf{V}_j = \{\mathbf{v}_{j1}, \ldots, \mathbf{v}_{jc}\} \subset \Re^{cp}$. Sato et al. only discuss the case where δ is the Euclidean norm, but equations (2.140) are easily generalized to any inner product norm, and, we suspect, are also easily generalizable to many instances of non-point prototype models as well. AO between (2.140a) and (2.140b) produces, at termination, the set $\{(U, \mathbf{V}_1), \ldots, (U, \mathbf{V}_T)\}$. Thus, U is a common fuzzy c-partition of all T situations, and the $\{\mathbf{V}_j\}$ provide an estimate of the trajectory of the c point prototypes through time (that is, through the 3-way data). Because tumors come and go in a set of magnetic resonance image slices of a human with a brain tumor, TFCM seems inappropriate for the application discussed by Clark et al. (1998), but we can imagine the sequence $\{\mathbf{V}_j\}$ being very useful in situations such as the seeker example in automatic target recognition, or hurricane tracking via real time satellite imagery. However, it is clear that the effectiveness of TFCM is very dependent upon "good" choices for the T fixed, user-defined weights $\{\omega_j\}$.

Sato et al. give several examples of TFCM, including a data set of 60 dental patients who have had underbite treatment. Each patient has p=8 numerical features measured at T=3 different post-treatment times. TFCM with c = 4, m = 2 and the Euclidean norm are applied to this data, and the resultant prototypes of the four clusters seem to have physically interpretable meanings over time. The only complaints we have about the examples in Sato et al.'s book are that none of them are compared to other methods (such as applying FCM to each data set in the sequence independently); and no guidance is given about the choice of the T weights $\{\omega_j\}$. Nonetheless, we think this is a very promising extension of FCM for some problems with a temporal aspect.

Much of the general theory for AO (also called grouped coordinate descent) is contained in Bezdek et al. (1986a, 1987a), Redner et al. (1987) and Hathaway and Bezdek (1991). AO schemes are essentially split gradient descent methods, and as such, suffer from the usual numerical analytic problems. They need good initializations, can get trapped at undesirable local extrema (e.g., saddle points), and can even exhibit limit cycle behavior for a given data set. Karayiannis (1996) gives fuzzy and possibilistic clustering algorithms based on a generalization of the reformulation theorem discussed in Section 2.2.E.

There are many hybrid clustering models that combine crisp, fuzzy, probabilistic and possibilistic notions. Simpson (1993) uses fuzzy sets to find crisp clusters (directly, without hardening). Moreover, this method adjusts the number of clusters dynamically, so does not rely on posterior validation indices. The method of Yager and Filev (1994a) called the "mountain clustering method" is often described as a fuzzy clustering method. However, this method, and a relative called the subtractive clustering method (Chiu, 1994) are not fuzzy clustering methods, nor are they even clustering methods. They both seek point prototypes in unlabeled data without reference to good partitions of the data, and then use the discovered prototypes non-iteratively to construct crisp clusters with the nearest prototype rule (equation 2.6a). These models will be discussed in Chapter 4.

Runkler and Bezdek (1998a) have recently introduced a new class of clustering schemes that are not driven by objective function models. Instead, they propose *alternating cluster estimation* (ACE), a scheme whereby the user selects update equations for prototypes and memberships from toolbars of choices for each of these sets of variables. All of the AO models of this chapter can be imbedded in the ACE framework (including probabilistic models), and additionally, ACE enables users to build new clustering algorithms by a "mix and match" paradigm, that is, by mixing formulae from various sources. This type of algorithm trades mathematical interpretability (the objective function and necessary conditions for it) for user-defined properties of presumably desirable prototypes and membership functions (e.g., convexity of membership functions, a property not enjoyed by continuous functions satisfying the FCM necessary condition (2.7a)).

Cluster Validity

A third way (besides direct and indirect validity measures) to assess cluster validity is to assign each $U \in \wp$ some task, and then compare its performance on the task to other candidates in $\wp$ (Backer and Jain, 1981). For example, the labels in U can be used to design a classifier, and empirical error rates on labeled data can then be used to compare the candidates. This is *performance-based validity*. It is hard to give more than a general description of this idea because the performance criteria which dictate how to select the best solution are entirely context dependent. Nonetheless, for users with a real application in mind, this is an important methodology to remember. A best strategy when the end goal is known may be to first eliminate badly incorrect clustering outputs with whatever validity measures seem to work, and then use the performance goal to make a final selection from the pruned set of candidates.

Our discussion of cluster validity was made in the context that the choice of c is the most important problem in validation studies. Duda and Hart (1973) call this the "fundamental problem of cluster validity". A more complete treatment of cluster validity would also include validation of clustering methods as well as validation of individual clusters, neither of which was addressed in Section 2.4.

Applying direct, indirect or performance-based validity criteria to each partition in $\wp$ is called *static* cluster validity. When assessment criteria are integrated into the clustering scheme that alter the number of clusters during computation (that is, other than in the obvious way of clustering once at each c in some prespecified range), as in Ball and Hall's (1967) ISODATA or Tou's (1979) DYNOC, the resulting approach is called *dynamic* cluster validation. In this approach $\wp$ is not generated at all - rather, an algorithm generates U, assesses it, and then adjusts (or simply tries other) parameters (and possibly algorithms) in an attempt to find a "most valid" U or P for X. Surprisingly enough, a fuzzy version of ISODATA per se has never been developed. However, many authors have added merge-split (or progressive) clustering schemes based on values of various validity functionals to FCM/PCM in an attempt to make them dynamic (see

Davé and Bhaswan, 1991b, Krishnapuram and Freg, 1992, Bensaid et al., 1996b, Frigui and Krishnapuram, 1997).

Given the problems of indirect indices (functions of U alone, which are usually mediocre at best), it is somewhat surprising to see so much new work on functionals of this type. For example, Runkler (1995) discusses the use of a family of indirect indices (the mean, median, maximum, minimum and second maximum) of the c row maximums $\{M_i = \underbrace{\max}_{1 \le k \le n}\{u_{ik}\}, i = 1, \ldots, c\}$ of U for validation of clusters found by the FCE algorithm. Continued interest in measures of this type can probably be attributed to three things: their simplicity; the general allure of computing "how fuzzy" a non-crisp entity is; and most importantly, how important cluster validity really is for users of clustering algorithms. Trauwaert (1988) contains a nice discussion of some of the issues raised here about the use of the partition coefficient (historical note: Trauwaert mistakenly attributed the partition coefficient to Dunn in the title of and throughout his paper; Bezdek (1973) introduced the partition coefficient to the literature). See Cheng et al. (1998) for a recent application of the partition entropy at (2.99) to the problem of (automatically) selecting thresholds in images that separate objects from their backgrounds.

There are several principles that can be used as guides when building an index of validity. First, computational examples on many data sets with various indices suggest that the more reliable indices explicitly use *all of the data* in the computation of the index. And second, most of the better indices also use the cluster means $\overline{\mathbf{V}}(U)$ if U is crisp or whatever prototypes **B** in (2.24a) are available in their definition. Even when X is not used, using $\overline{\mathbf{V}}(U)$ or **B** implicitly involves all of X, and insulates indices from being brittle to a few noisy points in the data.

If it is possible to know, or to ascertain, the rough structure of the data, then of course an index that is designed to recognize that type of structure is most appealing. For example, mixtures of normal distributions with roughly equal covariance structure are expected to generate hyperellipsoidal clusters that are most dense near their means, and in this case any index that optimizes for this type of geometry should be more useful than those that do not. Bezdek et al. (1997b) discuss this idea at length, and show that both crisp and fuzzy validity indices as reliable as many of the most popular probabilistic criteria for validation in the context of normal mixtures.

When an indirect index is used (partition coefficient, partition entropy, etc.), the quality of **B** either as a compact representation of the clusters or as an estimate of parameters is never directly measured, so this class of indices cannot be expected to perform well unless the data have very distinct clusters. Thus, indirect indices that involve only U are probably not very useful for volumetric or shell cluster validation - in either case they simply measure the extent to which U is a non-crisp partition of the data. When parameters such as **B** are added to an indirect index (Gath-Geva or Xie-Beni for example), the issue of cluster type becomes more important. When the clusters are volumetric (because they are, or because the algorithm that produced them seeks volumetric structure), **B** should be a set of point prototypes. When the clusters use **B**, a parameter vector of a set of non-point prototypes as representatives, the cluster structure is shell like. In either case, the validity index should incorporate **B** into its definition. We feel that the best indices are direct or indirect parametric data indices. This is why we chose the classification of indices in Table 2.7 as the fundamentally important way to distinguish between types of measures.

The literature of fuzzy models for feature analysis when the data are unlabeled as in this chapter is extremely sparse and widely scattered. The few papers we know of that use fuzzy models for feature analysis with *labeled data* will be discussed in Section 4.11.

Finally, we add some comments about clustering for practitioners. Clustering is a very useful tool that has many well documented and important applications: to name a few, data mining, image segmentation and extraction of rules for fuzzy systems. The problem of validation for truly unlabeled data is an important consideration in all of these applications, each of which has developed its own set of partially successful validation schemes. Our experience is that no one index is likely to provide consistent results across different clustering algorithms and data structures. One popular approach to overcoming this dilemma is to use many validation indices, and conduct some sort of vote among them about the best value for c. Many votes for the same value tend to increase your confidence, but even this does not prevent mistakes (Pal and Bezdek, 1995). We feel that the best strategy is to use several very different clustering models, vary the parameters of each, and collect many votes from various indices. If the results across various trials are consistent, the user may assume that meaningful structure in the data is being found. But if the results are inconsistent, more simulations are needed before much confidence can be placed in algorithmically suggested substructure.

3 Cluster Analysis for Relational Data

3.1 Relational Data

In Chapter 1 we mentioned that two types of data, object (X) and relational (R), are used for numerical pattern recognition. Relational methods for classifier design are not as well developed as methods for object data. The most compelling reason for this is probably that sensors collect object data. Moreover, when each object is not represented by a feature vector, the problem of feature analysis is non-existent. Consequently, the models in this chapter deal exclusively with clustering. There are many applications that depend on clustering relational data - e.g., information retrieval, data mining in relational databases, and numerical taxonomy, so methods in this category are important. Several network methods for relational pattern recognition are given in Chapter 5.

The basic idea in relational clustering is to group together objects in an object set O that are "closely related" to each other, and "not so closely" related to objects in other clusters, as indicated by relative relational strengths. The objects are usually implicit, so we find groups in O by clustering based on the strength of relationships between pairs of objects. If we have object data $X \subset \Re^p$, we can generate many relations R(X) from X.

Relational clustering algorithms are usually *hierarchical* (local, graph theoretic) or *partitional* (global, objective function driven). Many hierarchical algorithms are designed to find clusters in any proximity relation and hence, these methods will also work for fuzzy relational data. Consequently, this chapter describes several non-fuzzy methods that produce crisp partitions; and several fuzzy methods that can produce crisp or fuzzy partitions from proximity relations.

Hierarchical clustering can be divided into *agglomerative* (bottom up, clumping) and *divisive* methods. Agglomerative algorithms start with each object in its own singleton cluster (c=n), and subsequently merge similar clusters until the procedure terminates at a single cluster (c=1). In the top down or divisive approach all points begin in a single cluster (c=1) and then clusters are split by some rule until every object eventually ends up in its own singleton cluster (c=n). Good expositions of many of these methods appear in Sneath and Sokal (1973) and Jain & Dubes (1988). We will briefly describe one family of agglomerative algorithms - the linkage algorithms - because of their connection to some fuzzy relational methods that produce hierarchical clusters. The chapter concludes with discussions about several partitional algorithms that are driven by minimizing relational objective functions.

A. Crisp Relations

Relations on finite data sets need not be square nor binary, but in pattern recognition they almost always are square and binary, so our presentation is confined to this special case. Let the set of objects be $O = \{o_1, \ldots, o_n\}$. A crisp binary relation $\mathcal{R}$ in O is a crisp subset $\mathcal{R} \subset O \times O$. Pairs of objects, say (o_i, o_j) are either fully related under $\mathcal{R}$, or they are not related at all. Since $\mathcal{R}$ is a crisp subset, we can describe it by a membership function, say $\rho: O \times O \mapsto \{0,1\}$. The n^2 numbers $\{\rho(o_i, o_j)\}$ characterize the membership of (o_i, o_j) in the relation $\mathcal{R}$, and we write $\rho(o_i, o_j) = 1 \Leftrightarrow o_i \mathcal{R} o_j \Leftrightarrow o_i$ is $\mathcal{R}$-related to o_j.

It is convenient to array the relationships as an $n \times n$ *relation matrix* $R(\rho; O) = \left[r_{ij} \doteq \rho(o_i, o_j)\right]_{n \times n}$. We may write $R(\rho; O)$ simply as R, and we follow others in sloppily calling the matrix R variously "the relation (even though the subset $\mathcal{R}$ actually is, by writing, e.g., aRb instead of a$\mathcal{R}$b)", "the relation matrix'" and even "the relational data". This terminology accrues from crisp relations where the three descriptions of R are equivalent. Since crisp relations are crisp subsets, the notation $\mathcal{R}_1 \subseteq \mathcal{R}_2$ is well defined for two relations $\mathcal{R}_1, \mathcal{R}_2 \subseteq O \times O$. We extend this notation to the relation matrices R_1 and R_2 of $\mathcal{R}_1$ and $\mathcal{R}_2$ by writing $R_1 \le R_2$, meaning $r_{1,ij} \le r_{2,ij}$ for $1 \le i,j \le n$.

More generally, real binary relations in O are functions $\rho: O \times O \to \Re$ called *proximity* relations that represent similarity or *dissimilarity* between pairs of objects. In this case R is a proximity matrix. When $\rho(o_i, o_j) \notin \{0,1\}$, it is customary to regard r_{ij} as the *strength* of the relationship between o_i and o_j. Given an object data set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \Re^p$, $\mathbf{x}_i \in \Re^p$ characterizing object o_i, there are many functions that can be used to convert X into a proximity relation. For example, every metric δ on $\Re^p \times \Re^p$ produces a proximity relation in $X \times X$. We discuss this in detail in Section 3.2.

The conceptual basis of relational clustering is $\rho: O \times O \to \Re$. We identify three basic types of relations:

$\rho: O \times O \to \Re$ *Real* Binary Relation (on or in) O (3.1a)

$\rho: O \times O \to [0,1]$ *Fuzzy* Binary Relation (on or in) O (3.1b)

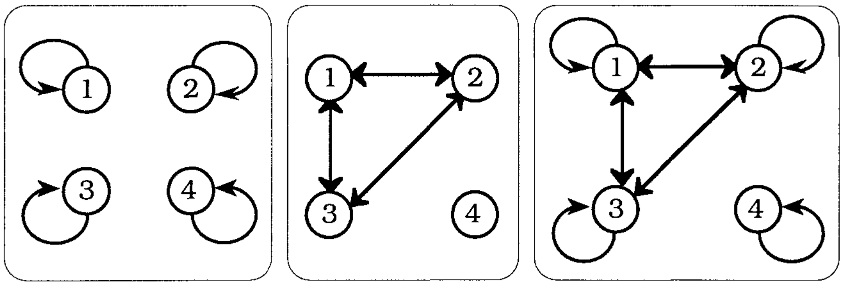
$\rho : O \times O \to \{0,1\}$ *Hard* Binary Relation (on or in) O (3.1c)

These relations are binary because ρ has two arguments. Equation (3.1c) displays the membership function of a crisp subset $\mathcal{R} \subset O \times O$. Similarly, equation (3.1b) shows that we can regard fuzzy relations in O as fuzzy subsets of $O \times O$ characterized by the membership function r. An arbitrary finite proximity relation R can always be converted into a fuzzy relation by a suitable normalization.

A (square binary) relation R is reflexive if $r_{ii} = \rho(o_i, o_i) = 1 \ \forall \, o_i \in O$. $(I_n \le R)$. Reflexivity means that every element is fully related to itself. R is symmetric when $r_{jk} = r_{kj} \ \forall \, j,k \ (R = R^T)$. This means that whenever o_j is related to o_k at any level, o_k is related to o_j at the same level. R is transitive if $r_{jk} = 1$ whenever, for some i, $r_{ji} = 1$ and $r_{ik} = 1$ $(R^2_{\vee\wedge} = R(\vee\wedge)R = R$, where $R^2_{\vee\wedge}$ is the Boolean matrix product of R with itself, $\left[R^2_{\vee\wedge}\right]_{ij} = \overset{n}{\underset{k=1}{\vee}} (r_{ik} \wedge r_{kj})$).

A finite crisp binary relation on $O \times O$ can be viewed as a graph G = (V, E) where V={o_j} are the vertices of G; and E is the set of edges in G, $(o_i, o_j) \in E \Leftrightarrow r_{ij} = 1$. In this context R is the adjacency matrix of G. A path from o_i to o_j in G is any set of nodes that have edges connecting o_i and o_j. The path length is the number of edges in the path. R is transitive if, whenever there is a path of length greater than 1 from o_i to o_j, there is a direct path of length 1 from o_i to o_j.

Example 3.1 Let $O = \{a,b,c,d\}$ and $R = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ be a relation on $O \times O$. This R is reflexive, symmetric and transitive. Reflexivity is represented by self loops at each node as in Figure 3.1(a) ; symmetry is expressed by pairs of edges (shown in Figure 3.1(b) as edges directed in both directions) between pairs of related nodes. Figure 3.1(c) gives the complete graphical representation of the reflexive, symmetric, and transitive relation R.



(a) Reflexivity **(b) Symmetry** **(c) The graph of R**

Figure 3.1 Reflexivity, symmetry and transitivity of R

It is easy to see that R is transitive: the only paths of lengths > 1 in R are paths of length 2 between any pair of the nodes 1,2 and 3; and for each such path, there is a direct path of length 1. So transitivity adds no edges to the graph of this relation that are not required for reflexivity and symmetry.

Definition 3.1 A crisp square binary relation R in $O \times O$ is an *equivalence relation* (ER) if it is (i) reflexive, (ii) symmetric and (iii) transitive. The set of all ERs on n objects will be called $\mathbf{R}_n$.

The ER is important in pattern recognition because it defines a set of c equivalence classes which are isomorphic to a crisp c- partition of O. To see this, let $C_{o_i} = \left\{o_j : o_i R o_j, o_j \in O\right\}$ be the set of objects that are equivalent to o_i. Then for two objects o_i and o_j, since the relation is transitive, either $C_{o_i} = C_{o_j}$ or $C_{o_i} \cap C_{o_j} = \varnothing$. Moreover, $\bigcup_{o_i \in O} C_{o_i} = O$. In Example 3.1 R is a crisp ER, and it induces the unique 2-partition $\{a, b, c\} \cup \{d\}$ on the objects $O = \{a, b, c, d\}$.

An important concept for any crisp relation R is its *closure* with respect to a given property P that R might possess. Generally, the P-closure of R is the *smallest* relation containing R that has property P. The *symmetric closure* of R = {(a, b), (a, c), (c, a), (b, c), (c, b)} on {a, b, c} is formed by adding the single pair (b, a) to R. Other pairs such as (a, a) can be added too, but the *smallest* relation that is symmetric and contains R as a subset is {(a, b), (b, a), (a, c), (c, a), (b, c), (c, b)}. R is not reflexive either. The *reflexive closure* of R is formed by adding the three pairs (a, a), (b, b) and (c, c) to the original relation R without adding (b, a). The smallest relation that contains R that is both reflexive and symmetric is the union of its reflexive and symmetric closures, obtained by adding the four pairs just displayed to R.

Similarly, the *transitive closure* R^{∞} of a crisp relation R is the smallest transitive relation that contains R as a subset. The construction of R^{∞} from a given R does not require that R be reflexive or symmetric. If R is not transitive, we add just enough pairs to the relation to give it this property. In the example of the preceding paragraph, making R symmetric also makes it transitive (coincidentally), and hence, the union of the reflexive and symmetric closures of this R is an ER on {a, b, c}.

If R is not an ER, we can take its closure with respect to the three properties required by Definition 3.1. This gives us an ER $\hat{R}$ on the objects which is the smallest extension of R that is an ER. Clusters in the given objects are obtained from the equivalence classes of $\hat{R}$.

Example 3.2 Let $O = \{a, b, c, d, e\}$ and $R = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ be a relation on $O \times O$. R is not reflexive, symmetric or transitive. Adding 1's at addresses (1,1) and (3,3) yields the reflexive closure of R. Adding 1's at addresses (2,1), (2,3), (3,1) and (5,4) yields the symmetric closure of R. Taking the union of these two closures yields the relation $\hat{R} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$. Every path between pairs of nodes in $\hat{R}$ can be realized by a direct path, so $\hat{R} = R^{\infty}$ is an ER on O, and the unique partition it corresponds to is the 2-partition $\{a, b, c\} \cup \{d, f\}$ of O. This partition of O is based on $\hat{R}$, not on the given data R. Since $\hat{R}$ is a transformation of the given data, it is not correct to assert that $\{a, b, c\} \cup \{d, f\}$ is a partition of O obtained from R. It is proper to regard this partition as the partition of O supported by the ER closest to R.

Comparing matrices of the ERs in Examples 3.1 and 3.2, notice that they both have c sub-blocks of 1's that are $n_i \times n_i$ in size, where n_i is the number of points in the equivalence class (and therefore, the number of points in the i-th crisp cluster in O). This is always the case, up to a permutation of the objects (and hence the columns) of the matrix representing the ER.

R^∞ can be constructed in various ways. Conceptually, the easiest method is the well known result that combines the n (Boolean) powers of R:

$$R^\infty_{\vee\wedge} = R \vee R^2_{\vee\wedge} \vee \ldots \vee R^n_{\vee\wedge} \quad \text{, where} \qquad (3.2)$$

the k-th power, for $k \geq 2$, is defined as $R^k_{\vee\wedge} = \underbrace{R(\vee\wedge)R(\vee\wedge)\cdots(\vee\wedge)R}_{\text{k times}}$.
When R is symmetric and transitive (3.2) collapses to

$$R^\infty_{\vee\wedge} = R^{n-1}_{\vee\wedge} \ (\Leftrightarrow I_n \leq R \text{ and } R = R^T) \quad . \qquad (3.3)$$

These equations are convenient for small n, but direct calculation of R^∞ by matrix multiplication has time complexity $O(n^4)$, so this method is impractical for large n (Cormen et al., 1990). Warshall's algorithm for the transitive closure of a crisp relation is $O(n^3)$, and there are minimal spanning tree approaches (Dunn, 1974b) that are $O(n^2)$. Nonetheless, (3.2) is useful for small to moderately sized data sets and also for pedagogical purposes, so we illustrate its use in Example 3.3.

Example 3.3 Find the transitive closure of $R = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$. First we compute the max-min powers of R:

$$R^2_{\vee\wedge} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix};$$

$$R^3_{\vee\wedge} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix};$$

$$R^4_{\vee\wedge} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

It is easy to check that all higher even powers will equal $R^2_{\vee\wedge}$, and that all higher odd powers will equal $R^3_{\vee\wedge}$. Now use (3.2): take the element by element maximum of all three relations:

$$R^{\infty}_{\vee\wedge} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Notice that $R^{\infty}_{\vee\wedge}$ is neither reflexive or symmetric, so it is not an ER and does not induce a crisp partition on its set of objects.

B. Fuzzy Relations

Symmetry and reflexivity extend uniquely and naturally to fuzzy relations. Extending transitivity, however, is a much more subtle task. A fuzzy relation R can be regarded as a weighted graph G = (V, E, R), with R the (weighted) adjacency matrix of G. $r_{ij} = o_i R o_j$ is the weight of edge $(o_i, o_j) \in E$. This view is advantageous for interpreting transitivity in fuzzy relations.

For a crisp relation $\left[R^2_{\vee\wedge}\right]_{ij} = \overset{n}{\underset{k=1}{\vee}} (r_{ik} \wedge r_{kj})$. The min and max operators correspond to intersection and union in crisp logic. Fuzzy transitivity is defined in terms of two more general operators that are used for the intersection and union of pairs of fuzzy sets. Specifically, intersections are represented by *T-norms* and unions with *S-norms* (T co-norms) of fuzzy sets (Klir and Yuan, 1995).

We use $\oplus$, $\otimes$ respectively as the S and T-norms of any pair of real numbers a, b in [0, 1], $S(a,b) = a \oplus b$, $T(a,b) = a \otimes b$. (There are seven infinite families of T and S norms. See Volume 1 of this handbook for an extensive treatment.) In the fuzzy literature the min and max operators are called T_3 and S_3, $T_3(a,b) = a \wedge b$, $S_3(a,b) = a \vee b$.

The ij-th element of the n×n relation matrix in the $\oplus - \otimes$ composition of two square fuzzy relations R_1 and R_2 is $\left[R_1(\oplus\otimes)R_2\right]_{ij} = \overset{n}{\underset{k=1}{\oplus}} (r_{1,ik} \otimes r_{2,kj})$. If $R_1 = R_2 = R$, the k-th power of R, for k ≥ 2, is $R^k_{\oplus\otimes} = \underbrace{R(\oplus\otimes)R(\oplus\otimes)\cdots(\oplus\otimes)R}_{k \text{ times}}$.

Definition 3.2 Zadeh (1971) A square fuzzy relation R is $\vee - \otimes$ transitive if and only if $r_{ij} \geq \overset{n}{\underset{k=1}{\vee}} (r_{ik} \otimes r_{kj}) \; \forall \; i \neq j$ (i.e., $R \geq R^2_{\vee\otimes}$), where $\otimes$ is associative and monotone non-decreasing in each of its arguments.

Zadeh (1971) gave T_3 and T_2 as examples of intersection operators that could be used for $\otimes$ in Definition 3.2. Bezdek and Harris (1978) studied $\vee-\Delta$ transitivity for $T_1(a,b) = a\Delta b = \max\{0, a+b-1\}$, and interpreted fuzzy $\vee-\otimes$ transitivity graphically for T_1, T_2 and T_3. More generally, a fuzzy relation R is $\oplus-\otimes$ transitive when $r_{ij} \geq r^2_{ij,\oplus\otimes} = \overset{n}{\underset{k=1}{\oplus}}(r_{ik} \otimes r_{kj})$ $\forall$ i, j, $(R \geq R^2_{\oplus\otimes})$, but this is a little too general for our purposes. In practice, the only S norm that finds applications in pattern recognition is $S_3 = \vee$. Zadeh used his concept of fuzzy $\vee-\otimes$ transitivity to extend the concept of ERs to fuzzy relations as follows:

Definition 3.3 Zadeh (1971) A fuzzy relation R is a *fuzzy similarity relation* (or fuzzy equivalence relation) if R is reflexive, symmetric and $\vee-\otimes$ transitive.

The set of all fuzzy $\vee-\otimes$ transitive similarity relations on n objects will be called $\mathbf{R}_{\vee\otimes}$. The sets $\{\mathbf{R}_{\vee\otimes}$: $\otimes$ is a T-norm$\}$ are important in relational clustering, so we formalize them as

$$\mathbf{R}_{\vee\otimes} = \{R \in \Re^{nn}:\ I_n \leq R,\ R = R^T,\ R \geq R(\vee\otimes)R\} \quad . \tag{3.4}$$

If R is crisp and $\otimes = \wedge$, the condition in Definition 3.3 guarantees that R is a crisp equivalence relation, so $\mathbf{R}_n \subseteq \mathbf{R}_{\vee\wedge}$. Zadeh noted that because $ab \leq a \wedge b$, $\mathbf{R}_{\vee\wedge} \subseteq \mathbf{R}_{\vee\bullet}$ For the choice $\otimes = \wedge$ the condition $r_{ij} \geq \overset{n}{\underset{k=1}{\vee}}(r_{ik} \wedge r_{kj})$ $\forall$ $i \neq j$ requires that the weight of any direct path in G = (V, E, R) from node i to node j be at least as large as the smallest weight of any other path from i to j. Not surprisingly, Zadeh used this to show that $R \in \mathbf{R}_{\vee\otimes} \Leftrightarrow \delta(o_i, o_j) \doteq 1 - r_{ij}$ was an ultrametric on the object set. Bezdek and Harris (1978) established that $R \in \mathbf{R}_{\vee\Delta} \Leftrightarrow \delta(o_i, o_j) \doteq 1 - r_{ij}$ was a psuedometric, and exhibited a hierarchy of seven nested sets of fuzzy similarity relations, the most important of which are $\mathbf{R}_n \subseteq \mathbf{R}_{\vee\wedge} \subseteq \mathbf{R}_{\vee\bullet} \subseteq \mathbf{R}_{\vee\Delta}$.

Zadeh (1971) also gave the first exposition of transitive closures of fuzzy relations, confining his analysis to the $\vee-\wedge$ case. More generally, the $\vee-\otimes$ transitive closure $R^\infty_{\vee\otimes}$ of fuzzy relation R is the smallest fuzzy relation containing R that is $\vee-\otimes$ transitive. $R^\infty_{\vee\otimes}$ can be computed as

$$R^{\infty}_{\vee\otimes} = R \vee R^{2}_{\vee\otimes} \vee \ldots \vee R^{n}_{\vee\otimes} \quad . \tag{3.5}$$

Furthermore, if R is reflexive and symmetric, then at worst we need only the (n-1)-st power of R,

$$R^{\infty}_{\vee\otimes} = R^{k}_{\vee\otimes}, \text{ where } k = \underbrace{\min}_{1 \le j \le n-1} \{j : R^{j}_{\vee\otimes} = R^{j+1}_{\vee\otimes}\} \quad . \tag{3.6}$$

Example 3.4 Find the $\vee - \wedge$ transitive closure of $R = \begin{bmatrix} 1.0 & 0.4 & 0.5 \\ 0.4 & 1.0 & 0.3 \\ 0.5 & 0.3 & 1.0 \end{bmatrix}$.

$$R^{2}_{\vee\wedge} = \begin{bmatrix} 1.0 & 0.4 & 0.5 \\ 0.4 & 1.0 & 0.4 \\ 0.5 & 0.4 & 1.0 \end{bmatrix} ; R^{3}_{\vee\wedge} = \begin{bmatrix} 1.0 & 0.4 & 0.5 \\ 0.4 & 1.0 & 0.4 \\ 0.5 & 0.4 & 1.0 \end{bmatrix} = R^{2}_{\vee\wedge} = R^{\infty}_{\vee\wedge}, \text{ the last}$$

equality holding because R is reflexive and symmetric.

Table 3.1 The $\vee - \otimes$ transitive closure by matrix multiplication

Store	$R \in [0,1]^{n \times n}$ (fuzzy) or $R \in \{0,1\}^{n \times n}$ (crisp)
Pick	$\otimes$= any T-norm. If $R \in \{0,1\}^{n \times n}$, $\otimes = \wedge$
Do	see algorithm below

```
R^1_{v⊗} = R
For j = 2 to n
    $R^{j}_{\vee\otimes} = R^{j-1}_{\vee\otimes}(\vee\otimes)R$
        If($I_n \le R$ and $R = R^T$ and $R^{j}_{\vee\otimes} = R^{j-1}_{\vee\otimes}$)
            $R^{\infty}_{\vee\otimes} = R^{j}_{\vee\otimes}$; Stop
Next j
$R^{\infty}_{\vee\otimes} = R^{1}_{\vee\otimes}$
For j = 2 to n
    $R^{\infty}_{\vee\otimes} = R^{j}_{\vee\otimes} \vee R^{\infty}_{\vee\otimes}$
Next j
```

Equations (3.5) and (3.6) can be used to compute the $\vee - \otimes$ transitive closure of a fuzzy relation, and like (3.2) and (3.3), they both have complexity $O(n^4)$. Faster algorithms for computing $R^{\infty}_{\vee\otimes}$ will be discussed in the next section. Table 3.1 gives the naive algorithm for $R^{\infty}_{\vee\otimes}$ based on (3.5) and (3.6). Since every crisp relation is fuzzy, this algorithm also produces the transitive closure of any crisp relation via (3.2) or (3.3) provided the T-norm is the minimum, $\otimes = \wedge$. Bezdek et al. (1986b) showed that the algorithm in Table 3.1 was correct for the six T-norms now called $T_0, T_1, T_{1.5}, T_2, T_{2.5}, T_3$. See Nguyen and Sugeno (1998) for a more complete discussion of T-norms.

3.2 Object Data to Relational Data

Before discussing algorithms that find clusters in relational data, we discuss some methods for constructing a proximity relation matrix R(X) from an object data $X \subset \Re^p$. Once this is done, clustering may be done in X (as in Chapter 2), or in R(X) using methods discussed in this chapter. Sometimes it is advantageous to make this conversion from object data to relational data.

A *similarity measure* is a real binary relation $s: O \times O \to \Re^+$. $s(o_i, o_j)$ is the similarity (for dissimilarity, we use $\delta(o_i, o_j)$) between o_i and o_j. The values $\{s(o_i, o_j)\}$ or $\{\delta(o_i, o_j)\}$ are sometimes assigned by an expert. For example, this is often the case in numerical taxonomy (Sneath and Sokal, 1973). More commonly, $\{s(o_i, o_j)\}$ or $\{\delta(o_i, o_j)\}$ are computed from characteristics - numerical or otherwise - of pairs of the objects. There are many similarity measures: for example, measures of association, resemblance, correlation, matching, and so on. Similarity measures may be based on heuristic, probabilistic, deterministic, fuzzy or semantic principles.

An object data set $X \subset \Re^p$ can be converted into a dissimilarity relation $R = [r_{ij}]$ using any metric δ on $\Re^p \times \Re^p$,

$$r_{ij} = \rho(o_i, o_j) = \delta(\mathbf{x}_i, \mathbf{x}_j),\ 1 \le i,j \le n \quad . \tag{3.7}$$

If the objects are characterized by qualitative attributes, e.g., color $\in$ {red, blue, green}, then we cannot use (3.7) directly. Using numerical representations such as 1 for red, 2 for blue and 3 for green before applying (3.7) may distort structural relationships that exist or do not exist between pairs in the qualitative data. For example, any distance relation using these numerical values for colors suggests that 3 is closer to 2 than it is to 1, even though red, blue and green are qualitatively equivalent.

Numerical representation of qualitative features can be based on binary vectors. For example, red, blue and green can be represented by the strings 100 = red, 010 = blue and 001= green. More generally, if there are p qualitative features and the i-th feature can take n_i values, then the original p features can be represented by a $\hat{p} = \sum_{i=1}^{p} n_i$ dimensional binary vector $\mathbf{x} \in \{0,1\}^{\hat{p}}$. With this representation (3.7) can be used because distances between any two vectors are unbiased and well defined. On the other hand, the feature values may or may not make sense physically, and the dimension of **x** can be very large. In our colors example, the Euclidean distance between any pair of colors is 1, so false proximity is not imposed on the data by this numerical representation.

Table 3.2 lists a few of the many different ways object data can be converted to relational data. In this table the data type real means that **x** and **y** are in $\Re^p$, and data type 0-1 means that **x** and **y** are in $\{0,1\}^p$.

Table 3.2 Some transformations of $X \subset \Re^p$ **into** $R \in \Re^{n\times n}$

Symbol	Name	Data Type	Formula for $\rho(\mathbf{x},\mathbf{y})$
δ_A	A-Norm (1.6)	Real or 0-1	$\lVert\mathbf{x}-\mathbf{y}\rVert_A = \sqrt{(\mathbf{x}-\mathbf{y})^T A(\mathbf{x}-\mathbf{y})}$
δ_q	q-Norm (1.11)	Real or 0-1	$\lVert\mathbf{x}-\mathbf{y}\rVert_q = \left[\sum_{j=1}^{p}\lvert x_j - y_j\rvert^q\right]^{\frac{1}{q}}, \ q\ge 1$
s_C	cos(x_y) if real	Real or 0-1	$\dfrac{\langle\mathbf{x},\mathbf{y}\rangle}{\lVert\mathbf{x}\rVert\ \lVert\mathbf{y}\rVert}$
s_T	Tanimoto coefficient	Real or 0-1	$\dfrac{\langle\mathbf{x},\mathbf{y}\rangle}{\langle\mathbf{x},\mathbf{x}\rangle+\langle\mathbf{y},\mathbf{y}\rangle-\langle\mathbf{x},\mathbf{y}\rangle}$
s_S	simple match	0-1	$\dfrac{a+d}{p}$
s_{DM}	double match	0-1	$\dfrac{2(a+d)}{2(a+d)+b+c}$
s_{DMM}	double mismatch	0-1	$\dfrac{a+d}{a+d+2(b+c)}$
s_J	ignore 0-0 (Jacard)	0-1	$\dfrac{a}{a+b+c}$

In Table 3.2 the last four similarity coefficients are shown as functions of a, b, c and d. These four numbers are computed from the binary vectors **x** and **y** as follows:

a = # of 1-1 matches
b = # of 1-0 mismatches
c = # of 0-1 mismatches
d = # of 0-0 matches

between the p binary coordinates in **x** and **y**. For example, if $\mathbf{x}^T = (1\,0\,0\,1\,1\,1)$, $\mathbf{y}^T = (0\,0\,1\,0\,1\,0)$, then $a = 1, b = 3, c = 1, d = 1$.

Example 3.5 For the 10 dimensional binary vectors **x** and **y** given by $\mathbf{x}^T = (0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0)$ and $\mathbf{y}^T = (1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0)$, we have a=3, b=2, c=2 and d =3. Consequently,

$$\delta_1(\mathbf{x},\mathbf{y}) = 4$$
$$\delta_2(\mathbf{x},\mathbf{y}) = 2$$
$$\delta_\infty(\mathbf{x},\mathbf{y}) = 1$$

$$s_C(\mathbf{x},\mathbf{y}) = \frac{\langle \mathbf{x},\mathbf{y}\rangle}{\|\mathbf{x}\|\ \|\mathbf{y}\|} = \frac{3}{\sqrt{5}\sqrt{5}} = 0.60$$
$$s_T(\mathbf{x},\mathbf{y}) = \frac{\langle \mathbf{x},\mathbf{y}\rangle}{\langle \mathbf{x},\mathbf{x}\rangle + \langle \mathbf{y},\mathbf{y}\rangle - \langle \mathbf{x},\mathbf{y}\rangle} = \frac{3}{5+5-3} = 0.42$$
$$s_S(\mathbf{x},\mathbf{y}) = 0.60$$
$$s_{DM}(\mathbf{x},\mathbf{y}) = 0.75;$$
$$s_{DMM}(\mathbf{x},\mathbf{y}) = 0.42;$$
$$s_J(\mathbf{x},\mathbf{y}) = 0.42$$

As an example of transforming an object data set X into relational data, we transform data set X_9 (the first two rows of Table 3.3 and Figure 3.2) by the Euclidean norm, which yields the relation R_9 in the last nine rows of Table 3.3. We show only the lower triangular part of the symmetric relation R_9. R_9 will be used to exemplify several of the clustering algorithms in subsequent sections of this chapter.

Table 3.3 X_9 and relational data $R_9 = \delta_2[X_9]$

x	1	2	2	1	4	5	4.5	5	4
y	1	1	3	3	1.5	1.5	1.5	2.5	2.5
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$	$\mathbf{x}_7$	$\mathbf{x}_8$	$\mathbf{x}_9$
$\mathbf{x}_1$	0								
$\mathbf{x}_2$	1.00	0							
$\mathbf{x}_3$	2.24	2.00	0						
$\mathbf{x}_4$	2.00	2.24	1.00	0					
$\mathbf{x}_5$	3.04	2.06	2.50	3.35	0				
$\mathbf{x}_6$	4.03	3.04	3.35	4.27	1.00	0			
$\mathbf{x}_7$	3.54	2.25	2.92	3.81	0.50	0.50	0		
$\mathbf{x}_8$	4.27	3.35	3.04	4.03	1.41	1.00	1.12	0	
$\mathbf{x}_9$	3.35	2.50	2.06	3.04	1.00	1.41	1.12	1.00	0

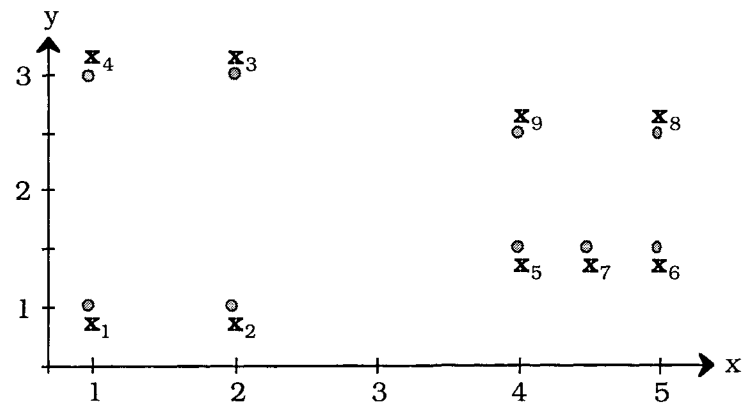


Figure 3.2 Data set $\mathbf{X}_9$

3.3 Hierarchical Methods

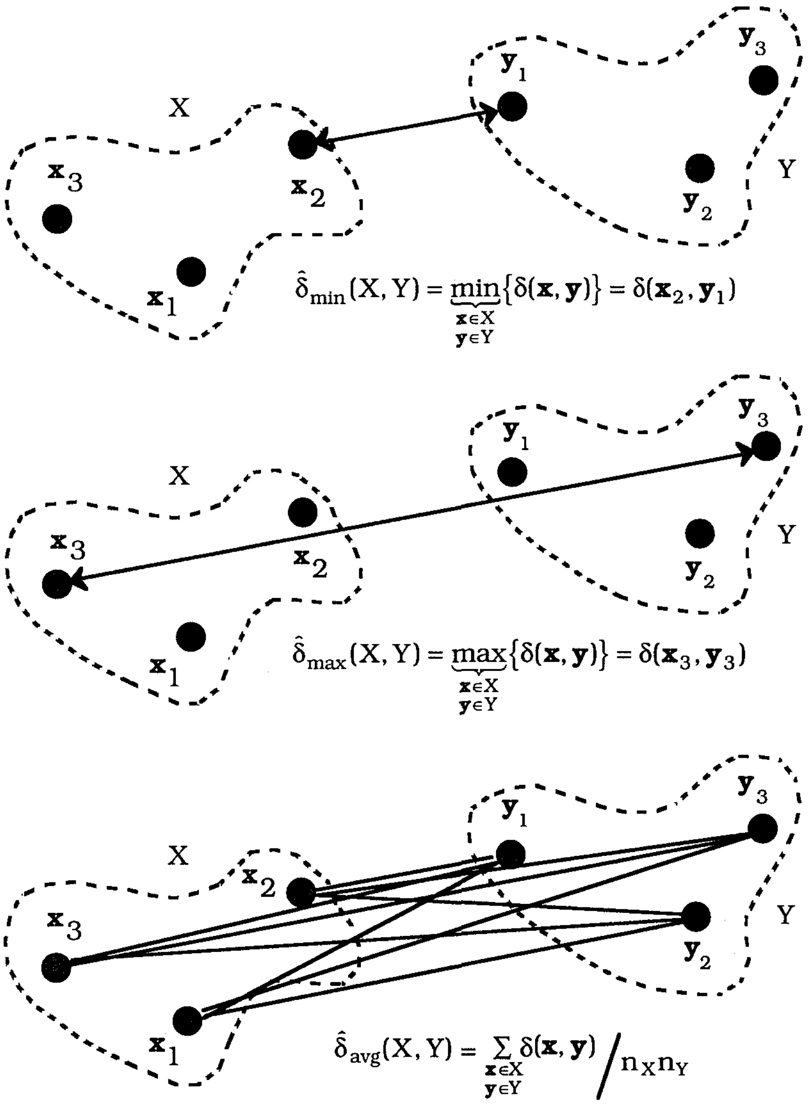
Sequential Agglomerative Hierarchical Non-Overlapping (SAHN) models (Sneath & Sokal, 1973) yield crisp clusters in fuzzy relations. Cluster merger (agglomeration, clumping) is based on a set distance $\hat{\delta}(X, Y)$ between crisp sets X and Y. The three most common set distances used are $\hat{\delta}_{min} = \hat{\delta}_1$ at (2.92), $\hat{\delta}_{max}$ and $\hat{\delta}_{avg}$.

Figure 3.3 depicts the geometric meaning of these three set distances.

$\hat{\delta}_{min}$ and $\hat{\delta}_{max}$ are the nearest and furthest distances (as measured by any metric δ on $X \times Y$) between pairs of points in $X \times Y$. $\hat{\delta}_{avg}$ is the average of all the pairwise distances between points in the two sets, and uses their cardinalities, $n_x = |X|$, $n_y = |Y|$.

We describe and illustrate the SAHN bottom up approach with relational data set R_9. Each object begins in its own singleton cluster so c = n = 9. Next, find the pair of most closely related objects, as indicated by values in relational data matrix R (find the pair of indices in R that satisfy some criterion for merger). To group the two objects in X_9 that are closest in the sense of Euclidean distance, search R_9 and find the minimum distance (0.50) : this occurs at two pairs (5,7) and (6,7). Deciding ties arbitrarily, suppose we merge points 5 and 7. At this first step the *set distance* $\hat{\delta}$ plays no role - two objects will be merged if their dissimilarity is minimum (or their similarity is maximum). We now have c=8 clusters in R_9 and

X_9, The cluster $\{\mathbf{x}_5, \mathbf{x}_7\}$ has two points, and the other 7 points are still singleton subsets of X_9.



X
Y
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\hat{\delta}_{\min}(X,Y) = \underbrace{\min}_{\substack{\mathbf{x}\in X\\ \mathbf{y}\in Y}}\{\delta(\mathbf{x},\mathbf{y})\} = \delta(\mathbf{x}_2,\mathbf{y}_1)$
$\hat{\delta}_{\max}(X,Y) = \underbrace{\max}_{\substack{\mathbf{x}\in X\\ \mathbf{y}\in Y}}\{\delta(\mathbf{x},\mathbf{y})\} = \delta(\mathbf{x}_3,\mathbf{y}_3)$
$\hat{\delta}_{\text{avg}}(X,Y) = \sum_{\substack{\mathbf{x}\in X\\ \mathbf{y}\in Y}} \delta(\mathbf{x},\mathbf{y}) \Big/ n_X n_Y$

Figure 3.3 Inter-cluster distances $\hat{\delta}_{\min}$, $\hat{\delta}_{\max}$ and $\hat{\delta}_{\text{avg}}$

Next, merge the two clusters in the current set of 8 that are closest to each other in the sense of *set distance* $\hat{\delta}$. Two other singletons might merge, or perhaps one or more singletons will join $\{\mathbf{x}_5, \mathbf{x}_7\}$. In any case, different $\hat{\delta}$'s may result in different ensuing sequences of crisp clusters. This merging process continues until c=1.

The SAHN procedure is hierarchical in that c proceeds from c=n to c=1, nesting more and more objects together as it proceeds. Since the process is non-iterative, there is no need for initialization, and the clusters found at each value of c are unique. The algorithms that use $\hat{\delta}_{min}$, $\hat{\delta}_{max}$ and $\hat{\delta}_{avg}$ are known as the *single, complete* and *average linkage* clustering algorithms respectively. The linkage methods are well defined for any relational data matrix that has positive real-valued proximities. In particular, these algorithms generate hierarchies of *crisp* partitions in the object set from arbitrary *fuzzy* relational data.

Partition hierarchies produced by single and complete linkage applied to R_9 are displayed as dendrograms (trees) in Figure 3.4. The left half of Figure 3.4 shows the dendrogram obtained by single linkage for R_9. The vertical scale is set distance $\hat{\delta}_{min}$. This indicates the level at which clusters are merged. At the top of the tree each point is in its own cluster and c=9. For single linkage at the first stage points (5 and 7) and (6 and 7) are possible candidates for merging. Breaking ties arbitrarily, suppose we merge (5 and 7) first, and then this cluster merges with 7 at the same level in the next step.

Then, at level $\hat{\delta}_{min}$=1, points 1 and 2 merge, as do points 3 and 4, and 5,7,6 merge with 8 and 9. In this example single linkage never produces, e.g., c=7 clusters if we generate clusters by cutting the dendrogram horizontally. However, in the process of development of the dendrogram a unique (up to arbitrary breaking of ties) partition is generated for every possible value of c, $1 \le c \le n$. Cutting the dendrogram horizontally at any level in-between merger levels shows c as the number of vertical lines cut. In Figure 3.4 at $\hat{\delta}_{min}$=1.60, the single linkage cut shows 3 clusters resulting in the crisp 3 - partition $\{1,2\} \cup \{3,4\} \cup \{5,6,7,8,9\}$.

The right half of Figure 3.4 shows the complete linkage dendrogram using set distance $\hat{\delta}_{max}$. Comparing the single and complete linkage solutions shows that the two hierarchies are structurally quite different. For example, c = 3 at $\hat{\delta}_{min}$ =1.60, but c = 4 for $\hat{\delta}_{max}$= 1.60.

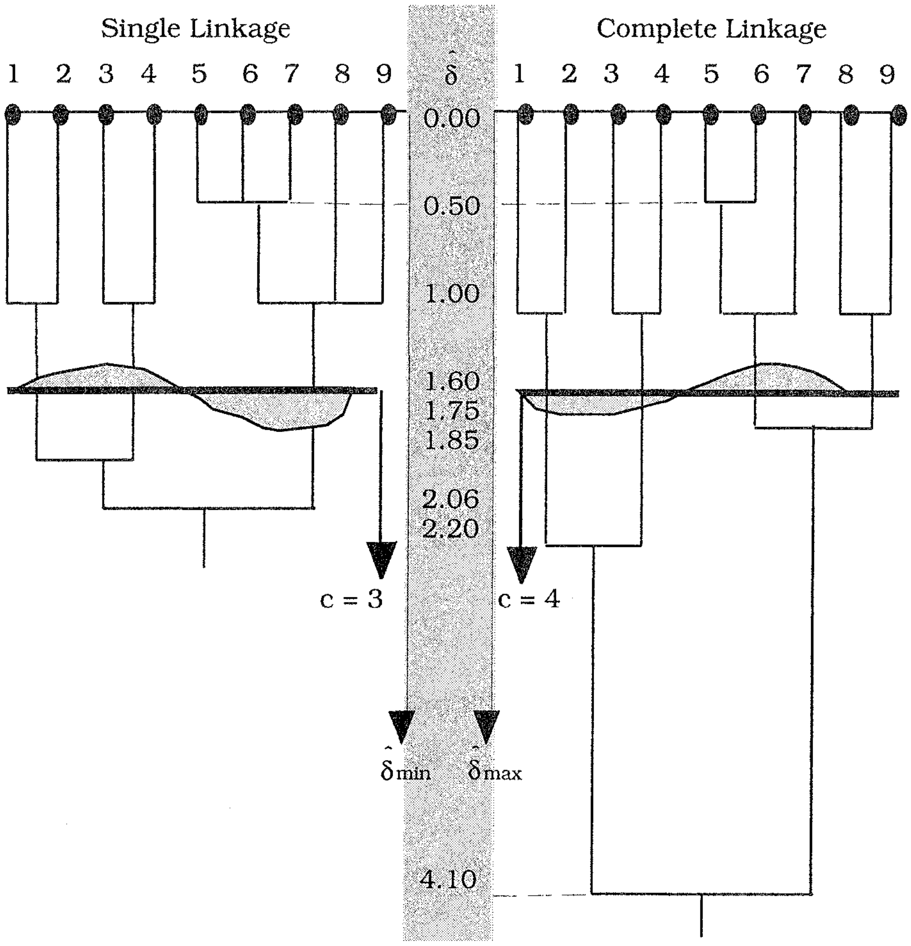


Figure 3.4 Single and complete linkage dendrograms on R_9

In terms of the fuzzy graph G = (V, E, R), the single linkage algorithm can be interpreted as follows. At initialization, each object (node) is in its own singleton cluster; this corresponds to a forest of n trees in G. At any succeeding time in the procedure, say at c = q, the graph is composed of q subtrees that are again a forest in G. Each merger of two clusters via $\hat{\delta}_{min}$ corresponds to adding a minimum weight edge between the two closest subtrees, thereby creating a forest with one less tree. At termination of single linkage there is c = 1 cluster. In terms of G, the sequence of linking edges is at this point a *minimal spanning tree* (MST). This is essentially Kruskal's (1956) MST algorithm, which has complexity $O(|E|\log_2|V|)$ for a relation on $|V| = n$ objects that has $|E|$ edges.

3.4 Clustering by decomposition of fuzzy relations

Clustering in fuzzy relational data often utilizes α-cuts of R. An *α-cut* or crisp *α-level-set*, $\alpha \in (0,1]$, of a fuzzy relation R is the crisp binary relation $\mathcal{R}_\alpha = \{(o_i, o_j) \in O \times O : r_{ij} = \rho(o_i, o_j) \geq \alpha\}$. As α runs through (0, 1], the α- cuts of R form a nested sequence of crisp relations such that $\alpha_1 \geq \alpha_2 \Rightarrow \mathcal{R}_{\alpha_1} \subseteq \mathcal{R}_{\alpha_2}$, i.e., $R_{\alpha_1} \leq R_{\alpha_2}$.

In this section we give two methods for fuzzy relational data that yield sets of *crisp* c-partitions of the objects. One method produces hierarchically nested clusters while the second approach does not. We begin with clustering in the max-min transitive closure of R. Given a fuzzy similarity relation, Zadeh's (1971) resolution identity can be used to generate nested partitions of the objects. The algorithm is based on:

Theorem Z (Zadeh, 1971): Any fuzzy relation R on $X \times X$ has the decomposition $R = \bigvee_\alpha R_{(\alpha)}$, $0 < \alpha \leq 1$ where $R_{(\alpha)} = \alpha R_\alpha$ is the fuzzy relation defined by

$$R_{(\alpha)}(x,y) = \begin{cases} \alpha; & \text{if } R(x,y) \geq \alpha \\ 0; & \text{otherwise} \end{cases} . \tag{3.8}$$

We give an example illustrating the use of Zadeh's theorem to decompose a fuzzy relation on 3 objects.

Example 3.6

$$R = \begin{bmatrix} 0.5 & 0.0 & 0.7 \\ 0.3 & 1.0 & 0.0 \\ 0.5 & 0.3 & 1.0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.3 & 0.0 & 0.3 \\ 0.3 & 0.3 & 0.0 \\ 0.3 & 0.3 & 0.3 \end{bmatrix} \vee \begin{bmatrix} 0.5 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 \\ 0.5 & 0.0 & 0.5 \end{bmatrix} \vee \begin{bmatrix} 0.0 & 0.0 & 0.7 \\ 0.0 & 0.7 & 0.0 \\ 0.0 & 0.0 & 0.7 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= 0.3 \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \vee 0.5 \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \vee 0.7 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \vee 1 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

✦

If R is a $\vee - \wedge$ transitive *similarity* relation, then R_α is an *equivalence* relation on O. To see this, note that reflexivity and symmetry are preserved for all α. Now suppose $(i,j) \in R_{(\alpha)}$ and $(j,k) \in R_{(\alpha)}$, then $r_{ij} \geq \alpha$ and $r_{jk} \geq \alpha$. Since R is $\vee - \wedge$ transitive $r_{ik} \geq \max_{s=1,\ldots,n} \{\min\{r_{is}, r_{sk}\}\} \Rightarrow r_{ik} \geq \min\{r_{ij}, r_{jk}\} \Rightarrow r_{ik} \geq \alpha \Rightarrow (i,k) \in R_{(\alpha)}$. Therefore for $\vee - \wedge$ transitive similarity relations on n objects theorem Z will generate a unique set of nested crisp partitions of the objects.

Example 3.7 $R = \begin{bmatrix} 1.0 & 0.8 & 0.4 & 0.8 & 0.8 \\ 0.8 & 1.0 & 0.4 & 0.8 & 0.9 \\ 0.4 & 0.4 & 1.0 & 0.4 & 0.4 \\ 0.8 & 0.8 & 0.4 & 1.0 & 0.8 \\ 0.8 & 0.9 & 0.4 & 0.8 & 1.0 \end{bmatrix}$ on $X = \{x_1, x_2, x_3, x_4, x_5\}$

is a $\vee - \wedge$ transitive *similarity* relation. Using theorem Z,

$$R = 0.4\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \vee 0.8\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\vee 0.9\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \vee 1\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The ERs in this decomposition yield the following partitions :

$U_{0.4} = \{x_1, x_2, x_3, x_4, x_5\}$,

$U_{0.8} = \{x_1, x_2, x_4, x_5\} \cup \{x_3\}$,

$U_{0.9} = \{x_1\} \cup \{x_2, x_5\} \cup \{x_3\} \cup \{x_4\}$,

$U_1 = \{x_1\} \cup \{x_2\} \cup \{x_3\} \cup \{x_4\} \cup \{x_5\}$.

Figure 3.5 illustrates this graphically with a dendrogram. The vertical axis corresponds to the values of α at which clusters are merged bottom up, starting at c = 5 and α = 1.

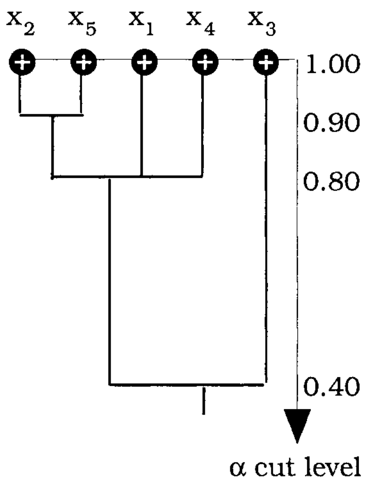


Figure 3.5 α cut tree for Example 3.7

Zadeh (1971) used matrix multiplication ($O(n^4)$) to find the max-min transitive closure of a symmetric, reflexive fuzzy relation. Tamura et al. (1971) gave a slightly different method based on successive approximations that is at worst $O(n^3 \log n)$ and at best $O(n^3)$. Dunn (1974b) showed that the hierarchies generated by Tamura et al.'s (1971) method were equivalent to single linkage hierarchies, and that the equivalence classes used by Tamura et al. could be generated from a family of nested graphs obtained by deleting edges in a maximal spanning tree defined on O, assuming r_{ij} as the edge weight between o_i and o_j. Dunn gave an algorithm for constructing the max-min transitive closure of a symmetric, reflexive fuzzy relation based on maximal spanning trees and maximal capacity routes that is $O(n^2)$. Other authors have studied construction of the transitive closure (see Kandel and Yelowitz, 1974 or Larsen and Yager, 1990), but none are asymptotically faster than Dunn's method. A result giving the equivalence between partitions generated by four relational algorithms is :

Theorem M (Miyamoto, 1990)

$O = \{o_1, \ldots, o_n\}$, $R: O \times O \to [0,1]$ is a symmetric, reflexive fuzzy relation. For arbitrary $\alpha \in [0,1]$, the crisp partitions of O obtained by the following four schemes are identical.

(i) Perform hierarchical clustering using the single linkage algorithm. Cut the resulting dendrogram at level α to generate a hard partition of $O = \{o_1, \ldots, o_n\}$.

(ii) $G = (V, E, R)$ is a complete graph. (R need not be reflexive and symmetric). Let the *maximal* spanning tree of G be $\hat{G}$. Let $\hat{G}_\alpha$ be the graph that is obtained by deleting all edges in E with weights $r_{ij} < \alpha$ (edges in $\hat{G}_\alpha$ satisfy $r_{ij} \geq \alpha$). Let the connected components of $\hat{G}_\alpha$ be denoted by subgraphs $\{\hat{G}^i_\alpha;\ i = 1, \ldots, k\}$. Then the vertices of the connected components of $\hat{G}_\alpha$ are a partition of O.

(iii) $G = (V, E, R)$, and $G_\alpha = (O, E_\alpha, R_\alpha)$ is any α-cut of G. If R is reflexive and symmetric, the vertices of the connected components in G_α are a partition of O.

(iv) Let the transitive closure of R be R^∞. Then the α-cuts of R^∞ induce a partition of O.

The method of this section has been studied in information retrieval, where it has been used, for example, to construct fuzzy thesauri. Good articles related to this include: Radecki (1976), Miyamoto et al. (1983), Zenner et al. (1985), Bezdek et al. (1986b) and Larsen and Yager (1993).

Theorem Z affords a way to decompose a fuzzy similarity relation into a nested hierarchy of crisp partitions of O with associated scalars α in [0, 1]. A different decompositional method was suggested by Bezdek and Harris (1978, 1979). Recall that $\mathbf{R}_n$ is the set of all hard ERs on O,

$$\mathbf{R}_n = \{R \in \Re^{nn} : r_{ij} \in \{0,1\}, I_n \leq R, R = R^T, R = R^2_{\vee\wedge}\} \quad . \tag{3.9}$$

Let conv($\mathbf{R}_n$) be the *convex hull* of $\mathbf{R}_n$. $R \in \text{conv}(\mathbf{R}_n)$ guarantees at least one convex decomposition

$$R = \sum_{k=1}^{l} c_k R_k, \quad R_k \in \mathbf{R}_n \quad \forall \ k \quad , \tag{3.10}$$

where $\{c_k\}$ in [0, 1] are convex weights, l is the *length* of the convex decomposition, $\sum_{k=1}^{l} c_k = 1$, and each $R_k \in \mathbf{R}_n$ is a hard ER and hence, isomorphic to a hard c-partition of O. Equation (3.10) holds for any

$R \in \mathrm{conv}(\mathbf{R}_n)$. Unlike decomposition by resolution of the transitive closure of R, the set of partitions generated by convex decomposition is *not* a hierarchy of nested partitions.

Example 3.8

$R = \begin{bmatrix} 1.0 & 0.3 & 0.6 & 0.0 \\ 0.3 & 1.0 & 0.7 & 0.0 \\ 0.6 & 0.7 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$ is reflexive and symmetric but not max-min transitive. Because of the special structure of column 4, R has a *unique* convex decomposition :

$$R = 0.4\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + 0.3\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + 0.3\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The ERs in this convex decomposition yield the partitions

$$\begin{aligned} U_{0.4} &= \{x_1\} \cup \{x_2, x_3\} \cup \{x_4\} & ; \\ U_{0.3} &= \{x_1, x_2, x_3\} \cup \{x_4\} & ; \text{and} \qquad (3.11) \\ U_{0.3} &= \{x_1, x_3\} \cup \{x_2\} \cup \{x_4\} & . \end{aligned}$$

There are two partitions for $c_k = 0.3$, one with c = 2 clusters, and one with c = 3 clusters. For comparison we decompose the max-min transitive closure of R.

$$R^\infty = \begin{bmatrix} 1.0 & 0.6 & 0.6 & 0.0 \\ 0.6 & 1.0 & 0.7 & 0.0 \\ 0.6 & 0.7 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$= 0.6\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vee 0.7\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vee 1.0\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The ERs in this decomposition by theorem Z yield the partitions

$$\begin{aligned} U_{0.6} &= \{x_1, x_2, x_3\} \cup \{x_4\} && ; \\ U_{0.7} &= \{x_1\} \cup \{x_2, x_3\} \cup \{x_4\} && ; \text{ and} \\ U_{1.0} &= \{x_1\} \cup \{x_2\} \cup \{x_3\} \cup \{x_4\} && . \end{aligned} \tag{3.12}$$

Comparing the partitions at (3.11) with those at (3.12), convex decomposition suggests $U_{0.4} = \{x_1\} \cup \{x_2, x_3\} \cup \{x_4\}$ with c = 3 as the best description of the structure in the data. The hierarchy based on transitive closure does not have a preferred value for c based on the values of α. However, at c = 3, the unique choice suggested by (3.12) is $U_{0.7} = \{x_1\} \cup \{x_2, x_3\} \cup \{x_4\}$, which is the partition "most highly recommended" by convex decomposition in the sense that its convex weight is maximum. Notice that convex decomposition never produces a partition for c = n.

Since (3.10) is applicable only to R's in conv($\mathbf{R}_n$), the important open question of how to recognize when this is true must be solved before this method is generally useful. Any R admitting decomposition (3.10) must be symmetric and reflexive. Bezdek and Harris (1978) showed that $\mathbf{R}_{\vee\wedge} \subset \text{conv}(\mathbf{R}_n) \subset \mathbf{R}_{\vee\Delta}$ for n > 3, where $\mathbf{R}_{\vee\wedge}, \mathbf{R}_{\vee\Delta}$ are the sets of fuzzy similarity relations defined at (3.4) that are $\vee - \wedge$ and $\vee - \Delta$ transitive, respectively. Thus, every fuzzy similarity relation in the sense of $\vee - \wedge$ transitivity on more than three objects also has at least one convex decomposition. Bezdek and Harris (1979) give three algorithms for the computation of convex decompositions of fuzzy *c-partitions* into crisp c-partitions, and show several ways to construct relations from them, but do not solve the problem of how to usefully characterize conv($\mathbf{R}_n$). The related question of when a convex decomposition is unique is, to our knowledge, also unsolved.

3.5 Relational clustering with objective functions

In this section we describe several models and algorithms which generate a *fuzzy* partition from relational data based on minimization of an objective function. These models all assume R to be a pairwise *dissimilarity* relation between objects in O. The first method of this type was given by Ruspini (1970). Here we discuss four representative models due to Roubens (1978), Windham (1985), Hathaway et al. (1989) and Hathaway and Bezdek (1994b).

A. The Fuzzy Non Metric (FNM) model

Rouben's (1978) model assumes that R is a dissimilarity relation satisfying three conditions : $\forall i, j,\ r_{ij} \geq 0,\ r_{ii} = 0$ and $r_{ij} = r_{ji}$. For example, every relation matrix produced from $X \subset \Re^p$ using (3.7) satisfies these three conditions. In order to partition the objects to c fuzzy clusters, Roubens proposed the objective function model

$$\underbrace{\min}_{U \in M_{fcn}} \left\{ K_{FNM}(U) = \sum_{i=1}^{c}\sum_{k=1}^{n}\sum_{j=1}^{n} u_{ik}^2 u_{ij}^2 r_{kj} \right\} \quad . \tag{3.13}$$

Rewrite the objective function in (3.13) as

$$K_{FNM}(U) = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^2 \left(\sum_{j=1}^{n} u_{ij}^2 r_{kj} \right) = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^2 D_{ik} \quad , \tag{3.14}$$

where

$$D_{ik} = \sum_{j=1}^{n} u_{ij}^2 r_{kj} \quad . \tag{3.15}$$

Using the LaGrange multiplier technique under the assumption that $D_{ik} > 0 \ \forall i, k$, Roubens obtained the usual first order necessary conditions for optimality of U,

$$u_{ik} = \left(\sum_{j=1}^{c} \frac{D_{ik}}{D_{jk}} \right)^{-1} ; \ 1 \leq i \leq c;\ 1 \leq k \leq n \quad . \tag{3.16}$$

(3.16) is just an instance of (2.7a) when $D_{ik} = \|\mathbf{x}_k - \mathbf{v}_i\|_A^2$ is replaced by $D_{ik} = \sum_{j=1}^{n} u_{ij}^2 r_{kj}$ and m = 2. An alternating optimization scheme based on (3.15) and (3.16) can be used to iteratively minimize K_{FNM}. Initialization is made on U, the $\{D_{ik}\}$ in (3.15) are computed with it, U is updated with (3.16), and then return to (3.15) results in a new set of values for $\{D_{ik}\}$. This algorithm is summarized in Table 3. 4.

Table 3.4 The fuzzy non-metric (FNM-AO) algorithm

Store	Relation matrix $R = [r_{jk}]_{n \times n}$ $\forall i, j, \; r_{ij} \geq 0, \; r_{ii} = 0$ and $r_{ij} = r_{ji}$
Pick	☛ number of clusters : $1 < c < n$ ☛ maximum number of iterations : T ☛ termination threshold : $0 < \varepsilon$
Guess	Initial partition $U_0 \in M_{fcn}$

Iterate	$t \leftarrow 1$
	While $(t \leq T)$
	  For k=1 to n
	    For i=1 to c
	      $D_{ik,t-1} = \sum_{j=1}^{n} u_{ij,t-1}^{2} r_{kj}$
	    Next i
	    For s=1 to c
	      $u_{sk,t} = \left(\sum_{j=1}^{c} \frac{D_{sk,t-1}}{D_{jk,t-1}} \right)^{-1}$
	    Next s
	  Next k
	  If $\lvert K_{FNM}(U_t) - K_{FNM}(U_{t-1}) \rvert < \varepsilon$ Then Exit While
	  $t \leftarrow t+1$
	End while
	$U \leftarrow U_t$

Using an argument such as that in Diday (1975), it can be shown that the FNM algorithm converges to a local minimum of K_{FNM}. Libert and Roubens (1982) give some extensions and additional material on cluster validity associated with this model.

B. The Assignment-Prototype (AP) Model

Windham's (1985) AP algorithm assumes that R satisfies the same conditions as the FNM model. Suppose the objects are to be grouped into c crisp clusters $X_1, \ldots, X_c$. Windham assumes that for each cluster X_i there is an object (o_{k_i}) which is the best representative or prototype of that cluster. Then the quality of the clustering can be measured by

$$\tau = \sum_{i=1}^{c} \left(\sum_{o_j \in X_i} r_{jk_i} \right) \quad , \tag{3.17}$$

The smaller the value of τ, the more similar the objects in X_i are to the prototype of the class. Minima of τ point to crisp partitions of O that are well represented by their prototypes. Optimization of τ in (3.17) produces hard partitions. Windham modified τ so that it seeks fuzzy partitions U as part of optimal pairs (U, T) of the AP model

$$\underbrace{\min}_{M_{fcn} \times M^*_{fcn}} \left\{ K_{AP}(U,T) = \sum_{i=1}^{c} \sum_{k=1}^{n} \sum_{j=1}^{n} u_{ik}^2 t_{ij}^2 r_{kj} \right\} \quad \text{, where} \tag{3.18}$$

$$U \in M_{fcn} \text{ and } M^*_{fcn} = \left\{ T \in \Re^{cn} : \mathbf{T}_{(k)} \in N_{fn} \ \forall\, k \right\} \quad . \tag{3.19}$$

In (3.19) $\mathbf{T}_{(k)}$ is the k-th *row* of the $c \times n$ matrix T. In component form, the constraints on elements of T are that each row sum to one, $\sum_{k=1}^{n} t_{ik} = 1 \ \forall \ i = 1,\ldots,c$; and that $t_{ik} \geq 0 \ \forall \ i,k$.

U in (3.18) is a fuzzy partition of O, so u_{ik} gives the degree to which o_k belongs to fuzzy cluster u_i. The entry t_{ik} represents the degree to which o_k represents (or is typical of) the i-th prototype. Windham calls U an *assignment matrix,* and T the *prototype weight matrix.* Using the LaGrange multiplier technique twice (holding T fixed and optimizing on U, and then conversely) results in the usual first order necessary conditions

$$t_{i\ell} = \left(1 \Big/ \sum_k u_{ik}^2 r_{k\ell} \right) \Big/ \sum_m \left(1 \Big/ \sum_k u_{ik}^2 r_{km} \right) \quad \forall\, i,\ell \quad \text{, and} \tag{3.20a}$$

$$u_{ik} = \left(1 \Big/ \sum_\ell t_{i\ell}^2 r_{k\ell} \right) \Big/ \sum_j \left(1 \Big/ \sum_\ell t_{j\ell}^2 r_{k\ell} \right) \quad \forall\, i,k \quad . \tag{3.20b}$$

These equations can also be derived directly from (2.7a) in Chapter 2 by grouping the fixed variables for each problem together and calling them D_{ik} as in FCM for the special case m=2. Estimates of optimal pairs (U, T) are obtained through alternating optimization between (3.20a) and (3.20b). The AP algorithm is summarized in Table 3.5. Windham and Roubens both advocate termination when successive values of the objective function become close, rather than terminating when successive estimates of the fuzzy partition are close. However, termination on the closeness of successive estimates of U is better because a proper choice for ε when terminating on successive values of the objective function is very delicate. This is because the correct choice for ε depends strongly on the actual value of a local minimum in the attracting neighborhood, which is, of course, unknown.

Table 3.5 The assignment-prototype (AP-AO) algorithm

Store	Relation matrix $R = \left[r_{jk}\right]_{n \times n}$; $\forall\, i, j,\ r_{ij} \geq 0,\ r_{ii} = 0$ and $r_{ij} = r_{ji}$
Pick	☛ number of clusters : $1 < c < n$ ☛ maximum number of iterations : M ☛ termination threshold : $0 < \varepsilon$
Guess	Initial partition $U_0 \in M_{fcn}$

Iterate	$t \leftarrow 1$
	While ($t \leq M$)
	    For i=1 to c
	        For ℓ=1 to n
	            $t_{i\ell,t} = \left(1 \Big/ \sum_{k=1}^{n} u_{ik,t-1}^{2} r_{k\ell} \right) \Big/ \sum_{m=1}^{n} \left(1 \Big/ \sum_{k=1}^{n} u_{ik,t-1}^{2} r_{km} \right)$
	        Next ℓ
	    Next i
	    For i= 1 to c
	        For k= 1 to n
	            $u_{ik,t} = \left(1 \Big/ \sum_{\ell=1}^{n} t_{i\ell,t}^{2} r_{k\ell} \right) \Big/ \sum_{j=1}^{c} \left(1 \Big/ \sum_{\ell=1}^{n} t_{j\ell,t}^{2} r_{k\ell} \right)$
	        Next k
	    Next i
	    If $\left\|K_{AP}(U_t, T_t) - K_{AP}(U_{t-1}, T_{t-1})\right\| < \varepsilon$ Then Exit While
	    Else $t \leftarrow t+1$
	End while
	$(U, T) \leftarrow (U_t, T_t)$

Example 3.9 Windham (1985) considered the (11 x 11) symmetric relational matrix R_{11} listed in Table 3.6. Entries for object 6 are highlighted because this object plays a special role when interpreting the output of the AP algorithm.

R_{11} was generated from a two dimensional object data set X_{11}. The coordinates shown in Table 3.7 are *roughly* correct. Windham rounded off the squared Euclidean distance between each pair of points in Table 3.7 to the nearest integer to obtain the (relational data) integers in Table 3.6. For example, the squared distance between points 1 and 3 in Table 3.7 is 2.77, but in Table 3.6 this value is rounded up to 3.

Table 3.6 Windham's dissimilarity data R_{11}

	1	2	3	4	5	**6**	7	8	9	10	11
1	0	6	3	6	11	**25**	44	72	69	72	100
2		0	3	11	6	**14**	28	56	47	44	72
3			0	3	3	**11**	25	47	44	47	69
4				0	6	**14**	28	44	47	56	72
5					0	**3**	11	28	25	28	44
6						**0**	3	14	11	14	25
7							0	6	3	6	11
8								0	3	11	6
9									0	3	3
10										0	6
11											0

Table 3.7 (Approximate) coordinates of X_{11}

Datum	x	y
$\mathbf{x}_1$	-5.00	0.00
$\mathbf{x}_2$	-3.34	1.67
$\mathbf{x}_3$	-3.34	0.00
$\mathbf{x}_4$	-3.34	-1.67
$\mathbf{x}_5$	-1.67	0.00
$\mathbf{x}_6$	**0.00**	**0.00**
$\mathbf{x}_7$	1.67	0.00
$\mathbf{x}_8$	3.34	1.67
$\mathbf{x}_9$	3.34	0.00
$\mathbf{x}_{10}$	3.34	-1.67
$\mathbf{x}_{11}$	5.00	0.00

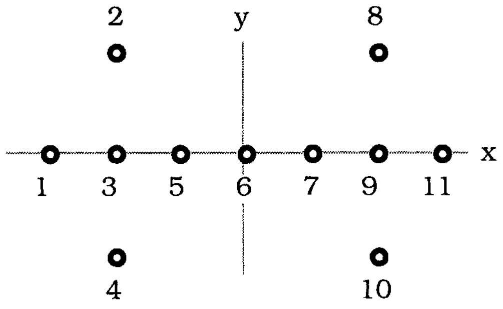


Figure 3.6 Data set X_{11}

Figure 3.6 displays the 11 points in Table 3.7. Although the AP algorithm uses only relational data R_{11} interpretation of the results is facilitated by knowing the (approximate) structure of the object data from which it was built.

The visual configuration of X_{11} suggests that it possesses c=2 clusters, (the left and right 5-point sets), with a bridge or neck between them provided by object 6. We initialize the AP algorithm with the 2-partition

$$U_0 = \begin{pmatrix} \kappa & \kappa & \kappa & \kappa & \kappa & \upsilon & \upsilon & \upsilon & \upsilon & \upsilon & \upsilon \\ \upsilon & \upsilon & \upsilon & \upsilon & \upsilon & \kappa & \kappa & \kappa & \kappa & \kappa & \kappa \end{pmatrix} \text{, where} \qquad (3.21)$$

κ=0.75 and ν = 0.25. Using other protocols specified in Windham (1985) leads to the outputs shown in Table 3.8. The rows of U and T are shown transposed, and as required, columns of U and rows of T sum to 1.

Table 3.8 (U, T) produced by AP-AO for $\mathbf{R}_{11}$

	Memberships		Prototype weights	
Datum	$\mathbf{U}^T_{(1)}$	$\mathbf{U}^T_{(2)}$	$\mathbf{T}^T_{(1)}$	$\mathbf{T}^T_{(2)}$
$\mathbf{x}_1$	0.92	0.08	0.13	0.01
$\mathbf{x}_2$	0.90	0.10	0.14	0.02
$\mathbf{x}_3$	0.95	0.05	0.27	0.02
$\mathbf{x}_4$	0.90	0.10	0.14	0.02
$\mathbf{x}_5$	0.86	0.14	0.16	0.03
$\mathbf{x}_6$	0.50	0.50	0.06	0.06
$\mathbf{x}_7$	0.14	0.86	0.03	0.16
$\mathbf{x}_8$	0.10	0.90	0.02	0.14
$\mathbf{x}_9$	0.05	0.95	0.02	0.27
$\mathbf{x}_{10}$	0.10	0.90	0.02	0.14
$\mathbf{x}_{11}$	0.08	0.92	0.01	0.13

The membership values in Table 3.8 are symmetric with respect to the y axis in Figure 3.6. Objects 3 and 9 have the highest memberships in clusters 1 and 2, respectively. The prototype assignment values suggest that object 3 is the best representative for cluster 1, and that object 9 is the best prototype for cluster 2. Visual inspection of X_{11} agrees with this.

Diday (1975) proposed a problem that seeks crisp clusters in R based on minimizing an objective function which is quite similar to the AP objective function,

$$K(U,T) = \sum_i \sum_k \sum_j u_{ik} t_{ij} r_{kj} \quad , \tag{3.22}$$

subject to $u_{ik}, t_{ik} \in \{0,1\} \ \forall \ i,k$, $\sum_{i=1}^{c} u_{ik} = 1 \ \forall \ k$ and $\sum_{k=1}^{n} t_{ik} = n_i \ \forall \ i$. Relation r_{kj} is a measure of dissimilarity between o_k and o_j satisfying $\forall i,j$, $r_{ij} \ge 0$, $r_{ii} = 0$ and $r_{ij} = r_{ji}$, where n_i is the number of points in crisp cluster i.

C. The relational fuzzy c-means (RFCM) model

Recall from Chapter 2 that for object data $X \subset \Re^p$, the FCM clustering model is defined by the optimization problem

$$\underbrace{\min}_{(U,\mathbf{V})} \left\{ J_m(U,\mathbf{V}) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik}^m D_{ik} \right\} \quad . \tag{3.23}$$

Equation (2.23b) shows the reformulation of J_m in terms of **V** alone when $D_{ik} = \|\mathbf{x}_k - \mathbf{v}_i\|_A^2$. For relational clustering Hathaway et al. (1989) applied the opposite-case reformulation to J_m, using (2.7a) to eliminate **V** instead of U from J_m. The effect of this substitution is to restrict J_m to a surface in (U,**V**) space which satisfies two important properties: (i) J_m is a function of U alone on this surface; and (ii) by the reformulation theorem, this surface contains all minimizing pairs (U*,**V***) of J_m. We represent the reformulation of J_m in terms of U as K_m. After some algebraic manipulation K_m takes the form

$$K_m(U) = \sum_{i=1}^{c} \left(\sum_{j=1}^{n} \sum_{k=1}^{n} \left(u_{ij}^m u_{ik}^m \|\mathbf{x}_j - \mathbf{x}_k\|_A^2 \right) \Big/ \left(2 \sum_{t=1}^{n} u_{it}^m \right) \right). \tag{3.24}$$

Equation (3.24) can be rewritten as

$$K_m(U) = \sum_{i=1}^{c} \left(\sum_{j=1}^{n} \sum_{k=1}^{n} \left(u_{ij}^m u_{ik}^m r_{jk} \right) \Big/ \left(2 \sum_{t=1}^{n} u_{it}^m \right) \right) \quad \text{, where} \tag{3.25a}$$

$$r_{jk} = \|\mathbf{x}_j - \mathbf{x}_k\|_A^2 \quad . \tag{3.25b}$$

By the reformulation theorem, minimization of K_m at (3.25) is equivalent to minimization of J_m in (3.23) or R_m at (2.23b) provided R satisfies (3.25b). Condition (3.25b) holds for some $X \subset \Re^p$ and positive definite $A \neq I$ if and only if it holds for some $Y \subset \Re^p$ and $A = I$. When there exists a set of n object data in some dimension p such that the pairwise distances define R, we say that K_m is the *relational dual* of J_m.

Table 3.9 The relational fuzzy c-means (RFCM-AO) algorithms

Store	Relation matrix $R = \left[r_{jk}\right]_{n \times n}$ satisfying, $\forall\, i, j, k$, $r_{ij} \geq 0$, $r_{ii} = 0$, $r_{ij} = r_{ji}$ *and* $r_{jk} = \left\| \mathbf{x}_j - \mathbf{x}_k \right\|_A^2$
Pick	☛ number of clusters : $1 < c < n$ ☛ maximum number of iterations : T ☛ weighting exponent : $1 < m < \infty$ ☛ termination threshold : $0 < \varepsilon$
Guess	Initial partition $U_0 \in M_{fcn}$
Iterate	see below

Iterate

```
$t \leftarrow 1$
While ($t \leq T$)
    For i=1 to c
        $\mathbf{v}_{i,t} = \left(u_{i1,t-1}^m, \ldots, u_{in,t-1}^m\right)^T \Big/ \sum_{k=1}^{n} u_{ik,t-1}^m$
    Next i
    For k= 1 to n
        For i=1 to c
            $d_{ik,t}^2 = (R\mathbf{v}_{i,t})_k - ((\mathbf{v}_{i,t})^T R\mathbf{v}_{i,t})/2$
        Next i
        If $d_{ik,t} > 0 \ \forall i$
        Then $u_{ik,t} = 1 \Big/ \left[\sum_{j=1}^{c} (d_{ik,t}/d_{jk,t})^{2/(m-1)}\right]$
        Else $u_{ik,t} = \begin{cases} 0 & ; d_{ik,t} > 0 \\ \alpha_{ik,t} & ; d_{ik,t} \leq 0; \alpha_{ik,t} \in [0,1], \sum_{i=1}^{c} \alpha_{ik,t} = 1 \end{cases}$
    Next k
    If $\left|K_m(U_t) - K_m(U_{t-1})\right| < \varepsilon$, Then Exit While
    $t \leftarrow t + 1$
End while
$U \leftarrow U_t$
```

RFCM implicitly assumes that R is obtained from (inner product) distances between pairs of object data. It is important to note that R is not necessarily a fuzzy relation. R must satisfy the same requirements as the AP and FNM models, and (3.25b) as well. First order necessary conditions for minimization of K_m lead to the alternating optimization scheme called the relational fuzzy c-means (RFCM) algorithms which are summarized in Table 3.9.

Protocols needed in case $d_{jk} = 0$ for some (j, k) are the same as for FCM. Let $X \subset \Re^p$ have n points and $R = \left[r_{jk} = \|\mathbf{x}_j - \mathbf{x}_k\|_A^2 \right]$ be the associated $n \times n$ relation matrix. If started at the same initial partition, FCM and RFCM yield identical iterate sequences of partition matrices (Hathaway et al., 1989). The update equation for U in FCM and RFCM has the same functional form, but the vectors $\{\mathbf{v}_i\}$ in the iteration of Table 3.9 lie in $\Re^n$, not in $\Re^p$ as they do for FCM. That is, RFCM does not generate cluster centers of object data during iteration because RFCM processes relational data. However, if object data satisfying $r_{jk} = \|\mathbf{x}_j - \mathbf{x}_k\|_A^2$ are known, the reformulation theorem guarantees that non-iterative computation of the cluster centers with (2.7b) based on the terminal partition found by RFCM will be the same as the cluster centers found directly with FCM, provided both algorithms are initialized at the same partition of X. The reformulation theorem can also be used to design relational versions of HCM and PCM.

Example 3.10

Table 3.10 shows terminal membership values for fuzzy cluster $\mathbf{U}_{(1)}$ ($u_2(o_k) = 1 - u_1(o_k)\ \forall k$) in partitions generated by the FNM, AP and RFCM (m=2) relational clustering models.

Since R_{11} in Example 3.9 is derived from X_{11} with Euclidean distance, we expect RFCM to produce reasonably good results for this data. All three algorithms were initialized with the 2- partition U_0 at (3.21), and all were terminated (in less than 18 iterations) when the absolute difference between successive values of their objective function was less than $\varepsilon = 0.0001$.

Table 3.10 shows that all three models behave similarly on this data set. They all produce membership functions that are symmetric with respect to the y axis, and they all assign the membership value 0.5 to object 6 in both fuzzy clusters. The RFCM result is the *crispest* of the three outputs, and FNM is very slightly the fuzziest, even though all three algorithms use squares for membership exponents in this example.

Table 3.10 Terminal cluster 1 memberships for R_{11}

	FNM	AP	RFCM
Datum	$\mathbf{U}^T_{(1)}$	$\mathbf{U}^T_{(1)}$	$\mathbf{U}^T_{(1)}$
$\mathbf{x}_1$	0.91	0.92	0.95
$\mathbf{x}_2$	0.88	0.90	0.94
$\mathbf{x}_3$	0.93	0.95	1.00
$\mathbf{x}_4$	0.88	0.90	0.94
$\mathbf{x}_5$	0.82	0.86	0.91
$\mathbf{x}_6$	**0.50**	**0.50**	**0.50**
$\mathbf{x}_7$	0.18	0.14	0.09
$\mathbf{x}_8$	0.12	0.10	0.06
$\mathbf{x}_9$	0.07	0.05	0.00
$\mathbf{x}_{10}$	0.12	0.10	0.06
$\mathbf{x}_{11}$	0.09	0.08	0.05

AP and FNM require one less assumption on R than RFCM. Thus, the AP and FNM models have a wider reach in applications than RFCM. What happens when RFCM is applied to *arbitrary* dissimilarity data that does not satisfy (3.25b)? Hathaway and Bezdek (1994b) provide a partial solution to this problem through an extension of RFCM that is discussed next.

D. The non-Euclidean RFCM (NERFCM) model

RFCM can be used to cluster a set of n objects described by pair-wise dissimilarity values in R if (and only if) there exist n object data points in some p-space whose squared Euclidean distances match values in R. More formally, a relation R is *Euclidean* if there exists a data set $X=\{x_1,\ldots,x_n\}$ in $\Re^{n-1}$ such that $R=\left[r_{jk}=\|\mathbf{x}_j-\mathbf{x}_k\|^2\right]$; otherwise, R is said to be non-Euclidean. Any object data set X corresponding to a Euclidean relation R is called a *realization* of R. If there exists a realization of R in p-space, p < n-1, we can get a realization in n-1 space by adding n-p-1 components with constant values to each point in the p dimensional data.

The duality theory of the relational (RFCM) and object (OFCM) data versions of the fuzzy c-means models says that RFCM applied to R corresponds to OFCM applied to object data X if and only if there exists a set of n points in $\Re^{n-1}$ whose squared Euclidean distances match the given dissimilarity data R. Given an arbitrary relation there is no reason to believe that the duality condition will hold. And if it does not, RFCM may fail. We will see later an example of this type where the relational data are generated as squared, pair-wise (object-data) distances in 1-norm.

NERF c-means assumes that *dissimilarity* relation R is irreflexive, positive and symmetric :

$$r_{jj} = 0 \quad , j = 1, .., n \qquad ; \qquad (3.26a)$$

$$r_{jk} \geq 0 \quad , 1 \leq j, k \leq n \qquad ; \text{ and} \qquad (3.26b)$$

$$r_{jk} = r_{kj} \quad , 1 \leq j, k \leq n \qquad . \qquad (3.26c)$$

Given a non-Euclidean R that satisfies (3.26), the basic idea in NERF c-means is to convert R into a Euclidean relation R_β using a β-spread transformation, and then apply RFCM-AO to R_β. This is very similar in spirit to clustering in the transitive closure of a relation after finding it, as we did in Example 3.2. The transformation is:

$$R_\beta = R + \beta(\mathbf{1}_{n\times n} - I_n) \qquad , \qquad (3.27)$$

where β is a suitably chosen real scalar, I_n is the $n\times n$ identity matrix and $\mathbf{1}_{n\times n}$ is the $n\times n$ matrix with 1's at every address. Choosing $\beta = 0$ in (3.27) reduces R_β to the original relation, $R = R_0$. The operation in (3.27) is called β-spreading since the addition of $\beta > 0$ to the off-diagonal elements of any Euclidean matrix R has the effect of spreading out the corresponding realization. We discuss the case $\beta < 0$ after Example 3.11.

Example 3.11 Let R be the Euclidean relation

$$R = R_0 = \begin{bmatrix} 0 & 81 & 100 \\ 81 & 0 & 1 \\ 100 & 1 & 0 \end{bmatrix} \qquad . \qquad (3.28)$$

One realization of $R = R_0$ is given by the three points

$$\mathbf{x}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \ \mathbf{x}_2 = \begin{pmatrix} 10 \\ 1 \end{pmatrix} \text{ and } \mathbf{x}_3 = \begin{pmatrix} 11 \\ 1 \end{pmatrix} \qquad , \qquad (3.29)$$

which are plotted along the horizontal line segment in Figure 3.7 indicated by $\beta = 0$.

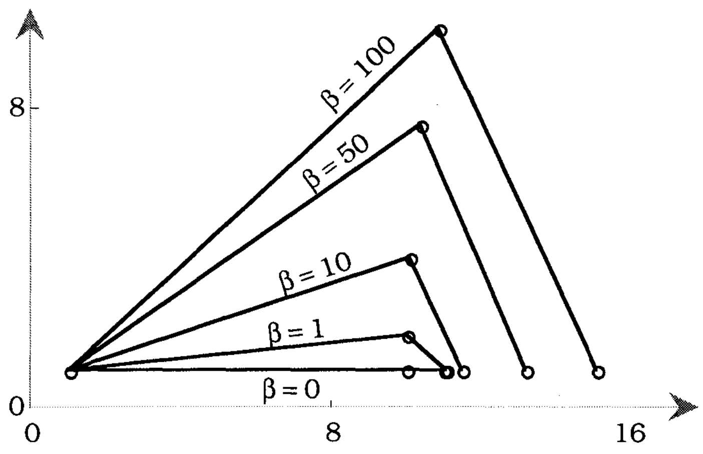


Figure 3.7 Some 3-point realizations for R_β using R from (3.28)

Figure 3.7 also exhibits realizations for R_1, R_{10}, R_{50}, and R_{100}. This demonstrates geometric spreading of the realization as β increases. Realizations are not generally unique. However, the ones shown in Figure 3.7 are the only ones satisfying these three conditions: the left point is $\mathbf{x}_1$; (2) the second coordinate of the right point = 1; and (3) the second coordinate of the middle point is at least 1. Visually, the natural crisp clustering of these three points for small values of β is c = 2 groups, $\{\mathbf{x}_1\} \cup \{\mathbf{x}_2, \mathbf{x}_3\}$; as β increases, this becomes less and less obvious.

To illustrate the effect of β on clustering, R_β was clustered with RFCM-AO for various values of β with m = c = 2. Initialization was at the (visually unnatural) crisp clusters $\{\mathbf{x}_1, \mathbf{x}_2\} \cup \{\mathbf{x}_3\}$. Results for every value of β shown in Figure 3.7 and several others as well are listed in Table 3.11.

The values shown in Table 3.11 are the terminal memberships of the three points in cluster 1 at each value of β. Cluster 2 memberships can be obtained by $u_{2k} = 1 - u_{1k}$, k = 1, 2, 3. First observe that RFCM-AO works for β = -0.25 and -0.50, even though $R_{-0.25}$ and $R_{-0.50}$ are non-Euclidean. This, as well as the failure of RFCM-AO at β = -1 will be explained below.

Table 3.11 Terminal RFCM -AO membership values in cluster 1

β	Iter.	$u_1(\mathbf{x}_1)$	$u_1(\mathbf{x}_2)$	$u_1(\mathbf{x}_3)$
-1.00	0	Fails	Fails	Fails
-0.50	4	1.000	0.002	0.001
-0.25	4	1.000	0.002	0.002
0	4	1.000	0.003	0.003
1	4	1.000	0.006	0.005
10	6	1.000	0.027	0.024
50	11	1.000	0.094	0.078
100	16	0.999	0.134	0.112
500	46	0.995	0.207	0.177
1,000	78	0.994	0.221	0.192
5,000	336	0.992	0.230	0.210
10,000	131	0.818	0.730	0.010

For $\beta \geq -0.5$, terminal partitions become fuzzier and the work required (iterations to termination) increases as β increases to 5000. In all cases except $\beta = 10{,}000$ the final partition reflects a strong association of the center point $\mathbf{x}_2$ with the right point $\mathbf{x}_3$; the hardened version of U in these cases is $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$. For $\beta = 10{,}000$ the spread is finally great enough that RFCM-AO stalls near the initial partition $\{\mathbf{x}_1, \mathbf{x}_2\} \cup \{\mathbf{x}_3\}$, as indicated by the memberships in the last row of Table 3.11.

R_β is non-Euclidean for any $\beta < 0$. To understand this recall the well-known result (Mardia et al., 1979) that

$$\text{R is Euclidean} \Leftrightarrow \mathbf{z}^T R \mathbf{z} \leq 0 \quad \forall \quad \mathbf{z} \in \Re^n \text{ with } \sum_{j=1}^{n} z_j = 0. \tag{3.30}$$

With $\mathbf{1}_{n \times n}$ and I_n as in (3.27),

$$P = I_n - \left(\frac{1}{n}\right)\mathbf{1}_{n \times n} \tag{3.31}$$

is the projector onto the n-1 dimensional subspace orthogonal to the n-vector $\mathbf{1}_{n \times 1}$. A condition equivalent to (3.30) is

$$\text{R is Euclidean} \Leftrightarrow \text{PRP is negative semi-definite.} \tag{3.32}$$

In other words, R is non-Euclidean, and RFCM-AO may fail, whenever the matrix PRP has a positive eigenvalue. It is also well-known in the literature on multidimensional scaling that for a Euclidean matrix R the *number* of strictly negative eigenvalues of PRP equals the (minimum) dimension s required for a realization of R. Example 3.12 illustrates this.

Example 3.12 Consider the dissimilarity matrix R of equation (3.28) in Example 3.11 where n = 3. To determine whether R is Euclidean or not, we calculate the eigenvalues of

$$\mathrm{PRP} = \begin{bmatrix} 2/3 & -1/3 & -1/3 \\ -1/3 & 2/3 & -1/3 \\ -1/3 & -1/3 & 2/3 \end{bmatrix} \times \begin{bmatrix} 0 & 81 & 100 \\ 81 & 0 & 1 \\ 100 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 2/3 & -1/3 & -1/3 \\ -1/3 & 2/3 & -1/3 \\ -1/3 & -1/3 & 2/3 \end{bmatrix},$$

which are {0, 0, -364/3}. Since all eigenvalues are non-positive, PRP is negative semi-definite, and R is Euclidean. Now the minimum dimension required for a Euclidean realization of R is 1, since only one of the eigenvalues is negative. It is easy to verify that the real numbers $\{y_1, y_2, y_3\} = \{0, 9, 10\} \subset \Re$ are a one dimensional realization of R. (Another would be {25, 34, 35}.) We can always find a higher dimensional realization by adding constant components to a lower dimensional one. For example, a 3-dimensional Euclidean realization of this R is $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\} = \left\{ \begin{pmatrix} 0 \\ \gamma \\ \chi \end{pmatrix}, \begin{pmatrix} 9 \\ \gamma \\ \chi \end{pmatrix}, \begin{pmatrix} 10 \\ \gamma \\ \chi \end{pmatrix} \right\}$, $\gamma, \chi \in \Re$. If the eigenvalues of PRP had been, for example, {0, -4, 1}, then R would not have been Euclidean, and no Euclidean realization of it would exist in any dimension.

A result that gives insight about the construction of a Euclidean relation using the β-spread transformation follows.

Theorem HB (Hathaway and Bezdek, 1994b)

Let $R \in \Re^{n \times n}$ satisfy (3.26), and let R_β and P be the matrices in (3.27) and (3.31) respectively. Then:

(a) $PR_\beta P = P(R - \beta I_n)P$.

(b) $\mathbf{1}_{n \times 1}$ is an eigenvector, with corresponding eigenvalue 0 for both PRP and $PR_\beta P$.

(c) **w** is an eigenvector of PRP if and only if it is an eigenvector of $PR_\beta P$.

(d) if $\mathbf{w}$ is an eigenvector of PRP and $PR_\beta P$ other than a multiple of $\mathbf{1}_{n\times 1}$, then the corresponding eigenvalues λ and λ_β of PRP and $PR_\beta P$ satisfy $\lambda - \beta = \lambda_\beta$.

This shows that adding β to the off diagonal elements of a matrix R satisfying (3.26) effects a shift of - β to all the eigenvalues of $PR_\beta P$, except the zero eigenvalue corresponding to the eigenvector $\mathbf{1}_{n\times 1}$, which is left unchanged. Now, let a given non-Euclidean R satisfy (3.26) and let λ be the largest eigenvalue of PRP. We must have $\lambda > 0$ by (3.32) since R is non-Euclidean, so it follows by (3.32) and Theorem HB that R_β will be Euclidean for all choices of $\beta \ge \lambda$.

Figure 3.8 depicts the general case for any relation R that satisfies (3.26) and the additional constraint that $R \ne \tau(\mathbf{1}_{n\times n} - I_n)$ for any τ in $\Re$. Then there is some value $\hat{\beta}$ for which $R_{\hat{\beta}}$ is Euclidean and realizable by a set $\{\mathbf{x}_1, \dots \mathbf{x}_n\} \subset \Re^s$ for some s, $1 \le s \le n-2$. Moreover, R_β is non-Euclidean for $\beta < \hat{\beta}$, and Euclidean for $\beta > \hat{\beta}$, but realizable only for $s \ge n-1$.

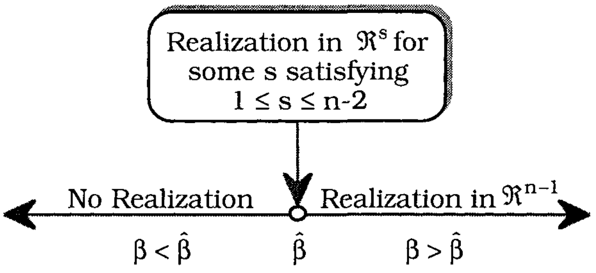

Realization in $\Re^s$ for some s satisfying $1 \le s \le n-2$
No Realization
Realization in $\Re^{n-1}$
$\beta < \hat{\beta}$
$\hat{\beta}$
$\beta > \hat{\beta}$

Figure 3.8 Minimum realization dimension for R_β

In Example 3.11, the cutoff value is $\hat{\beta} = 0$, where $R = R_0$ is realizable in $\Re$. For any choice of $\beta > 0$, the realization requires n-1 = 2 dimensions, and for any $\beta < 0$, no realization exists and R_β is non-Euclidean. Observe that rows 2 and 3 of Table 3.11 correspond to cases when RFCM-AO worked even though R_β was non-Euclidean.

Table 3.12 The NERFCM-AO algorithms

Store	Relation matrix $R = \left[r_{jk}\right]_{n\times n}$ satisfying, $\forall\, i, j$, $r_{ij} \geq 0$, $r_{ii} = 0$, $r_{ij} = r_{ji}$ *and* $R \neq \tau(\mathbf{1}_{n\times n} - I_n)$, $\tau \in \Re$
Pick	☛ number of clusters : $1 < c < n$ ☛ maximum number of iterations : T ☛ weighting exponent : $1 < m < \infty$ ☛ termination threshold : $0 < \varepsilon$
Guess	Initial partition $U_0 \in M_{fcn}$

Iterate

```
$\beta$=0 ; $t \leftarrow 1$
While ( $t \leq T$ )
  For i=1 to c
    $\mathbf{v}_{i,t} = \left((u_{i1,t-1})^m, \ldots\ldots, (u_{in,t-1})^m\right)^T \Big/ \sum_{k=1}^{n} (u_{ik,t-1})^m$        (3.33)
  Next i
  For i=1 to c
    For k= 1 to n
    $d_{ik} = (R_\beta \mathbf{v}_{i,t})_k - ((\mathbf{v}_{i,t})^T R_\beta \mathbf{v}_{i,t}) / 2$        (3.34)
  Next k
  Next i
  If $d_{ik} < 0$ for any i and k, then
  $\Delta\beta = \underbrace{\max}_{i,k}\left\{-2 * d_{ik} / (\|\mathbf{v}_{i,t} - \mathbf{e}_k\|^2)\right\}$        (3.35a)
      For i=1 to c
        For k=1 to n
          $d_{ik} = d_{ik} + (\Delta\beta / 2)\|\mathbf{v}_{i,t} - \mathbf{e}_k\|^2$        (3.35b)
        Next k
      Next i
      $\beta = \beta + \Delta\beta$        (3.35c)
  For k = 1 to n
    If $d_{ik} > 0 \ \forall\ i$
    $u_{ik,t} = 1 \Big/ \left[\sum_{j=1}^{c} (d_{ik} / d_{jk})^{1/(m-1)}\right]$
  (3.36)
  Else $u_{ik,t} = 0$ if $d_{ik} > 0$ and $u_{ik,t} \geq 0$ with $\sum_{i=1}^{c} u_{ik,t} = 1$
  Next k
 If $\left|K_m(U_t) - K_m(U_{t-1})\right| < \varepsilon$ Then Exit While
 $t \leftarrow t + 1$
End while
$U \leftarrow U_t$
```

A straightforward way of using (3.27) with RFCM would be to simply compute (numerically) the largest non-negative eigenvalue λ (= $\hat{\beta}$ in Figure 3.8) of PRP, and then cluster the Euclidean matrix $R_{\hat{\beta}=\lambda}$ with RFCM-AO. Instead of doing unnecessarily costly eigenvalue computations, Hathaway and Bezdek (1994b) suggested an alternate approach that dynamically estimates in a computationally efficient way the β-spread needed to continue RFCM-AO. This approach is efficient because it depends primarily on by-products of the original RFCM iteration. Table 3.12 lists the NERFCM-AO algorithm

NERFCM-AO and RFCM-AO are identical except for the modifications in (3.35) that are active whenever some negative value of d_{ik} is encountered. The duality theory asserts that d_{ik} values correspond to certain squared Euclidean distances if an object-data realization of R_β exists. It follows that a negative value of d_{ik} signals the non-existence of a realization of R_β, which indicates that the current value of β should be incremented by some $\Delta\beta > 0$ so that the basic (RFCM-AO) iteration can be continued using the new shifted value $\beta + \Delta\beta$. Hathaway and Bezdek in (1994b) showed that the increment $\Delta\beta$ in (3.35a) is reasonable in the sense that it provides a meaningful lower bound of the minimum increment needed to make the new R_β Euclidean. They also proved that NERFCM-AO was correct in that the updated d_{ik} values in (3.35b) are non-negative and correspond to the d_{ik} values for the newly updated β in (3.35c).

To summarize, modification of the original RFCM-AO algorithm using (3.35) calculates a reasonable (under)estimate of the minimum shift required to transform the current R_β into a Euclidean matrix, and then implements this shift by updating the current d_{ik} values and value of β. The quantities used to determine the shift are the original d_{ik} values and the values $\{\|\mathbf{v}_i - \mathbf{e}_k\|^2\}$. Since these are exactly the quantities needed to perform the updating of the d_{ik}, there is no wasted computation done in determining the new increment to R_β. Moreover, whenever an increment in the shift is not needed, which is in the large majority of iterations, the work requirements for that particular iteration of NERFCM-AO are no greater than that for a RFCM-AO iteration, except for the additional negativity checks on d_{ik}, which are negligible in cost.

Example 3.13 Table 3.13 lists the coordinates of the data set $\hat{X}_{11}$ produced by truncating the decimal parts of X_{11} in Table 3.7.

Table 3.13 Coordinates of $\hat{X}_{11}$

Datum	x	y
$\hat{\mathbf{x}}_1$	-5	0
$\hat{\mathbf{x}}_2$	-3	2
$\hat{\mathbf{x}}_3$	-3	0
$\hat{\mathbf{x}}_4$	-3	-2
$\hat{\mathbf{x}}_5$	-2	0
$\hat{\mathbf{x}}_6$	0	0
$\hat{\mathbf{x}}_7$	2	0
$\hat{\mathbf{x}}_8$	3	2
$\hat{\mathbf{x}}_9$	3	0
$\hat{\mathbf{x}}_{10}$	3	-2
$\hat{\mathbf{x}}_{11}$	5	0

Figure 3.9 is a scatterplot of $\hat{X}_{11}$, which shows that $\hat{X}_{11}$ has the same basic structure as X_{11} (Figure 3.6). Visually, there are again clusters to the left and right of the bridge point $\mathbf{x}_6 = (0,0)^T$.

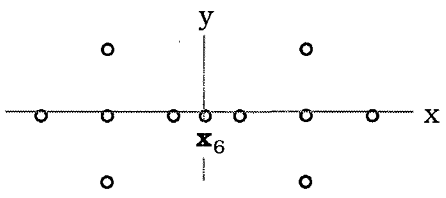


Figure 3.9 Data set $\hat{\mathbf{X}}_{11}$

All runs of NERFCM-AO reported here used c = m = 2, a stopping criterion $\varepsilon = 0.0001$, and the initialization shown at (3.21). Three transformations of $\hat{X}_{11}$ were made, resulting in three dissimilarity relation matrices $R_{\|*\|}$. Specifically, the entries of $R_{\|*\|}$ were computed using: (i) squared Euclidean distances $R_{\|*\|_2^2}$; (ii) squared 1-norm distances $R_{\|*\|_1^2}$; and (iii) squared 1-norm distances with an off-diagonal shift of 48.0, $R_{\|*\|_1^2+48}$. The third choice is motivated by the eigenvalues of $PR_{\|*\|}P$, all three sets of which are displayed in Table 3.14.

Table 3.14 Eigenvalues of $PR_{\|*\|}P$ for three cases

$R_{\|*\|_2^2}$	$R_{\|*\|_1^2}$	$R_{\|*\|_1^2+48}$
0.00	48.00	0.00
0.00	22.11	0.00
0.00	4.94	-25.89
0.00	0.00	-43.06
0.00	0.00	-48.00
0.00	0.00	-48.00
0.00	0.00	-48.00
0.00	-3.32	-51.32
0.00	-47.12	-95.12
-32.00	-80.00	-128.00
-212.00	-278.79	-326.79

The pair of negative eigenvalues for $PR_{\|*\|_2^2}P$ shown in column 1 of Table 3.14 imply that $R_{\|*\|_2^2}$ has a two-dimensional object-data realization. This is no surprise, since $R_{\|*\|_2^2}$ was derived using squared Euclidean distances between two-dimensional vectors. Table 3.14 also suggests that using the 1-norm gives a non-Euclidean R (also no surprise) as indicated by three positive eigenvalues, the largest of which is 48. Apparently $R_{\|*\|_1^2}$ can be made Euclidean using a β-spread with $\beta \geq 48$. Using $\beta = 48$ in (3.27) with $R = R_{\|*\|_1^2}$ renders R_β Euclidean, and this transformed matrix has a nine-dimensional object-data realization. This is seen in the third column of Table 3.14; $R_{\|*\|_1^2+48}$ has 9 negative and no positive eigenvalues.

Terminal membership values in cluster 1 obtained by applying NERFCM-AO to the three relational data sets generated by transforming $\hat{X}_{11}$ are listed in Table 3.15.

Membership values for $R_{\|*\|_1^2}$ and $R_{\|*\|_2^2}$ are relatively crisp and similar to each other, the maximum difference being 0.05. Membership values for clusters in $R_{\|*\|_1^2+48}$ (corresponding to Euclidean distances for some object data set in $\Re^9$) are much fuzzier, as expected. The shift needed for the $R_{\|*\|_1^2}$ data was only $\beta = 3.56$, much less than the $\beta = 48$ required to have actual Euclidean dissimilarities. Note that bridge point $\hat{\mathbf{x}}_6$ receives equal membership in both clusters in all three cases; this is expected and desirable.

Table 3.15 Terminal NERFCM-AO memberships in cluster 1

Data Set	$R_{\|*\|_2^2}$	$R_{\|*\|_1^2}$	$R_{\|*\|_1^2+48}$
Iter.	11	11	9
Final β	0	3.56	0
$\hat{\mathbf{x}}_1$	0.93	0.90	0.75
$\hat{\mathbf{x}}_2$	0.91	0.89	0.73
$\hat{\mathbf{x}}_3$	1.00	1.00	0.77
$\hat{\mathbf{x}}_4$	0.91	0.89	0.73
$\hat{\mathbf{x}}_5$	0.81	0.76	0.62
$\hat{\mathbf{x}}_6$	**0.50**	**0.50**	**0.50**
$\hat{\mathbf{x}}_7$	0.19	0.24	0.38
$\hat{\mathbf{x}}_8$	0.09	0.11	0.27
$\hat{\mathbf{x}}_9$	0.00	0.00	0.23
$\hat{\mathbf{x}}_{10}$	0.09	0.11	0.27
$\hat{\mathbf{x}}_{11}$	0.07	0.10	0.25

NERFCM shares all the good properties of RFCM. If negative d_{ik} values are not encountered, then NERFCM *is* RFCM; and when they are encountered, a simple modification adjusts the "spread" of (implicit realizations of) the data just enough to keep the iteration sequence $\{U_t\}$ in M_{fcn}.

3.6 Cluster validity for relational models

Methods for validation of clusters found from relational data are scarce. Validity for partitions based on object data X was discussed in Section 2.5. When $X \subset \Re^p$ is transformed into relational data R using the techniques in Section 3.2, subsequent processing with a relational algorithm leads to a crisp or fuzzy c-partition of X. In this case many of the indices in Section 2.5 can be used for validation of U, since X is available.

When the data are relational to begin with, validation methods that explicitly require numerical data (for example, all direct indices) are not applicable to the question of cluster validity. When a relational algorithm produces *fuzzy* partitions of O, indirect indices such as the partition coefficient and partition entropy can be used for validation.

Jain and Dubes (1988) cover several statistical hypothesis tests for validation of entire hierarchies (dendrograms) of clusters that are obtained from methods such as the linkage algorithms and Zadeh's decomposition of the transitive closure. Another validation strategy for linkage type algorithms is associated with the *size of jumps* taken by the set distance $\hat{\delta}$ that controls merger or splitting (Hartigan, 1975). For example, when the procedure begins with $\hat{\delta}=0$ at c = n and terminates at $\hat{\delta}=$ maximum at c = 1, the usual strategy is to look for the largest jump in $\Delta\hat{\delta} = \hat{\delta}(c-1) - \hat{\delta}(c)$. This is taken as an indicator that c is the most natural choice for the best number of clusters, on the presumption that the SAHN method works hardest to merge clusters that cause the biggest jump. In Figure 3.4, for example, successive jumps in the single linkage merger distances are 0.50, 0.50, 0.85 and 0.21. The largest jump, (0.85 from c = 3 to c =2) identifies the crisp partition $X = \{1,2\} \cup \{3,4\} \cup \{5,6,7,8,9\}, c=3$ clusters at $\hat{\delta}_{min}=1.00$, as the most natural ones. The configuration of the data in Figure 3.2 seems to confirm this visually, although a case can be made that c = 2 is just as natural.

The sequence of jumps for the complete linkage solution shown in Figure 3.4 is 0.50, 0.50, 0.75, 0.45 and 1.9, indicating that the clusters associated with c = 2 are the best choice, which is of course different than the solution offered by single linkage. One problem with this method is that the biggest jump can be severely influenced by the presence of a few outliers.

Zadeh's algorithm decomposes a fuzzy relation into crisp partitions with different values of c at values of α corresponding to α-cuts of R^{∞}. The scalars $\{\alpha\}$ are sometimes regarded as a rough indication of the validity of each hard clustering of O. In Example 3.7, we might assert that each object belongs by itself (c = 5) with confidence $\alpha = 1$. But this is always true, and leads to the wrong conclusion - i.e., that the best clustering is one with all singletons. Continuing this reasoning, x_2 and x_5 would belong together with confidence $\alpha = 0.9$, and so on. Since α is just the edge weight of the strongest adjacency link between each pair of nodes in R_α, the word confidence as used here has no statistical connotation, and this use of the values of α is pretty misleading, for they do not portray "better and better" partitions as α gets closer and closer to 1. Indeed, you might argue that the confidence in a partition by R_α should be *inversely* proportional to α, and we would not object. In view of Theorem M, we know that single linkage generates the same clusters as Theorem Z for fuzzy similarity relations. Consequently, the largest jump method can also be used for validation of clusters such as those associated with the dendrogram in Figure 3.5. In this figure, the successive jumps in α are : 0.10, 0.10, and 0.40, indicating a strong preference for c = 3 clusters.

When R is clustered with convex decomposition, since $\sum_{k=1}^{p} c_k = 1$, c_k indicates the "percentage" of R_k needed for convex factorization of R. In terms of cluster validity then, c_k can be loosely interpreted as an indicator of the relative merit of the associated c-partition induced on O by R_k. In Example 3.8, this leads to interpreting the 3-partition $U_{0.4}$ as the most valid choice, and the two partitions with $c_k = 0.3$ at different values of c are regarded as less but equally valid.

3.7 Comments and bibliography

The SAHN, transitive closure and convex decomposition techniques produce hard partitions from certain crisp or fuzzy relations. FNM, AP, RFCM and NERFCM all produce fuzzy partitions from particular classes of fuzzy relations by minimizing a relational objective function with alternating optimization. Kaufman and Rouseeuw (1990) discuss a method called FANNY that is closely related to RFCM. Sen and Dave (1998) show that using the method of LaGrange multipliers with the RFCM objective function in (3.25a) leads to the RFCM algorithm without making the assumption in (3.25b), but the derivation does not ensure that all of the memberships will be non-negative, because LaGrange multipliers only enforce equality constraints. Following Kaufmann and Rousseeuw's derivation of FANNY, these authors have very recently obtained an even stronger result using Kuhn-Tucker theory that proves that the memberships will satisfy the required non-negativity condition. This result will be published in a forthcoming paper. The equations obtained in their Kuhn-Tucker approach are slightly different than the ones given in our description of the RFCM algorithm.

Fuzzy partitions enable the user to quantitatively distinguish between objects which are strongly associated with particular clusters from those that have only a marginal (borderline) association with several classes. The AP algorithm assumes the existence of prototypical objects which should be good representatives of different clusters (but does not give a method for finding them, although the object with maximum typicality would be an obvious candidate), and has the least restrictions on relations that it can process. Runkler and Bezdek (1998b) give a relational version of the alternating cluster estimation method called RACE that explicitly finds prototypical object indices (and hence, prototypical objects too), even though the data are known only in relational form. RACE finds a fuzzy partition of the objects too, and, like its object data counterpart (ACE, Runkler and Bezdek, 1998a), is not objective function driven.

Using NERFCM-AO is in one sense like extracting clusters from a crisp relation by first computing its $\vee - \wedge$ transitive closure as in Example 3.2. Both methods group the n objects underlying the original relation by clustering in a transformed relational data matrix. For NERFCM-AO it is difficult to estimate of how closely clusters in R_β might resemble those extracted from R by some other method. Intuitively, if the required spread is not too large, structure inherent in R should be mirrored by that in R_β. Hathaway and Bezdek (1994c) discuss a crisp version of this model called, naturally, *non-Euclidean relational hard c-means* (NERHCM).

The usefulness of relational clustering methods is limited by several things. First is the matter of their computational complexity. $O(n^2)$ run times are fine if n = 150, as in the Iris data. But in large relational databases, n may be on the order of 10^6, and CPU time becomes an important factor. On the other hand, some information retrieval problems, for example, are cast naturally in the form of relational clustering, and there may be little choice but to use one of these schemes. Another limitation with the methods in this chapter is that they are explicitly limited to square relations, while a real application may have rectangular relational data.

Delgado et al. (1995) propose the use of hierarchical clustering algorithms such as single linkage for cluster validity. In their view the failure of cluster validity functionals such as the ones discussed in Chapter 2 can be ameliorated by pre-clustering data with a SAHN algorithm, and using the results to limit the range of c and provide good initializations for the "real" clustering to follow (presumably by a non-crisp clustering algorithm). Having said this, they try to address the topic of Section 3.6 - how to validate relational clusters - by proposing several validity measures for their SAHN algorithms. In other words, they end up with the same problem in one domain they are trying to avoid in another domain! They give several numerical examples of their methods, including one with the Iris data.

Sato et al. (1997) propose three relational clustering models they call *additive clustering models*. In the language of our book these correspond to crisp (ordered additive), fuzzy (simple additive) and possibilistic (overlapping additive) clustering schemes. All three methods are regarded as relatives of the crisp relational model of Shephard and Arabie (1979), extended using fuzzy data, fuzzy dissimilarity and multicriteria clustering. The basic objective function for Sato et al.'s three additive models is

$$\underset{\substack{U \in M_{fcn} \\ \alpha \in \Re^+}}{\underbrace{\min}} \left\{ K_{ACM}(U,\alpha) = \frac{\sum\limits_{\substack{j=1 \\ j \neq k}}^{n} \sum\limits_{k=1}^{n} \left(r_{kj} - \alpha \sum\limits_{i=1}^{c} u_{ik} u_{ij} \right)^2}{\sum\limits_{\substack{j=1 \\ j \neq k}}^{n} \sum\limits_{k=1}^{n} \left(r_{kj} - \bar{r} \right)^2} \right\} \text{, where} \quad (3.37a)$$

$$\bar{r} = \frac{\sum\limits_{\substack{j=1 \\ j \neq k}}^{n} \sum\limits_{k=1}^{n} r_{kj}}{n(n-1)} \quad . \quad (3.37b)$$

Sato et al. build three models (one each for $U \in M_{hcn}, M_{fcn}, M_{pcn}$) based on variations of (3.37) that are used with ratio, interval and ordinal relational data. The relational data matrix in their models is not restricted to inner product norm distance relations or even symmetric relations. Sato et al. also discuss a "generalized fuzzy clustering model" for relational data that uses an aggregation operator in the objective function

$$\underset{U \in M_{fcn}}{\underbrace{\min}} \left\{ K_T(U) = \sum\limits_{\substack{j=1 \\ j \neq k}}^{n} \sum\limits_{k=1}^{n} \left(r_{kj} - \sum\limits_{i=1}^{c} T(u_{ik}, u_{ij}) \right)^2 \right\} \quad , \quad (3.38)$$

where T is any T-norm. Three choices are discussed and exemplified in Sato et al. : the minimum (T_3), product (T_2) and Hamacher T-norms (Klir and Yuan, 1995). Also given are methods for optimizing (3.38) in each of the three cases. They give several examples of clustering with each of these models using small relational data sets, but like their discussion of TFCM (Section 2.6), no numerical comparisons to other relational clustering models are offered, so we are again at a loss to make any assessment of the utility of these models. However, the work presented in this little book considerably extends the body of fuzzy clustering algorithms available for relational data, so if this is the type of data you have, by all means consider trying one or more of these approaches.

4 Classifier Design

4.1 Classifier design for object data

In Section 1.1 we defined a *classifier* as any function $\mathbf{D}:\Re^p \mapsto N_{pc}$. The value $\mathbf{y} = \mathbf{D}(\mathbf{z})$ is the label vector for $\mathbf{z}$ in $\Re^p$. $\mathbf{D}$ is a *crisp classifier* if $\mathbf{D}[\Re^p] = N_{hc}$; otherwise, the classifier is fuzzy, possibilistic or probabilistic, which for convenience we lump together as *soft* classifiers. This chapter describes some of the most basic (and often most useful) classifier designs, along with some fuzzy generalizations and relatives.

Soft classifier functions $\mathbf{D}:\Re^p \mapsto N_{pc}$ are consistent with the *principle of least commitment* (Marr, 1982), which states that algorithms should avoid making crisp decisions as long as possible, since it is very difficult (if not impossible) to recover from a wrong crisp classification. This is particularly true in complex systems such as an automatic target recognition system, or a computer aided medical diagnostician that uses image data, because there are several stages where decisions are made, each affecting those that follow. For example, pixels in a raw image need to be classified as noise points for preprocessing, objects need to be segmented from the preprocessed images, features must be extracted and the objects classified, and the entire "scene" needs to be labeled. While we use mostly simple data sets to illustrate some of the algorithms in this chapter, keep in mind complex scenarios such as the ones just described to appreciate the potential benefits of fuzzy recognition approaches.

Many classifiers assign non-crisp labels to their arguments. When this happens, we often use the hardening function $\mathbf{H}:N_{pc} \mapsto N_{hc}$ defined at (1.15) to convert non-crisp labels into crisp ones; for c classes, $\mathbf{H} \circ \mathbf{D}(\mathbf{y}) = \mathbf{H}(\mathbf{D}(\mathbf{y})) \in \{\mathbf{e}_1, \ldots, \mathbf{e}_c\}$.

Designing a classifier simply means "finding a good $\mathbf{D}$ ". When this is done with labeled training data, the process is called *supervised learning.* We pointed out in Chapter 2 that it is the labels of the data that supervise; we will meet other forms of supervision later in this chapter, and they are also appropriately called supervised learning.

$\mathbf{D}$ may be specified functionally (e.g., the Bayes classifier), or as a computer program (e.g. computational neural networks or fuzzy input-output systems). Both types of classifiers have parameters. When $\mathbf{D}$ is a function, it has constants that need to be "learned" during training. When $\mathbf{D}$ is a computer program, the model it implements has both control parameters and constants that must also be acquired by "learning". In either case the word *learning* means finding good *parameters* for **D** - and that's all it means.

In supervised classifier design X is usually crisply partitioned into a *training* (or design) *set* X_{tr} with label matrix U_{tr} and cardinality $|X_{tr}| = n_{tr}$; and a *test set* $X_{te} = (X - X_{tr})$ with label matrix U_{te} and cardinality $|X_{te}| = n_{te}$. Columns of U_{tr} and U_{te} are label vectors in N_{pc}. Testing a classifier designed with X_{tr} means estimating its *error rate* (or probability of misclassification). The standard method for doing this is to submit X_{te} to **D** and count mistakes (U_{te} must have crisp labels to do this). This yields the *apparent* error rate $E_{\mathbf{D}}(X_{te}|X_{tr})$. Apparent error rates are conveniently tabulated using the $c \times c$ *confusion matrix* $C = [c_{ij}]$ = [# labeled class j | but were really class i]. (Some writers call C^T the confusion matrix.) More formally, the *apparent error rate* of **D** when trained with X_{tr} and tested with X_{te} is

$$E_{\mathbf{D}}(X_{te}|X_{tr}) = \left(\frac{\text{\# wrong}}{n_{te}}\right) = \left(1 - \left(\frac{\text{\# right}}{n_{te}}\right)\right) = \left(1 - \left(\frac{\text{tr}(C)}{n_{te}}\right)\right). \tag{4.1}$$

Equation (4.1) gives, as a fraction in [0, 1], the number of errors committed on test. This number is a function not only of **D**, but of two specific data sets, and each time any of the three parameters changes, $E_{\mathbf{D}}$ will in all likelihood change too.

Other common terms for the error rate $E_{\mathbf{D}}(X_{te}|X_{tr})$ include *test error* and *generalization error.* Our notation indicates that **D** was trained with X_{tr}, and tested with X_{te}. $E_{\mathbf{D}}$ is often the performance index by which **D** is judged, because it measures the extent to which **D** generalizes *to the test data.* Some authors call $E_{\mathbf{D}}(X_{te}|X_{tr})$ the "true" error rate of **D**, but to us, this term refers to a quantity that is not computable with estimates made using finite sets of data.

$E_{\mathbf{D}}(X|X)$ is the *resubstitution* error rate (some authors use this term synonomously with apparent error rate, but we prefer to have separate terms for these two estimates). Other common terms for $E_{\mathbf{D}}(X|X)$ include *training error* and *recall error* rate. Resubstitution uses the same data for training and testing, so it usually produces a somewhat optimistic error rate. That is, $E_{\mathbf{D}}(X|X)$ is not as reliable as $E_{\mathbf{D}}(X_{te}|X_{tr})$ for assessing *generalization*, but this is not an impediment to using $E_{\mathbf{D}}(X|X)$ as a basis for *comparison* of different designs. Moreover, unless n is very large compared to p and c (an often used rule of thumb is $n \in [10pc, 100pc]$), the credibility of either error rate is questionable. An unfortunate terminology associated with algorithms that reproduce all the labels (i.e., make no errors) upon resubstitution of the training data is that some authors call such a method *consistent* (Dasarathy, 1994). Don't confuse this with other uses of the term, as for example, a consistent statistic.

A third error rate that is sometimes used is called the *validation error* of **D**. This idea springs from the increasingly frequent practice of using X_{te} to decide when **D** is "well trained", by repeatedly computing $E_{\mathbf{D}}(X_{te} | X_{tr})$ while varying the parameters of **D** and/or X_{tr}. Knowing that they want the minimum test error rate, many investigators train **D** with X_{tr}, test it with X_{te}, and then repeat the training cycle with X_{tr} for other choices (such as the number of nodes in a hidden layer of a neural network), until they achieve a minimal or acceptable test error. On doing this, however, X_{te} unwittingly becomes part of the training data (this is called "training on the testing data by Duda and Hart, 1973).

To overcome this complication, some researchers now subdivide X into *three* disjoint sets: $X = X_{tr} \cup X_{te} \cup X_{va}$, where X_{va} is called a *validation set.* When this is done, $X_{tr} \cup X_{te}$ can be regarded as the "overall" training data, and X_{va} as the "real" (or blind) test data. Some authors now report all three of these error rates for their classifiers : resubstitution, test and validation errors. Moreover, some authors interchange the terms test and validation as we have used them, so when you read about these error rates, just make sure you know what the authors mean by each term. We won't bother trying to find a sensible notation for what we call the validation error rate (it would be something like $E_{\mathbf{D}}(X_{va} | X_{te}; X_{tr})$). For the few cases that we discuss in this chapter that have this feature, we will simply use the phrase "validation error" for this third error rate. Finally, don't confuse "validation error" with the term "cross-validation", which is a method for rotating (sometimes called jackknifing) through the pair of sets X_{tr} and X_{te} without using a third set such as X_{va}.

The small data sets used in some of our examples do not often justify worrying about the difference between $E_{\mathbf{D}}(X|X)$ and $E_{\mathbf{D}}(X_{te}|X_{tr})$, but in real systems, *at least* $E_{\mathbf{D}}(X_{te}|X_{tr})$ should always be used, and the selection and manipulation of the three sets $\{X_{te}, X_{tr}, X_{va}\}$ is a very important aspect of system design. At the minimum, it is good practice to reverse the roles of X_{te} and X_{tr}, redesign **D**, and compute (4.1) for the new design. If the two error rates obtained by this "cross validation" procedure are quite different, this indicates that the data used for design and test are somehow biased and should be tested and/or replaced before system design proceeds.

Cross validation is sometimes called "1-fold cross validation", in contrast to k-fold cross validation, where the cross validation cycle is repeated $k > 1$ times, using different pairs (X_{te}, X_{tr}) for each pair of cross validation tests. Terms for these training strategies are far from standard. Some writers use the term "k-fold cross validation" for rotation through the data k time without "crossing" - that is, the total number of training/test cycles is k; "crossing" each time in the sense used here results in 2k train/test cycles. And some authors use the term "cross validation" for the scheme based on the decomposition of X into $\{X_{te}, X_{tr}, X_{va}\}$ just discussed, e.g., (Haykin, 1996). There are a variety of more sophisticated schemes for constructing design and test procedures; see Toussaint (1974) or Lachenbruch (1975) for good discussions of the "rotation" and "leave-one-out" procedures.

There is another aspect to the handling of training and test data in the design of any real classifier system that is related to the fact that training is almost always based on some form of random initialization. This includes most classifiers built from, for example: clustering algorithms, single and multiple prototype decision functions, fuzzy integral classifiers, many variants of sequential classifier designs based on competitive learning models, decision tree models, fuzzy systems, and recognition systems based on neural networks. The problem arises because - in practice - training data are normally limited. So, given a set X of labeled data, the question is: how do you get a good error estimate and yet give the "customer" the best classifier. If the classifier can change due to random initialization (we will see this happen in this chapter), then you are faced with the *training and testing dilemma*:

- If you use all the data to produce (probably) the best classifier you can for your customer, you can only give the resubstitution error rate, which is almost always overly optimistic.

- If you split the data and rotate through different training sets to get better test statistics, then which of the classifiers built during training do you deliver to your customer?

Consider, for example, the leave-one-out estimate of the error rate, in which n classifiers $\{\mathbf{D}_k\}$ are designed with n-1 of the data, and each design is then tested with the remaining datum, in sequence, n times. Since the $\{\mathbf{D}_k\}$ can all behave differently, and certainly will have different parameters, it is not clear that the leave-one-out error rate is very realistic as far as estimating the performance of a delivered system. Averaging the parameters of the n $\mathbf{D}_k$'s, for example, may not give a system that performs anything like any of the tested classifiers.

This is a real world trade-off that, many times, those of us who earn our keep by teaching and doing research tend to ignore. Do we know the answer to this perplexing problem? Nope. The real solution, of course, is to design the classifier with all the data available, and then have someone who is not associated with the design collect a separate test set to generate error statistics. In Section 4.9 we will discuss classifier fusion, one methodology that at least in principle can be used to ameliorate the training versus testing dilemma. Our objective here is to simply point out that constructing a training approach for classifier design is the first step in delivering a workable system, and doing it correctly so that error rate statistics have reliable meanings is far from a trivial consideration.

Crisp labels assigned to data that are collected by domain experts are usually accepted at face value as being physically correct (aside from errors that can always occur), but in many instances the numerical representation of each object is not distinct from a computational point of view. Anderson's (1935) Iris data is a famous example of this. He assigned physical labels to individuals from populations of three subspecies of Iris flowers (Sestosa, Versicolor and Virginica). But the four numerical features he chose to measure for each object (petal width, petal length, sepal width and sepal length) do not provide many algorithms with enough discriminatory power to recognize and represent three algorithmically well-defined classes.

Don't be surprised if your favorite algorithm wants to use a different number of classes than the number of physical labels. It may mean nothing more than the classes are inseparable (to your model, anyway) in the chosen numerical representation. Clustering is sometimes performed on labeled data for just this purpose - to detect whether or not the data do in fact seem to agree with their labels. A danger in doing this is, of course, that clustering algorithms always produce clusters, so algorithmic disagreement does not prove that the data have this disquieting property. On the other hand, agreement is reassuring, and establishing a class (such as "unknown" or "in-between") in labeled data with a clustering algorithm can be used to improve classifier performance by biasing it away from the objects whose representations fall in the overlap portions of the feature space. See House et al. (1999) for a nice application of this technique, where FCM is used with c = 3 to establish an intermediate class in data with c = 2 labeled classes (faulty and non-faulty states in an air handling unit).

Another aspect of training is related to the (much overworked) word *adaptive*, which in our context refers to the style used to acquire the parameters of **D**. So many authors have used this word in so many different ways that we cannot avoid a short discussion of it here. Indeed, we have already used adaptive in Chapter 2 in several ways, principally to distinguish between local cluster estimation (as the GK and adaptive FCV algorithms do) from global approaches such as the c-means models. In the current context we can distinguish three cases:

Non-adaptive off-line training. X_{tr} is used non-iteratively just once to find **D**, and is not revisited with a view towards improving it thereafter. This is the case, for example, when designing a Bayes classifier with labeled data under the assumptions of the normal mixture case discussed in connection with probabilistic clustering in Chapter 2. For i = 1 to c, labeled data $X_{tr,i}$ are used to estimate the parameters of the i-th discriminant function by substitution into analytic necessary conditions, and the design is complete.

Static off-line adaptive training. X_{tr} is used to improve estimates of the parameters of **D** either iteratively or recursively. The most common example of this case is iterative training of a learning model such as a fuzzy system or neural network. In either case input vectors from X_{tr} are used over and over while parameters are adjusted to improve some measure of model performance. Once trained, however, X_{tr} is put aside during the operational phase of classification. A familiar example from calculus may help you to understand this case. Newton's method for estimating a root of the real equation f(x) = 0 adjusts iterative estimates of the root at each step in the algorithm - this is "adaptive learning" in the same sense as it is often used in the literature of learning models.

Dynamic on-line adaptive training. In this scheme the initial classifier might be found using either non-adaptive or adaptive off-line training. As time passes, (features of) the observed data may change, or new data may be available, and the classifier attempts to keep up with these changes by continuously reevaluating (adapting) its parameters in response to changes in the incoming data. Some authors refer to this as a *temporally adaptive classifier.* We all want classifiers that are temporally adaptive, but we are aware of only a very few cases that actually come close to this type of operation.

Classifier performance depends on the quality of X_{tr}. If X_{tr} is large enough and its substructure is well delineated, we expect classifiers trained with it to yield small error rates. On the other hand, when the training data are large in dimension p and/or number of samples n, classifiers such as the *k-nearest neighbor rule* (cf. Section 4.4) can require too much storage and CPU time for efficient deployment. To circumvent time and storage problems caused by very large data sets, as well as to improve the efficiency of supervision by X_{tr}, many authors have studied ways to *edit* the training data (Dasarathy, 1990).

Two common schemes for editing a labeled data set are *selection and replacement*. Selection means : find a *proper subset* $\hat{X}_{tr} \subset X_{tr}$. Replacement means : use a transformation $\Omega: \Re^p \mapsto \Re^p$ to find $\hat{X}_{tr} = \Omega[X_{tr}]$. Subset selection is a special case of replacement. Replacements are almost always labeled prototypes (such as **V** from one of the c-means clustering models) produced by Ω.

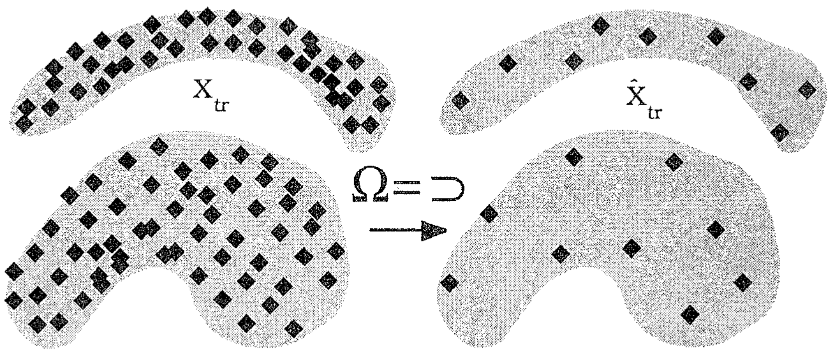


Figure 4.1 Editing by selection of labeled data in X_{tr}

Figure 4.1 depicts *data selection*. (Be careful to distinguish this from *feature selection*, Section 2.6.) The density of labeled data over each cluster in the left side of the figure is high. A selected subset (or skeleton) of the original data is shown on the right. This approach has many variants, and is well summarized in Devijver and Kittler (1982). The aim is to condense X_{tr} while approximately preserving the shape of the decision boundaries set up by training **D** with it.

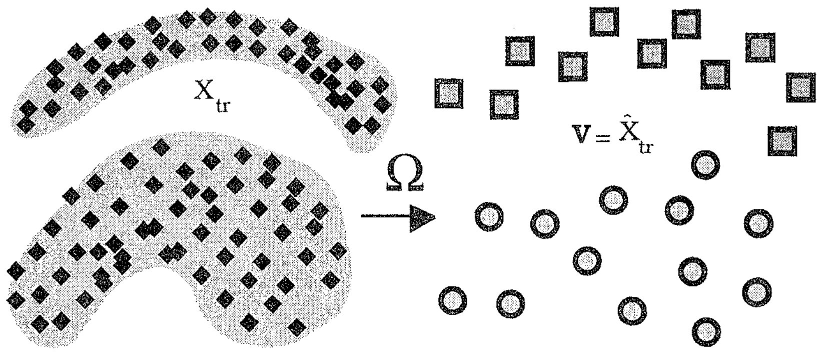


Figure 4.2 Replacing X_{tr} with multiple point prototypes

Figure 4.2 illustrates replacement by multiple point prototypes, where X_{tr} is *replaced* by **V**, a set of labeled prototypes for classes 1 (□) and 2 (○). There is more than one prototype per class in Figure 4.2, but we will use the notation **V** for both the single and multiple prototype cases. The *self-organizing feature map* (SOFM) discussed later is one very good way to accomplish replacement (Kohonen, 1989). It is also possible to replace the data in Figure 4.2 with non-point prototypes (called **B** in Chapter 2) such as rings, lines, hyperquadric surfaces, etc., leading to more sophisticated classifiers that can match prototypical *shapes* to objects having similar representations.

4.2 Prototype classifiers

Prototype representation is based on the idea illustrated in Figure 4.3. The vector $\mathbf{v}_i$ is taken as a prototypical representation for the vectors in the crisp cluster X_i.

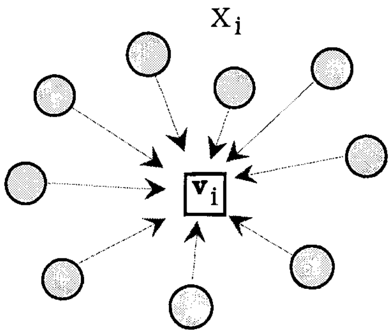


Figure 4.3 Representation of many vectors by a point prototype

For simplicity, our presentation of prototype classifier design is confined to the point prototype case, but you should bear in mind that many of the ideas discussed in this section generalize easily to non-point prototypes. There are many synonyms for the word prototype: centroid, vector *quantizer* (VQ), signature, template, codevector, paradigm, exemplar, etc. In the context of clustering as in Chapter 2 $\mathbf{v}_i$ is usually called the *cluster center* of crisp cluster $X_i \subset X$.

A. The nearest prototype classifier

Once the point prototypes are found (and possibly relabeled to agree most usefully with the data if the training data have physical labels), they can be used to define a crisp nearest prototype (1-np) classifier $\mathbf{D}_{\mathbf{V},\mathbf{E},\delta}$:

Definition 4.1 (1-np classifier). Let **V** be a set of c crisply labeled prototypes, one per class, ordered so that $\mathbf{e}_i$ is the crisp label for $\mathbf{v}_i$, $1 \le i \le c$; let δ be any distance measure on $\Re^p$, and let $(\mathbf{V},\mathbf{E}) = \{(\mathbf{v}_i, \mathbf{e}_i): i = 1,\ldots,c\} \in \Re^{cp} \times N_{hc}^c$. The *crisp nearest prototype* (1-np) classifier $\mathbf{D}_{\mathbf{V},\mathbf{E},\delta}$ is defined, for $\mathbf{z} \in \Re^p$, as

$$\text{Decide } \mathbf{z} \in i \Leftrightarrow \mathbf{D}_{\mathbf{V},\mathbf{E},\delta}(\mathbf{z}) = \mathbf{e}_i \quad \Leftrightarrow \quad \delta(\mathbf{z},\mathbf{v}_i) \le \delta(\mathbf{z},\mathbf{v}_j) \ \ \forall \ \ j \ne i. \tag{4.2}$$

Equation (4.2) says : find the closest prototype to **z**, and assign its label to **z**. Ties are broken randomly. Note that HCM uses (4.2) to determine the crisp memberships shown at (2.6a). The most familiar choice of dissimilarity is the inner product induced norm metric shown in equation (1.6). The crisp 1-np design can be implemented using prototypes from *any* algorithm that produces them. It would be careless to call $\mathbf{D}_{\mathbf{V},\mathbf{E},\delta}$ a fuzzy classifier, for example, just because fuzzy c-means produced the prototypes **V**.

One of the most important classifier structures is the *hyperplane* H_A in $\Re^p$ defined, for any positive definite matrix A, as

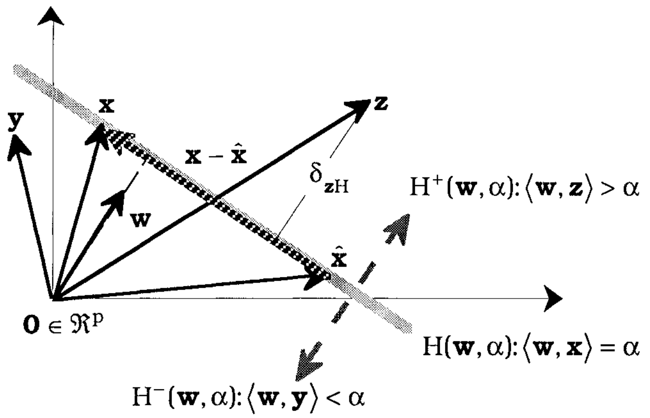
$$H_A(\mathbf{w},\alpha) = \{\mathbf{x} \in \Re^p : \langle \mathbf{x},\mathbf{w}\rangle_A = \mathbf{x}^T A\mathbf{w} = \alpha; \alpha \in \Re\} \quad . \tag{4.3}$$

As usual, when A is the identity, the inner product is Euclidean and we suppress the subscript A. Without loss of generality, we confine further discussion to the Euclidean case.

H(**w**, 0) is a *vector subspace* of dimension p-1 through the origin of $\Re^p$, and $g(\mathbf{x}) = \langle \mathbf{x},\mathbf{w}\rangle$ is a linear function of **x**. H(**w**, α) is a p-1 dimensional *affine subspace* parallel to H(**w**, 0), and the function $g(\mathbf{x}) = \langle \mathbf{x},\mathbf{w}\rangle + \alpha$ is an *affine* function of **x** (linear plus a constant).

The parameter α is the *offset* of the hyperplane H from the origin, and the vector **w** is called a (there are infinitely many) *normal* vector to H, because **w** is perpendicular to H in the given inner product, i.e., whenever a vector, such as $(\mathbf{x} - \hat{\mathbf{x}})$ in Figure 4.4, is parallel to H (lies in H), $\langle (\mathbf{x} - \hat{\mathbf{x}}), \mathbf{w}\rangle_A = 0$.

These properties are illustrated in Figure 4.4, which shows the geometric structure of H in terms of its two parameters. Changing **w** *rotates* H, and changing α *translates* H parallel to itself. The effect of using a weight matrix A other than the identity in (4.3) can now be seen. Letting $\mathbf{w}' = A\mathbf{w}$, $\langle \mathbf{x},\mathbf{w}\rangle_A = \mathbf{x}^T A\mathbf{w} = \mathbf{x}^T(A\mathbf{w}) = \langle \mathbf{x}, A\mathbf{w}\rangle = \langle \mathbf{x},\mathbf{w}'\rangle$, so changing the inducing weight matrix rotates the normal vector **w**, and hence, the hyperplane, keeping it perpendicular to **w** in the new inner product.



$\mathbf{y}$
$\mathbf{x}$
$\mathbf{x}-\hat{\mathbf{x}}$
$\mathbf{w}$
$\mathbf{z}$
δ_{zH}
$H^+(\mathbf{w},\alpha):\langle\mathbf{w},\mathbf{z}\rangle > \alpha$
$\hat{\mathbf{x}}$
$\mathbf{0} \in \Re^p$
$H(\mathbf{w},\alpha):\langle\mathbf{w},\mathbf{x}\rangle = \alpha$
$H^-(\mathbf{w},\alpha):\langle\mathbf{w},\mathbf{y}\rangle < \alpha$

Figure 4.4 Geometry of hyperplanes

For fixed **w** a family of parallel linear varieties (hyperplanes) are generated by (4.3) as α runs through $\Re$. Hyperplanes are the "flat" sets in $\Re^p$, and consequently, $g(\mathbf{x}) = \langle\mathbf{x},\mathbf{w}\rangle + \alpha$ is called a linear decision function even though it is by definition affine. Consequently, classifier functions defined by g are called *linear classifiers* whether g is linear or affine. When **w** is a unit vector, it is routine to check that, given any point in H(**w**, α) such as $\hat{\mathbf{x}}$ in Figure 4.4, the orthogonal distance δ_{zH} from **z** to H(**w**, α) is $\delta_{zH} = \langle\mathbf{z}-\hat{\mathbf{x}},\mathbf{w}\rangle$.

As illustrated in Figure 4.4, H divides $\Re^p$ into three disjoint sets, viz., $\Re^p = H^- \cup H \cup H^+$. The set H^+ is the *positive half-space* associated with H, so called because, as shown in Figure 4.5, every vector **z** that lies "above" H (and therefore in H^+) yields a value for the dot product that is greater than α, $\langle\mathbf{z},\mathbf{w}\rangle > \alpha$. Similarly, vectors **y** that lie in the *negative half space* H^- yield $\langle\mathbf{y},\mathbf{w}\rangle < \alpha$; and of course, for vectors **x** in H, $\langle\mathbf{x},\mathbf{w}\rangle = \alpha$. H is called a *separating hyperplane* between its two half spaces, and when a labeled data set X with c = 2 classes can be completely separated by H (so that all points from one class lie on one side of H, while all points from the other class lie on the opposite side), X is said to be *linearly separable.*

The geometry of the crisp 1-np inner product norm classifier is shown in Figure 4.5, using Euclidean distance for δ. This 1-np design erects a linear boundary between the i-th and j-th prototypes, viz., the hyperplane H(**w**, α) through the midpoint of and perpendicular to the line joining $\mathbf{v}_i$ and $\mathbf{v}_j$. Figure 4.5 illustrates the labeling decision represented in equation (4.2); vector $\mathbf{z}$ is assigned to class i because it is closest to the i-th prototype. Some authors use the terms prototypes and neighbors interchangeably, but we will consistently call *nearest prototypes* new vectors made *from* the data *or* points in the data, while *nearest neighbors* are labeled points *in* the data.

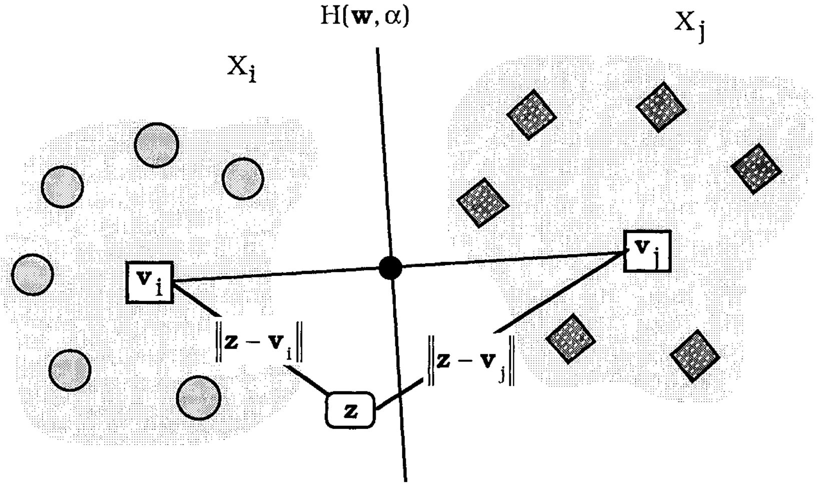


Figure 4.5 The 1-np classifier for the Euclidean norm

All 1-np designs that use inner product norms erect (piecewise) linear decision boundaries. Thus, the *geometry* of 1-np classifier *boundaries* is fixed by the way distances are measured in the feature space; and *not* by geometric properties of the model that produces the cluster prototypes. The location in $\Re^p$ of the prototypes determines the location and orientation of the c(c-1)/2 hyperplanes that separate pairs of prototypes. The locations of the prototypes do depend importantly on both the computational model and data used to produce them. Hence, 1-np classifiers based on different prototype generating schemes can certainly be expected to yield different error rates, even though they all share the same type of decision surface structure.

Example 4.1 Table 4.1 lists 20 vectors in $\Re^2$ in c = 2 labeled classes, 10 apples in class 1 (crisp label $\mathbf{e}_1$), and 10 pears in class 2 (crisp label $\mathbf{e}_2$). $\mathbf{v}_A = \bar{\mathbf{v}}_1 = (1.22, 0.40)^T$ and $\mathbf{v}_P = \bar{\mathbf{v}}_2 = (2.43, 1.03)^T$ are the sample mean vectors for the apples and pears, respectively. These two prototypes are listed in the last row of Table 4.1 and appear graphically along with the 20 labeled data in Figure 4.6.

Table 4.1 Apples and pears data

$\mathbf{e}_i$	$\mathbf{x}_i$	x_i	y_i	$\mathbf{e}_i$	$\mathbf{x}_i$	x_i	y_i
🍎	1	1.00	0.60	🍐	11	2.00	0.70
🍎	2	1.75	0.40	🍐	12	2.00	1.10
🍎	3	1.30	0.10	🍐	13	1.90	0.95
🍎	4	0.80	0.20	🍐	14	2.00	0.95
🍎	5	1.10	0.70	🍐	15	2.30	1.20
🍎	6	1.30	0.60	🍐	16	2.50	1.15
🍎	7	0.90	0.50	🍐	17	2.70	1.00
🍎	8	1.60	0.60	🍐	18	2.90	1.10
🍎	9	1.40	0.15	🍐	19	2.80	0.90
🍎	10	1.00	0.10	🍐	20	3.00	1.05
	$\mathbf{v}_A = \overline{\mathbf{v}}_1 \rightarrow$	1.22	0.40		$\mathbf{v}_P = \overline{\mathbf{v}}_2 \rightarrow$	2.41	1.01

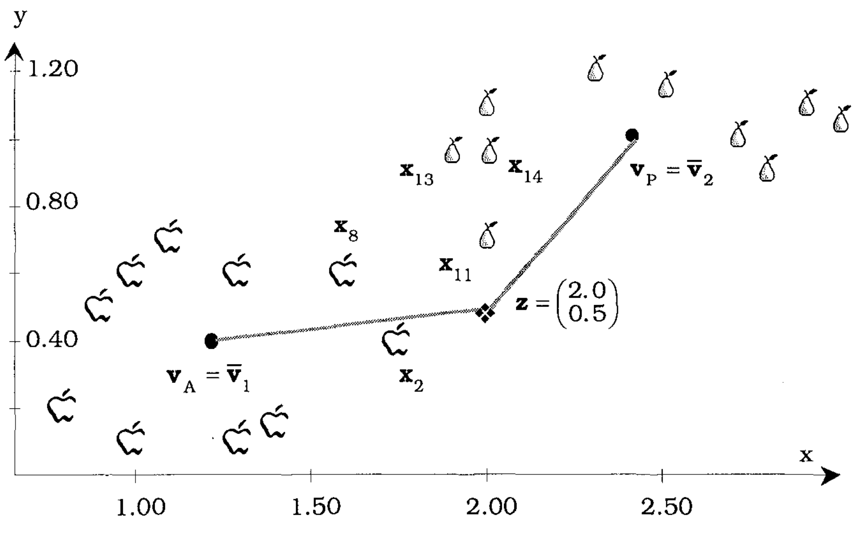


Figure 4.6 Data in Table 4.1 and their sample mean prototypes

Once the prototypes - which in this example are the sample means $\overline{\mathbf{V}}$ - are determined, (their physical labels are known since the set **E** is known, and each prototype is built from data with only one class label), we need only to choose a distance measure to implement the 1-np classifier in (4.2). Choosing Euclidean distance, suppose that the vector $\mathbf{z}$ = $(2.0, 0.5)^T$ shown in Figure 4.6 is unlabeled, and we submit it to this 1-np classifier. The distances $\delta(\mathbf{z}, \mathbf{v}_P) = 0.68$ and $\delta(\mathbf{z}, \mathbf{v}_A) = 0.79$ then yield $\delta(\mathbf{z}, \mathbf{v}_P) < \delta(\mathbf{z}, \mathbf{v}_A) \Rightarrow \mathbf{D}_{\mathbf{V},\mathbf{E},\delta}(\mathbf{z}) = \mathbf{e}_2$ ="pear".

The geometry shown in Figures 4.4 and 4.5 is illustrated by computing three parameters. First we calculate a normal to H, which is any scalar multiple of the intermean vector

$$\mathbf{w} = \mathbf{v}_P - \mathbf{v}_A = (1.19,\ 0.61)^T \quad . \tag{4.4}$$

With δ as Euclidean distance, we compute

$$\frac{\left|\|\mathbf{v}_P\|^2 - \|\mathbf{v}_A\|^2\right|}{2\|\mathbf{w}\|} = \frac{|\alpha|}{\|\mathbf{w}\|} = 1.94 \quad . \tag{4.5}$$

Since $\|\mathbf{v}_P\| > \|\mathbf{v}_A\|$, (4.5) yields $\alpha = 2.59$. To graphically construct the hyperplane $H_{AP}(\mathbf{w}, \alpha)$ just found for the apples and pears data, we find a third parameter, the midpoint **m** of the line joining $\mathbf{v}_P$ to $\mathbf{v}_A$,

$$\mathbf{m} = (\mathbf{v}_A + \mathbf{v}_P)/2 = (1.82, 0.71)^T \quad . \tag{4.6}$$

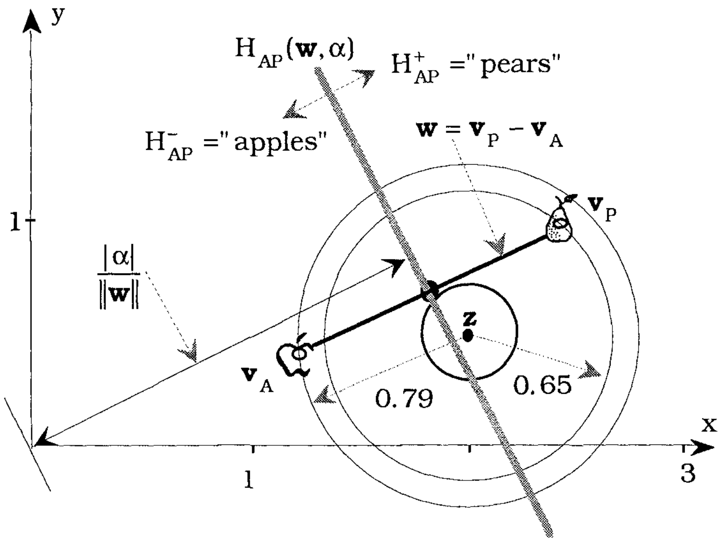

y
$H_{AP}(\mathbf{w},\alpha)$
H^+_{AP} = "pears"
H^-_{AP} = "apples"
$\mathbf{w} = \mathbf{v}_P - \mathbf{v}_A$
$\mathbf{v}_P$
1
$\frac{|\alpha|}{\|\mathbf{w}\|}$
z
$\mathbf{v}_A$
0.79
0.65
x
1
3

Figure 4.7 The 1-np classifier for Example 4.1

The geometric structure of this classifier is shown in Figure 4.7. The decision that **z** be labeled a pear can be reached another way by simply calculating $\langle \mathbf{z}, \mathbf{w} \rangle = 2.69 > \alpha = 2.59$. This tells us that **z** has landed in the pears *decision region* $H^+_{AP}(\mathbf{w},\alpha)$ as shown in Figure 4.7; similarly, the apples decision region is the negative half-space $H^-_{AP}(\mathbf{w},\alpha)$. Although you cannot see it in Figure 4.7 (because the data are not shown), $H_{AP}(\mathbf{w}, \alpha)$ is a *separating hyperplane* between the apples (A) and pears (P) regions - that is, these data are linearly separable.

Geometrically, the 1-np classifier grows neighborhoods about the point **z** that take their shape from the topology induced by the metric δ. The circles in Figure 4.7 remind you that the norm is Euclidean, so the shape of the neighborhoods is circular. Changing the metric changes the shape of the neighborhoods. For example, if the 1- norm had been used instead, the neighborhoods would be diamond shaped as shown in Figure 2.11, and the hyperplane structure illustrated here would be invalid, since the 1-norm is not inner product induced.

B. Multiple prototype designs

What can we do when a single prototype is not sufficient to describe a class accurately? This can easily happen when feature vectors that possess the same physical label for a particular class fall into two or more clusters, as in the famous "XOR" data that cannot be separated by a single hyperplane (Zurada, 1992). For example, defective parts may have oversized holes drilled into them or they may have surface defects in the material. If those two defects are manifested in the measured feature vectors, then the defective-part class could have three clusters, one where the "hole diameter" is big, one where the "material homogeneity" is low, and one where both problems are present. Single prototype classifiers will not provide good classification accuracy in this situation. Another situation that can require multiple prototypes for a single class is when two physically labeled classes overlap in the chosen feature space (as in classes 2 and 3 of the Iris data). In this case, and for that matter, in almost all real data sets, it is advantageous to have several prototypes for each class.

Definition 4.2 (1- nearest multiple prototype (1-nmp) classifier). $(\mathbf{V}_c, \mathbf{E}_{\hat{c}}) = \{(\mathbf{v}_j, \mathbf{e}_{i(j)}): j = 1,\ldots,c; i(j) = 1,\ldots,\hat{c}\} \in \Re^{cp} \times N^{\hat{c}}_{h\hat{c}}$. Here X has $\hat{c}$ classes, $\hat{c} \le c$, $\mathbf{V}_c$ is a set of c crisply labeled prototypes, with more than one per class for at least one class if $\hat{c} < c$, $\mathbf{e}_{i(j)}$ labels $\mathbf{v}_j$ as class i, and δ is any distance measure on $\Re^p$. The *crisp 1- nearest multiple prototype* (1-nmp) classifier $\mathbf{D}_{\mathbf{V}_c,\mathbf{E}_{\hat{c}},\delta}$ is defined, for $\mathbf{z} \in \Re^p$, as

$$\text{Decide } \mathbf{z} \in i \Leftrightarrow \mathbf{D}_{\mathbf{V}_c, \mathbf{E}_{\hat{c}}, \delta}(\mathbf{z}) = \mathbf{e}_{i(j)} \quad \Leftrightarrow \quad \delta(\mathbf{z}, \mathbf{v}_j) \le \delta(\mathbf{z}, \mathbf{v}_s) \;\; \forall \;\; s \ne j. \qquad (4.7)$$

When c = ĉ equation (4.7) reduces to (4.2). Two opportunities arise from this simple extension of the 1-np design. First, we now have more flexibility to generate prototypes, as will be discussed in the next section. Perhaps a bigger opportunity, however, afforded by the increase of exemplars from ĉ to c, is the possibility of assigning fuzzy labels to the prototypes, and hence, to construct fuzzy decision rules with them. Instead of discussing this prospect here, we will postpone it to Section 4.4 on nearest neighbor rules since, in the "limit" case i.e., when ĉ=n, we can consider each training vector as a prototype. In other words, the decision rules (crisp, fuzzy and possibilistic) that are described for the k-nearest-neighbor classifiers in Section 4.4 can be implemented in the multiple prototype framework.

Example 4.2 This example demonstrates a novel use of multiple prototypes in a real world application: detection of landmines. The landmine problem has become a crisis in the world. It is estimated that more than 100 million active mines are scattered in 62 countries, with an equal number stockpiled around the world just waiting to be planted. Landmines kill or maim approximately 26,000 innocent civilians every year.

Currently, landmines are detected individually by prodding, metal detection or dogs. Gently prodding the ground is slow, confusing and dangerous, especially when the mines are laid in hard-packed or stony soil. Metal detection works well with metal mines, but recently, metal has been increasingly replaced by plastic. Dogs are effective, but like humans, can become easily distracted.

A variety of sensors have been proposed or are under investigation for landmine detection. In view of the life threatening nature of this application, it is desirable to have a very high detection rate with a low false alarm rate. However, many sensors can detect land mines reliably only at the expense of a high false alarm rate.

Frigui et al. (1998a) and Gader et al. (1998a, b) consider the problem of detecting landmines with sensor data obtained from a novel, three-dimensional *Ground Penetrating Radar* (GPR) system developed by Geo-Centers, Inc. (Rappaport and Reidy, 1996). Following Frigui et al. (1998a), multiple prototypes of objects and background are first generated by fuzzy clustering of features generated from the GPR imagery. Rather than use the prototypes generated from the clustering algorithm to form a nearest (multi-) prototype classifier, the authors used them to provide a more reliable estimate of the strength of the radar return from a particular spatial location.

The Geo-Centers GPR system is mounted on the front of a moving truck. Every two inches of forward travel in the y direction, a scan is formed by sweeping the radar signals across 16 bins in the x direction perpendicular to travel (cross-track) and 64 bins down into the ground in the time = t direction, thereby producing a 64 x 16 array of intensity values I(t, x, y). For fixed y, the array is referred to as a *scan*. A scan is formed every 2 inches, thereby producing a volume of data. Figure 4.8 depicts one such scan.

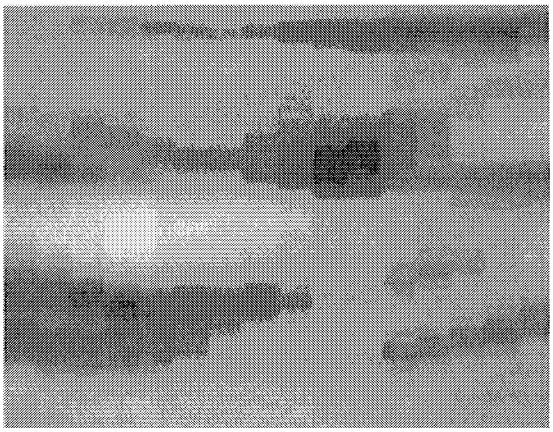


Figure 4.8 A typical 64 x 16 scan from the Geo-Centers GPR

We can look at the data from a different perspective by holding x constant and letting y and t vary, generating what we refer to as a *vertical plane*. A typical vertical plane from the *Defense Advanced Research Projects Agency* (DARPA) backgrounds data is shown in Figure 4.9.

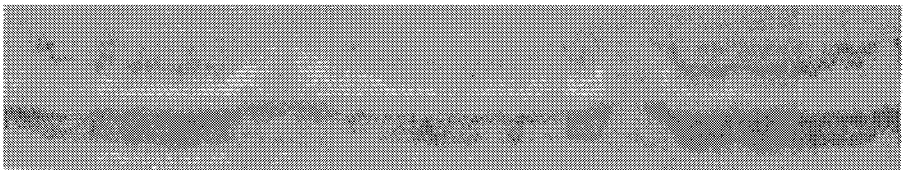


Figure 4.9 Vertical slice (down-track)

A 6-dimensional feature vector **f**(t, x, y) is computed at each point (t, x, y) and then used to evaluate membership in fuzzy sets defined by feature prototypes. **f**(t, x, y) is a vector of edge magnitudes from points in a pattern around (t, x, y) that roughly resembles the signature of a mine in a vertical plane. Let E(t, x, y) denote the edge strength in the horizontal direction (down-track). Since the shape of mine is variable and there is a considerable amount of uncertainty, the edge strengths are averaged in the vertical direction:

$$A(t,x,y) = \frac{1}{7}\left(\sum_{k=-3}^{3} E(t+k,x,y)\right) \quad . \tag{4.8}$$

For the experiment discussed in Frigui et al. (1998a), this value was clipped at 150. The 6-D feature vector is given by

$$\mathbf{f}(t,x,y) = \begin{pmatrix} A(t+5,x,y-5) \\ A(t+3,x,y-3) \\ A(t+1,x,y-1) \\ A(t+1,x,y+1) \\ A(t+3,x,y+3) \\ A(t+5,x,y+5) \end{pmatrix} \quad . \tag{4.9}$$

The goal is to use the features generated from a "calibration lane" to determine prototypes, and then to apply those prototypes in a classifier on test mine lanes. Generally, target pixels in GPR data constitute less than 5% of the data. Hence, traditional FCM-type algorithms have problems due to the large difference in size of the target and background clusters. Instead, the competitive agglomerative or CA clustering algorithm (cf. equations (2.75)-(2.81)) was run on the calibration lane. This choice may not be the best one for discovering clusters (because of greatly unequal cluster population sizes, the problem illustrated in Figure 2.3(a)), but the authors felt it was a good choice for finding multiple prototypes.

One prototype was sufficient for the background, whereas several were needed to describe the variation in the mine responses. The algorithm was run using Euclidean distance for FCM with $c = 6$, $m = 2$, and $\varepsilon = 0.1$. The prototypes were initialized heuristically based on "expected" gradient patterns for background and objects. The initialization was

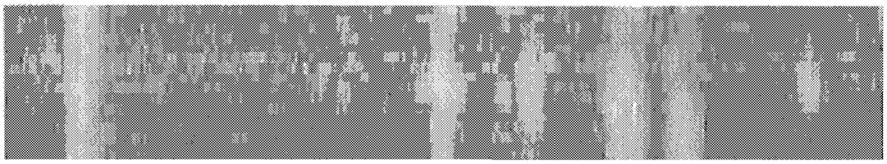
$$V = \begin{pmatrix} 50 & 150 & 150 & 50 & 50 & 150 \\ 50 & 150 & 150 & 50 & 150 & 150 \\ 50 & 150 & 150 & 50 & 150 & 50 \\ 50 & 150 & 50 & 150 & 150 & 50 \\ 50 & 150 & 50 & 150 & 150 & 150 \\ 50 & 150 & 50 & 150 & 50 & 150 \end{pmatrix} \quad .$$

FCM was run for two iterations to "prime" the partition matrix. Then the CA algorithm was run to termination (with a maximum of 30 iterations). Instead of using the prototypes directly in a classifier with $\hat{c} = 2$ as in (4.7), Frigui et al. (1998a) used the non-background prototypes to supply partial evidence for the confidence that a mine-like object is present at a point (t,x,y). This is due to the high degree of uncertainty present in the landmine detection problem.

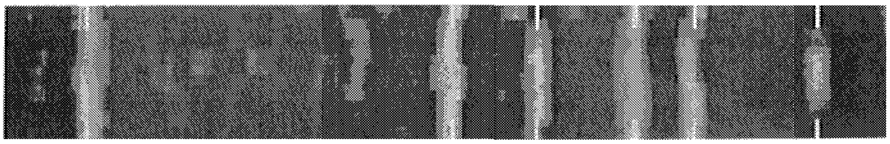
Specifically, the strength of the gradient values represented by the prototypes directly relate to the presence of an object reflecting the radar wave. Hence, in the test lanes, the inverse distance from each non-background prototype to a feature vector was calculated, and the mine confidence membership c(t, x, y) was generated as the weighted sum of the inverse distances, where the weights were proportional to the magnitudes of the prototypes. The confidence that a mine is present at a point on the surface was then computed as

$$\mathrm{conf}(x, y) = \max_t(c(t, x, y)) \quad . \tag{4.10}$$

The confidence map on the surface of a mine lane was then smoothed and a size-contrast filter applied to eliminate large "bright regions". Figure 4.10 shows the confidence and size-contrast filter outputs on part of Dirt Lane 17 containing the following mines: CULVERT (this is not a mine), M19, VS2.2, M15, TM46, VS2.2 at the positions indicated in the size-contrast output.



(a) Raw confidence map for a dirt test lane



(b) Output of the size-contrast filter with the object locations marked

Figure 4.10 Confidence and size -contrast filter outputs

A threshold was generated on the training data (which gave 100% detection), and then it was fixed for all of the tests. The hits were then examined to produce final detection marks in the tests. This initial approach at using multiple crisp point prototypes generated by a fuzzy model was tested on data collected by Geo-Centers at Fort A.P. Hill in October 1996. The data was collected from four passes over two mine lanes by Geo-Centers (two passes over each lane). The standard approach for GPR-based mine detection was to threshold the energy signature produced by the GPR at each point (x,y) formed by summing the values over t. Table 4.2 shows the results of the standard approach, while Table 4.3 lists the multi-prototype confidence results.

Table 4.2 Results from standard approach on mine lanes

Lane	No. of mines detected	No. of mines	No. of false alarms
1	13	19	49
2	15	19	43
3	16	20	49
4	19	21	39

The number of mines detected is not increased by using multiple prototypes, but the number of false alarms is significantly reduced. This is a good result, because false alarms caused by sensor noise, clutter, algorithmic processing, etc. are the major problem in mine remediation activities.

Table 4.3 Results of Multi-prototype approach on the same data

Lane	No. of mines detected	No. of mines	No. of false alarms
1	15	19	13
2	13	19	9
3	19	20	10
4	18	21	12

Much work continues to be done towards improving the sensing modalities and detection algorithms in this area. This example demonstrates the advantage that can be gained by a fairly simple application of multiple prototypes acquired by a fuzzy model over the simpler nearest prototype classifier.

4.3 Methods of prototype generation

Nearest prototype classifiers are simple, effective and cool. However, you got to pay your dues if you want to use (them). That is, you have to generate the prototypes, and you know that don't come easy! That's what this section is about. Roughly speaking, there are three approaches to prototype generation : (i) models such as the leader algorithm and sequential HCM (Hartigan, 1975), and batch models such as the c-means models (Chapter 2); (ii) network models such as learning vector quantization and its generalizations (Kohonen, 1989) and the generalized Lloyd algorithm (Gersho and Gray, 1992); and (iii) statistical models such as mixture decomposition (subsection 2.2.C).

The common denominator in all prototype generation schemes is a mathematical definition of how well prototype $\mathbf{v}_i$ represents X_i. Any measure of similarity on $\Re^p$ can be used. The usual choice is distance (dissimilarity), the most convenient is squared distance, and the most popular is squared Euclidean distance. *Local methods* attempt to optimize some function of the c squared distances $\left\{\|\mathbf{x}_k - \mathbf{v}_i\|_A^2 : 1 \le i \le c\right\}$ at each $\mathbf{x}_k$ in X_i. *Global methods* usually seek extrema of some function of $\left\{\|\mathbf{x}_k - \mathbf{v}_i\|_A^2 : 1 \le i \le c; 1 \le k \le n\right\}$, i.e., all cn squared distances. Don't confuse our use of the terms local and global *methods* with the local and global *extrema* found by a particular method.

One of the simplest approaches to multiple prototype generation when crisply labeled data are available is to run *any* clustering algorithm (e.g., from Chapter 2) that generates prototypes on the training data $X_{tr,i}$ one class at a time. This generates one or more prototypes for class i - already labeled by $\mathbf{e}_i$ - which can then be used for classifier design. All of the issues raised in Chapter 2 about clustering such as choice of distance and validation are relevant when clustering in $X_{tr,i}$.

Another way to find prototypes with a clustering algorithm is to run $\mathcal{C}$ on the entire labeled training set X_{tr} in an unsupervised mode (that is, simply ignoring the labels during training). When this is done using the knowledge that there are c labeled subsets in the training data, the result is (presumably) one prototype per class. Why do this if you have labeled data? We pointed out that the Iris data has 3 physically labeled classes, but that most researchers regard it as having 2 *geometrically* well separated clusters in the 4 dimensional feature space that was chosen by Anderson (1935). From the botanical point of view then, c = 3 is certainly the most useful interpretation of Iris, but from the computational viewpoint, forcing three clusters on this data strains algorithms that want it to have but two. Ignoring the labels during clustering *may* enable $\mathcal{C}$ to discover *geometrically better* prototypes than the labeled sample means for the classes because this allows geometric properties of the data (which are not necessarily captured by their labeled representatives) to drive the model towards a more useful solution.

A third possibility is to run any clustering algorithm (unsupervised, by definition) on all of X_{tr}, again ignoring the given physical labels, but with values for c that are greater than the given number of class labels ($\hat{c}$ in Definition 4.2). This introduces the necessity for cluster validation, but with labeled test data and a well defined performance objective (viz., minimum apparent error rate), this is less of a problem than with truly unlabeled data. This leads to multiple prototypes for classifier design.

The result of clustering X_{tr} in the unsupervised mode at any value of c is a set of prototypes with algorithmic labels. Now the given labels in $N_{h\hat{c}}$ for X_{tr} must be put into play, usually by a relabeling algorithm that assigns physical labels to the algorithmic prototypes. So, the labels are used in any case, and the data itself will determine which method (clustering in $X_{tr,i}$ or in X_{tr}) is more productive. Since X_{te} is available to test classifiers designed using both strategies, that is what we recommend - try both.

There also are many, many prototype generation algorithms based on crisp and fuzzy models that are not, per sé, clustering algorithms. Indeed, it would take an entire monograph to adequately discuss prototype generation methods. The best we can do here is review and illustrate a few methods not discussed in Chapter 2 that are fuzzy, have been generalized to a fuzzy case, or have appeared in connection with a fuzzy model in the literature. Many models of this kind are competitive learning models, the topic we now turn to.

A. Competitive learning networks

The primary goal for *competitive learning* (CL) models is to portray the input data by a much smaller number of prototypes that are good representatives of structure in the data *for classifier design.* Prototypes that are good for classifier design are not necessarily the same (even in form) as those that are used for other purposes. For example, prototypes good for compression, transmission and reconstitution of images may be quite poor as representatives of classes for pixel labeling in the same image. Identification of clusters is implicit, but not active, in the pursuit of this goal.

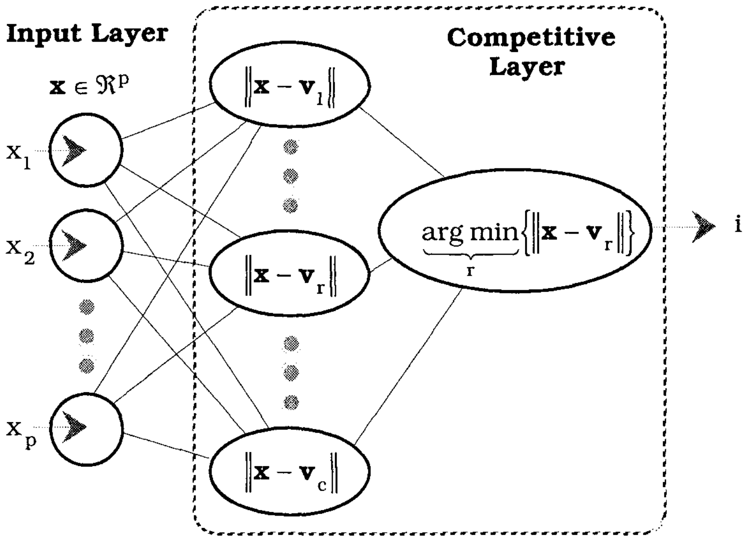

Input Layer
$\mathbf{x} \in \Re^p$
x_1
x_2
x_p
Competitive Layer
$\|\mathbf{x}-\mathbf{v}_1\|$
$\|\mathbf{x}-\mathbf{v}_r\|$
$\|\mathbf{x}-\mathbf{v}_c\|$
$\underbrace{\arg\min}_{r}\{\|\mathbf{x}-\mathbf{v}_r\|\}$
i

Figure 4.11 A general competitive learning network

The salient features of one general CL model are shown in Figure 4.11. The input or fanout layer is connected directly to the output layer. The circles in Figure 4.11 are sometimes called *nodes*, and the prototypes are then called node weights. In this context the p components $\{v_{ji}\}$ of $\mathbf{v}_i$ are often regarded as weights or connection strengths of the edges that connect the p inputs to node i. The prototypes $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_c)$, $\mathbf{v}_i \in \Re^p$ for $1 \le i \le c$, are the (unknown) vector quantizers we seek. The norm used in competitive layer nodes is most typically Euclidean, but there is no overpowering reason to restrict the measure of distance this way.

Sequential CL models update estimates of one or more of the $\{\mathbf{v}_i\}$ at each of n input events during pass t (one iteration is one pass through X). Upon presentation of an $\mathbf{x}_k$ from X, the general form of the update equation is:

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}),\, i = 1,\ldots, c;\ t=1,\ldots,T \quad . \tag{4.11}$$

See Figure 4.81 for an illustration of the geometric meaning of (4.11), which is just vector addition, with the length of the side parallel to the difference vector between the input and the prototype controlled by learning rate $\alpha_{ik,t}$. In (4.11) $\{\alpha_{ik,t}\}$ is the *learning rate distribution* over the c prototypes for input $\mathbf{x}_k$ during iterate t. When $\mathbf{x}_k$ is submitted to this network, distances are computed between it and each $\mathbf{v}_j$. The output nodes "compete", a (minimum distance) winner node, say $\mathbf{v}_i$, is found ; and finally, it and possibly other prototypes are then updated using one of many update rules that are most often of the form (4.11). There are at least four cases :

(i) Only $\mathbf{v}_i$ is updated	(*winner* take all, LVQ, SHCM e.g.)
(ii) Only one $\mathbf{v}_j$ is updated	(*some* vector takes all, ART1, e.g.)
(iii) Some $\mathbf{v}_j$'s are updated	(elite updates, SOFMs, e.g.)
(iv) Every $\mathbf{v}_j$ is updated	(all share updates, GLVQ -F, e.g.)

The acronyms we just used are : *learning vector quantization* (LVQ), *sequential hard c-means* (SHCM), *adaptive resonance theory* (ART), *self-organizing feature maps* (SOFMs) and *generalized learning vector quantization - fuzzy* (GLVQ-F). The prototypes that get updated (the *update neighborhood*) depend on the model chosen, and the update neighborhood can be imbedded in the definition of the learning rates for a particular model. A template that can be used for many CL models is given in Table 4.4.

Table 4.4 A general CL algorithm for unlabeled data

	A. Training phase : find $\mathbf{V}$ without U_{tr}	
Store	(Un)labeled Object Data $X_{tr} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$	
	$U_{tr} \in M_{hcn}$ = Labels of vectors in X_{tr}	
Pick	☺ number of nodes : $1 < c < n$	
	☺ max. # of iterations : T	
	☺ distance measure : $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|_A$	
	☺ termination measure : $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|$	
	☺ termination criterion : ε	
	▷ *special choices for a particular model*	
Get	☺ initial prototypes: $\mathbf{V}_0 \in \Re^{cp}$	
Do	$t \leftarrow 1; E_0$ = high value	
	DO UNTIL ($t > T$ or $E_{t-1} \le \varepsilon$)	
	For $k = 1$ to n	
	$\mathbf{x} \in X, \ \mathbf{x}_k \leftarrow \mathbf{x}, \ X \leftarrow X - \{\mathbf{x}_k\}$	
	Get distances $\left\{\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|; 1 \le i \le c\right\}$	(4.12a)
	Get learning rates $\{\alpha_{ik,t}; 1 \le i \le c\}$	(4.12b)
	$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1})$	(4.12c)
	Next k	
	$t \leftarrow t + 1$	
	END UNTIL	
	$\mathbf{V} \leftarrow \mathbf{V}_{t-1}$	

B. Prototype relabeling of $\mathbf{V}$ with U_{tr} using, e.g., equation (4.13)

C. Optional (crisp) clusters if U_{tr} is unknown,with, e.g., (2.6a) :

$$u_{ik} = \left\{\begin{matrix} 1; & \|\mathbf{x}_k - \mathbf{v}_i\| < \|\mathbf{x}_k - \mathbf{v}_j\| \ , \ 1 \le j \le c, j \ne i \\ 0; & \text{otherwise. Resolve ties arbitrarily} \end{matrix}\right\} \forall i, k$$

Several points need to made about Table 4.4. Notice that the data are considered to be *unlabeled* in step A, even if they are not. The labels for X_{tr} are used in step B *after the training phase is completed* to assign a physical label to each prototype. Different ways to use the labels in the context of CL models such as LVQ1-LVQ3 are discussed by Kohonen (1989). In either case, notice especially that no partition is needed or generated in training step A.

In the general CL model of Table 4.4, any norm can be used in (4.12a) and in Step C. Computation of the learning rates in (4.12b) is not specific in Table 4.4. Different models require choosing various parameters (▷ *special choices* in the "pick" block of the table), and all of them compute quantities which are functions of the distances in (4.12a). Thus, a good specific implementation might be laid out a little differently than the one shown in Table 4.4. We will identify the items needed for each model discussed in Chapter 4, and trust to your good judgment as to how best arrange the code for an actual implementation.

One of the main differences between various CL schemes is the form for the learning rate distribution (including the update neighborhood) in (4.12b). Prototype updating in (4.12c) cannot be done until the learning rates are well defined. Generally - but not very often - α is a function of i, k and t, but in some models it is fixed for all k's during each pass through X, and then we write $\alpha_{i,t}$. Most frequently, α is fixed for both i and k, depending only on t; in this case we write α_t. Infrequently, only one pass is made through X, in which case we write $\alpha_{i,k}$. The sign of α determines whether the update in (4.11) moves $\mathbf{v}_{i,t-1}$ towards $\mathbf{x}$ (attraction) or away from $\mathbf{x}$ (repulsion). Most competitive learning models use only positive learning rates, but there are algorithms that use negative learning rates for vectors that are far from the update neighborhood (e.g., the so called "Mexican hat function" discussed in Kohonen, 1989). We will discuss this more in connection with Figure 4.81.

The standard method of achieving stability for prototypes (we will make this notion specific in Section 4.8.A) is to begin with values for the $\{\alpha_{ik,t}\}$ close to, but less than, 1; and then to decrease the $\{\alpha_{ik,t}\}$ towards zero as time (iteration number t) increases. If $\alpha_{ik,t} \to 0$, updates will become very small, and so will successive estimates of the prototypes. This is how termination of many (but not all) competitive learning algorithms is effectively achieved. But… this strategy causes a problem that Grossberg (1976a, b) recognized and called the *plasticity* problem. We will return to this idea in Section 4.8.A.

The optional clustering phase, Step C in Table 4.4, produces n crisp label vectors *for the points in* X_{tr}. They are *usually* (usually, because there is no guarantee that each of the c classes defined by the nearest prototypes has at least one point in it) a crisp c-partition of X. This optional step, or one like it using some other strategy, often leads to semantic confusion. For example, Yager and Filev's (1994a) mountain clustering method, which does not produce clusters without using an equation such as (2.6a) after termination of the training phase, is incorrectly called a clustering algorithm. More precisely, it is a prototype generation algorithm whose terminal prototypes can be used to find clusters.

This terminology is fairly pervasive however. Any c point prototypes $\mathbf{V} = \{\mathbf{v}_1,\ldots,\mathbf{v}_c\} \subset \Re^p$ can be substituted into (2.6a), and the result is a crisp partition, say $U(\mathbf{V}) \in M_{hcn}$, which is sometimes called the *nearest prototype* (np) partition of X. When we want to emphasize this construction of U from **V** with (2.6a), we write $U(\mathbf{V}) = U_{np}(\mathbf{V})$. Moreover, subsequently applying (2.6b) to the rows of $U_{np}(\mathbf{V})$ results in the sample means, $\mathbf{V} = \overline{\mathbf{V}}$. Under these circumstances it is not incorrect to regard the prototypes $\mathbf{V} = \overline{\mathbf{V}}$ as a *representation* of the crisp partition $U_{np}(\mathbf{V})$, and this is why many point prototype generator algorithms are called clustering algorithms. Recognizing this, we nevertheless reserve the term "clustering algorithm" for those models that actively involve a partition of X during training, and in this sense the CL models embodied as special cases of the general scheme in Table 4.4 are not clustering algorithms.

A final comment: most CL models are not explicitly designed to find good clusters in the sense that partitions of the data are never examined during the training phase. Consequently clusters built "after the fact" by approaches such as step C of Table 4.4 may or may not be satisfactory in the sense of partitioning X for substructure. Forewarned, don't be surprised if a CL model produces unsatisfactory clusters in unlabeled data - that's not its job.

B. Prototype relabeling

What should we do when the labels of points in X_{tr} are not used during training to guide iterates towards a good **V**? In this case, at the end of the learning phase the c prototypes have *algorithmic* labels that may or may not correspond to the *physical* labels of X_{tr}. The relabeling algorithm discussed next uses the labels in U_{tr} to attach the most likely (as measured by a simple percentage of the labeled neighbors) physical label to each $\mathbf{v}_i$.

Recall that $\hat{c}$ is the number of classes in X_{tr}, labeled by the crisp vectors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{\hat{c}}\} = N_{h\hat{c}}$ Now define p_{ij}, i=1,2,..., $\hat{c}$, j=1,2,..., c, to be the percentage (as a decimal) of training data from class i closest to $\mathbf{v}_j$ via the 1-np rule. Matrix $P = [p_{ij}]$ has $\hat{c}$ <u>rows</u> in N_{fc}, and c <u>columns</u> $\mathbf{p}_j$ in $N_{p\hat{c}}$. We assign label $\mathbf{e}_i$ to $\mathbf{v}_j$ when $\mathbf{H}(\mathbf{p}_j) = \mathbf{e}_i$, with ties broken arbitrarily,

$$\text{label } i \leftarrow \mathbf{v}_j \Leftrightarrow \mathbf{H}(\mathbf{p}_j) = \mathbf{e}_i \;\; ; \;\; i = 1,2,\ldots,\hat{c} \; ; \; j = 1,2,\ldots,c. \qquad (4.13)$$

We illustrate the labeling algorithm at (4.13). Suppose X_{tr} has $\hat{c} = 3$ classes, labeled with the crisp vectors $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} = N_{h3}$. Let $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)$ be four prototypes found by some algorithm. Let P be the 3 × 4 percentage matrix shown in Table 4.5. Labeling algorithm (4.13) assigns $\mathbf{v}_1$ to class 1, $\mathbf{v}_2$ and $\mathbf{v}_3$ to class 3, and $\mathbf{v}_4$ to class 2.

Table 4.5 Example of a multiple prototype labeling algorithm

	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$
$\mathbf{e}_1$	0.57	0.10	0.13	0.20
$\mathbf{e}_2$	0.15	0.10	0.15	0.60
$\mathbf{e}_3$	0.05	0.40	0.40	0.15
	↓	↓	↓	↓
	$\mathbf{H(p_1)=e_1}$	$\mathbf{H(p_2)=e_3}$	$\mathbf{H(p_3)=e_3}$	$\mathbf{H(p_4)=e_2}$

C. Sequential hard c-means (SHCM)

The oldest model that can properly be identified as a CL model is probably *sequential hard c-means* (SHCM). As we shall see, the update rule of MacQueen's (1967) SHCM algorithm is very similar to the more recent and popular LVQ designs. MacQueen attempted to partition feature space $\Re^p$ into c subregions, say $(S_1,\ldots,S_c)$, in such a way as to minimize the functional

$$J_M(\hat{\mathbf{V}}) = \sum_{i=1}^{c}\left\{\int\cdots\int\|\mathbf{x}-\hat{\mathbf{v}}_i\|^2 df(\mathbf{x})\right\} \quad , \qquad (4.14)$$

where f is an (unknown) *probability density function* (pdf), $\hat{\mathbf{V}} = (\hat{\mathbf{v}}_1,\ldots,\hat{\mathbf{v}}_c) \in \Re^p$, $\hat{\mathbf{v}}_i$ is the (conditional) mean of $\mathbf{x}$ estimated by the pdf f_i obtained by restricting f to S_i, normalized by the prior probability π_i of class i, i.e., $f_i(\mathbf{x}) = f(\mathbf{x})|_{S_i}/\pi_i$.

In MacQueen's SHCM algorithm to approximately minimize J_M, the weight vectors are initialized with the first c samples in the data set X. In other words, $\mathbf{v}_{r,0} = \mathbf{x}_r$, r=1,..,c. Let $q_{r,0}=1$ for r=1,..,c ($q_{r,t}$ represents the number of samples that have so far been used to update $\mathbf{v}_{r,t}$). MacQueen's process terminates when all the samples have been used once (i.e., take $\mathbf{V} = \hat{\mathbf{V}}$ after one pass through X). For this implementation, we need only indices k and i in Table 4.4. Suppose $\mathbf{x}_k$ is the current input and that $\mathbf{v}_{i,k-1}$ is closest to it, as in

Figure 4.11, $i = \underbrace{\arg\min}_{r}\left\{\left\|\mathbf{x}_k - \mathbf{v}_{r,k-1}\right\|\right\}$. MacQueen's algorithm updates the $\mathbf{v}_r$'s as follows :

$$\mathbf{v}_{i,k} = (\mathbf{v}_{i,k-1}\, q_{i,k-1} + \mathbf{x}_k)/(q_{i,k-1} + 1); \qquad (4.15a)$$
$$q_{i,k} = q_{i,k-1} + 1 \quad ; \qquad (4.15b)$$
$$\mathbf{v}_{r,k} = \mathbf{v}_{r,k-1} \text{ for } r \neq i, \quad ; \qquad (4.15c)$$
$$q_{r,k} = q_{r,k-1} \text{ for } r \neq i. \quad . \qquad (4.15d)$$

Other versions of SHCM pass through the data set many times (Forgy, 1965). Rearranging (4.15a), we can rewrite Macqueen's update equation for the winning prototype as

$$\mathbf{v}_{i,k} = \mathbf{v}_{i,k-1} + (\mathbf{x}_k - \mathbf{v}_{i,k-1}) / q_{i,k} \quad . \qquad (4.16)$$

Equation (4.16) takes the general form shown in equations (4.12b) and (4.12c) by setting $\alpha_{i,k}^{SHCM} = 1 / q_{i,k}$ in (4.16).

If crisp clusters are desired, the sample points can then be labeled using HCM necessary condition (the 1-np rule) in equation (2.6a). This usually produces a hard c-partition U_{SHCM}. Since the $\{\mathbf{v}_j = \hat{\mathbf{v}}_j\}$ are conditional means, the partition obtained this way may not be desirable from the point of view of clustering. Moreover, this method does not eliminate the possibility of slow but indefinite oscillation of the centroids (limit cycles). Nonetheless, this is a historically important and still popular method of prototype generation, and the terminal prototypes *can* be used for nearest prototype classifier design.

D. Learning vector quantization (LVQ)

The learning rate distribution for LVQ that is used in equation (4.12b) of our CL template is well known:

$$\alpha_{jk,t}^{LVQ} = \begin{cases} \alpha_t & ; \quad i = \underbrace{\arg\min}_{j}\left\{\left\|\mathbf{x}_k - \mathbf{v}_{j,t-1}\right\|\right\} \\ 0 & , \quad j = 1,2,\ldots c \ ; \ j \neq i \end{cases} \quad . \qquad (4.17)$$

Equation (4.17) shows that, like SHCM, this form of LVQ is a winner take all strategy - that is, the update neighborhood is just a single point. In (4.17) learning rate α_t is usually: (i) independent of i and k; (ii), initialized to some value in (0, 1); and (iii), decreased nonlinearly with t, usually $\alpha_t \propto (1/t)$. There are some differences between our version of unsupervised LVQ and MacQueen's algorithm: (i) in LVQ sample points are used repeatedly until termination is achieved, while in MacQueen's method, sample points are used only once; (ii) in MacQueen's algorithm $\alpha_{i,t}^{SHCM}$ is inversely proportional to the number of points found closest to $\mathbf{v}_{i,t-1}$, so it is possible to have $\alpha_{i,t_1}^{SHCM} < \alpha_{i,t_2}^{SHCM}$ when $t_1 > t_2$.

So much has been written about supervised and unsupervised versions of LVQ (there are many variations to the form embodied by using (4.17) in (4.12c)) that our discussion of it here will be limited to several examples that compare it to several soft generalizations of it. But before we leave this subsection, we point out that LVQ is a special case of a more general model due to Kohonen (1989) called the *self-organizing feature map* (SOFM), which will put in an appearance in Example 4.26.

We give a very brief description of the SOFM scheme, again using t to stand for iterate number (or time). In SOFM each prototype $\mathbf{v}_{j,t} \in \Re^p$ is associated with a *display node*, say $\mathbf{d}_{j,t} \in \Re^q$. Usually q = 1 or 2, but the display "space" could have more dimensions, and it is not really a space, but a set or lattice D of integers (addresses) in $\Re^q$. The purpose of the display set is to establish a topological neighborhood for the address or index associated with each prototype vector, so there are exactly as many cells in the display space as there are prototypes. For example, if you have 100 prototypes for the Iris data, then $\mathbf{V} = \{\mathbf{v}_1, \ldots \mathbf{v}_{100}\} \subset \Re^4$, so a natural display set for these prototypes would be a linear array, the integers $D = \{1, \ldots, 100\}$. On the other hand, if the prototypes had spatial identities in two dimensions, they might be doubly indexed, as, for example $\mathbf{V} = \{\mathbf{v}_{11}, \ldots \mathbf{v}_{10,10}\} \subset \Re^4$, and then a natural display set would be the 100 pairs of integers $D = \{(1,1), \ldots, (10,10)\}$ arranged in a square lattice. Topological neighbors in D are neighboring *addresses* - the cells in D are only indices, and do not possess numerical features (like pixels in images in Chapters 4 and 5, for example, which contain at least intensities at their addresses). In the SOFM scheme, each address is associated with a unique prototype in $\Re^p$.

In SOFM the winning vector $\mathbf{v}_{i,t}$ that best matches (usually, but not necessarily, in the sense of minimum Euclidean distance) an input vector $\mathbf{x}_k$ is found. Next, a topological (spatial) *update neighborhood* $\mathcal{N}(\mathbf{d}_{i,t}) \subset D$ centered at $\mathbf{d}_{i,t} \in D$ is defined in D, and the winner node neighbors are located *in D*. This means that you must define what a neighborhood *is* in D, and this requires two concepts - shape and size. For linear arrays, the shape of $\mathcal{N}(\mathbf{d}_{i,t})$ is usually adjacent indices to the left and right of D out to a specified radius; for 2D display sets, it could mean the 4-connected neighbors of $\mathbf{d}_{i,t}$ diagonally or parallel to the axes of D, or the 8-connected neighbors of $\mathbf{d}_{i,t}$ that surround it in D, etc. Along with the shape of $\mathcal{N}(\mathbf{d}_{i,t})$ there must be a concept of order or size, usually defined through its radius, which will decrease with time (iteration).

Finally, $\mathbf{v}_{i,t}$ and other prototype vectors in the inverse image $[\mathcal{N}(\mathbf{d}_{i,t})]^{-1}$ of the spatial neighborhood $\mathcal{N}(\mathbf{d}_{i,t})$ are updated using equation (4.12c). We mentioned that the update neighborhood could be imbedded into the learning rate schedule (the $\{\alpha_{jk,t}\}$) in connection with equation (4.11), and SOFM is an example of the need to do this. For the current situation, we accomplish this by setting $\alpha_{jk,t} = 0$ for all $\mathbf{v}_{j,t} \notin [\mathcal{N}(\mathbf{d}_{i,t})]^{-1}$, and use whatever learning rates are defined at this set of subscripts to update the prototypes in the update neighborhood. The usual way to operate SOFM is to decrease both the values of the learning rates and size of the update neighborhood over time. When the update neighborhood is reduced to the winner alone $(\mathbf{v}_{i,t} = [\mathcal{N}(\mathbf{d}_{i,t})]^{-1})$, SOFM becomes the LVQ algorithm. The relationship between and manipulation of $\mathbf{V}$ and $\mathcal{N}(\mathbf{d}_{i,t})$ can a pretty difficult concept to grasp for first time readers about SOFM; please refer to Kohonen (1989) for amplification.

E. Some soft versions of LVQ

SHCM and LVQ attempt to minimize objective functions that place all of their emphasis on the winning prototype for each data point. However, structural information due to data point $\mathbf{x}$ is carried by *all* c of the distances $\left\{\|\mathbf{x}-\mathbf{v}_i\|\right\}$. Many authors have suggested modifications to winner take all models that update all c prototypes during each updating epoch, thereby eliminating the need to define an update neighborhood. We will discuss three CL models of this type, GLVQ-F (this subsection), SCS (subsection 4.3.G) and FLVQ (subsection 4.3.H). The model underlying GLVQ-F contains LVQ as a subcase and is discussed extensively in Karayiannis et al. (1996). GLVQ-F is based on minimizing the functional

$$J_{GLVQ-F}(\mathbf{x}_k;\mathbf{V}) = \sum_{r=1}^{c} u_r \left\|\mathbf{x}_k - \mathbf{v}_r\right\|^2$$

$$= \sum_{r=1}^{c}\left(\sum_{j=1}^{c}\left(\frac{\left\|\mathbf{x}_k - \mathbf{v}_r\right\|^{2/(m-1)}}{\left\|\mathbf{x}_k - \mathbf{v}_j\right\|^{2/(m-1)}}\right)\right)^{-1} \left\|\mathbf{x}_k - \mathbf{v}_r\right\|^2, \ m>1. \qquad (4.18)$$

In (4.18) the vector $\mathbf{u} = (u_1, u_2, \ldots, u_c)^T \in N_{fc}$ is a fuzzy label vector whose entries take the form of FCM necessary condition (2.7a). The real number m > 1 in (4.18) is the same fuzziness parameter that appears in FCM and PCM. The value of m affects the quality of representation by the terminal prototypes it finds. And m also controls the speed of termination of the GLVQ-F algorithm, which is just steepest descent applied to J_{GLVQ-F}. The GLVQ-F update rule for the prototypes **V** at iterate t in the special (and simple) case m=2 gives the following learning rate distribution for use in equation (4.12b) :

$$\alpha_{ik,t}^{GLVQ\text{-}F(m=2)} = 2c\alpha_t \underbrace{\left(\sum_{r=1}^{c} \left(\frac{\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|^2}{\|\mathbf{x}_k - \mathbf{v}_{r,t-1}\|^2} \right) \right)^{-2}}_{u_{ik,t-1}^2} , \quad 1 \le i \le c \ . \tag{4.19}$$

Equation (4.19) has the same singularity condition as FCM in its denominator. When no $\|\mathbf{x}_k - \mathbf{v}_{r,t-1}\| = 0$, (4,19) produces a learning rate for each value of i, so all c prototypes are updated at each input. As in (4.17), α_t in (4.19) - now one *factor* of the learning rates $\{\alpha_{ik,t}\}$ - is usually proportional to 1/t, and the constant (2c) is absorbed in it without loss. Limiting properties of GLVQ-F are : (i) as m approaches infinity, all c prototypes receive equal updates and the $\mathbf{v}_i$'s all converge to the grand mean $\bar{\mathbf{v}}$ of the data; whereas (ii) as m approaches 1 from above, only the winner is updated, and GLVQ-F reverts to LVQ. Finally, we mention that the winning prototype in GLVQ-F for m=2 receives the largest (fraction) of $\alpha_{ik,t}$ at iterate t; and that other prototypes receive a share that is *inversely* proportional to their distance from the input. The GLVQ-F learning rates satisfy the additional constraint $\sum_{i=1}^{c} \alpha_{ik,t} \le 1$.

F. Case Study : LVQ and GLVQ-F 1-nmp designs

This subsection abstracts part of an example discussed in Bezdek et al. (1998b). Here Anderson's (1935) Iris data is used to illustrate 1-nmp classifier design with prototypes found by LVQ and GLVQ-F. Figure 4.12 scatterplots the third and fourth features of Iris (hereafter called Iris_{34}) and the subsample means (listed in Table 4.6) for each of the three classes. Class 1 is well separated from classes 2 and 3 in these two dimensions; classes 2 and 3 show some overlap in the central area of the figure, and this region contains the vectors that are usually mislabeled by nearest prototype designs. The dashed boundaries indicate the physically labeled 2D cluster boundaries. Thus, $\hat{c}$ = 3 in the terminology of Definition 4.2.

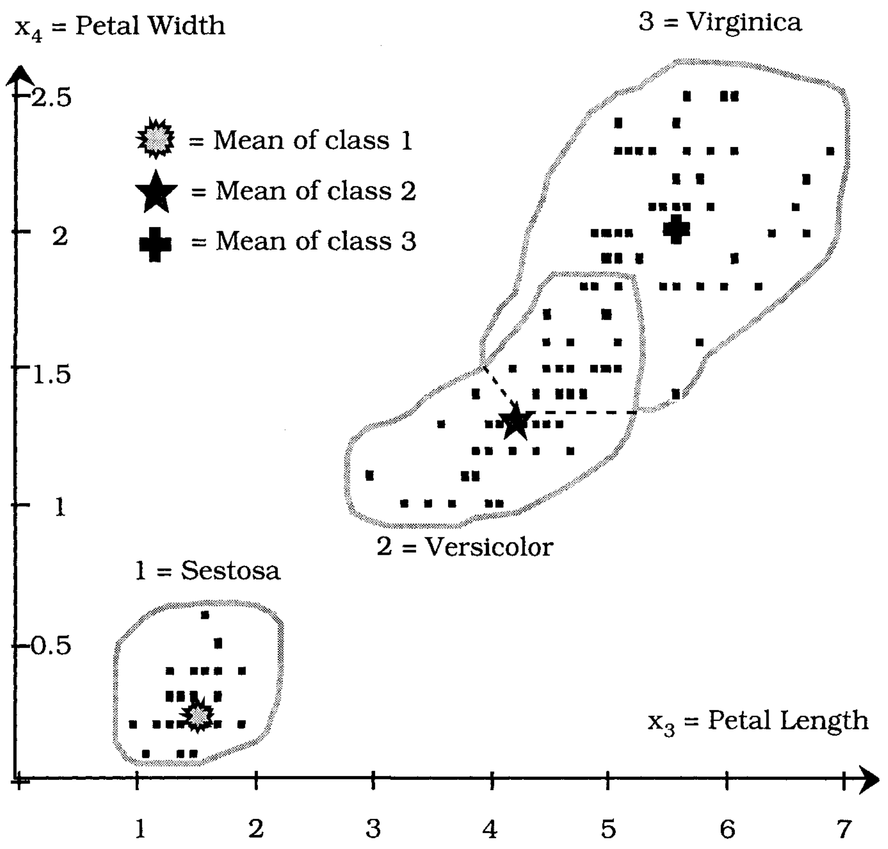


Figure 4.12 The Iris data : {(feature 3, feature 4)} = Iris$_{34}$

The resubstitution error rate for the supervised 1-np design that uses the class means (listed in Table 4.6 and shown on Figure 4.12) as single prototypes is 11 errors in 150 submissions using the Euclidean norm, i.e., $E_{D_{\bar{\mathbf{V}},E,\delta_2}}$ (Iris|Iris) = 7.33% (see the confusion matrix for this case in Table 4.14).

Table 4.6 Labeled sample (mean) prototypes $\bar{\mathbf{V}}$ in $\Re^4$ for Iris

Symbol	Name	x_1	x_2	x_3	x_4
✹	$\bar{\mathbf{v}}_1$	5.01	3.43	1.46	0.25
★	$\bar{\mathbf{v}}_2$	5.94	2.77	4.26	1.33
✚	$\bar{\mathbf{v}}_3$	6.59	2.97	5.55	2.03

The "hyberbox diagonal" method is used to generate an initial set of c prototypes $\mathbf{V}_0$ for this example. To build the hyperbox compute

$$\text{Minimum of feature j : } m_j = \underbrace{\min}_{k}\{x_{jk}\}: \; j = 1,2,\ldots,p \; ; \tag{4.20a}$$

$$\text{Maximum of feature j : } M_j = \underbrace{\max}_{k}\{x_{jk}\}: \; j = 1,2,\ldots,p. \tag{4.20b}$$

The set $hb(\mathbf{m},\mathbf{M}) = [m_1, M_1] \times \ldots \times [m_p, M_p]$ is a *hyperbox* in $\Re^p$. The main diagonal of $hb(\mathbf{m},\mathbf{M})$ connects $\mathbf{m}$ and $\mathbf{M}$ with the line segment $\{\mathbf{m} + \alpha(\mathbf{M} - \mathbf{m}); 0 \le \alpha \le 1\}$. Initial prototypes for LVQ and GLVQ-F in this example were:

$$\mathbf{v}_{i,0} = \mathbf{m} + \left(\frac{i-1}{c-1}\right)(\mathbf{M} - \mathbf{m}) \; ; i = 1, 2, \ldots, c \quad . \tag{4.21}$$

Thus, $\mathbf{v}_{1,0} = \mathbf{m} = (m_1, m_2, \ldots, m_p)^T$; $\mathbf{v}_{c,0} = \mathbf{M} = (M_1, M_2, \ldots, M_p)^T$; and the remaining (c-2) initial prototypes are uniformly distributed along the diagonal of $hb(\mathbf{m},\mathbf{M})$. A useful variation of this initialization strategy is to choose c points randomly from the diagonal $\{\mathbf{m} + \alpha(\mathbf{M} - \mathbf{m}); 0 \le \alpha \le 1\}$. For the present case, Table 4.7 shows the initial prototypes produced by uniform draws as in (4.21) with the Iris data at c = 6.

Table 4.7 Initial prototypes for Iris at c = 6 computed with (4.21)

$$\mathbf{v}_{1,0} = (\,4.30\ 2.00\ 1.00\ 0.10\,) = \mathbf{m}$$
$$\mathbf{v}_{2,0} = (\,5.02\ 2.48\ 2.18\ 0.58\,)$$
$$\mathbf{v}_{3,0} = (\,5.74\ 2.96\ 3.36\ 1.06\,)$$
$$\mathbf{v}_{4,0} = (\,6.46\ 3.44\ 4.54\ 1.54\,)$$
$$\mathbf{v}_{5,0} = (\,7.18\ 3.92\ 5.72\ 2.02\,)$$
$$\mathbf{v}_{6,0} = (\,7.90\ 4.40\ 6.90\ 2.50\,) = \mathbf{M}$$

The Euclidean norm was used in (4.12a), and the number of prototypes generated ranged from c = $\hat{c}$ = 3 to c = 30. The termination threshold ε had one of the three values ε= 0.1, 0.01 and 0.001. The primary termination criterion that was compared to ε was the 1-norm between successive estimates of the c prototypes, i.e., $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|_1 = \sum_{r=1}^{c}\sum_{j=1}^{p}\left|v_{jr,t} - v_{jr,t-1}\right|$; if this failed to stop an algorithm, secondary termination occurred at the iterate limit T = 1000. The initial learning rate was α_0 = 0.4 and α was decreased *linearly*, viz., $\alpha_t = \alpha_0((T-t)/T)$ for both algorithms. For the results displayed, (4.19) was used for GLVQ-F.

Samples were drawn randomly from X_{tr} = Iris without replacement. One iteration corresponds to one pass through Iris. Each algorithm was run 5 times for each case discussed to see how different input sequences affected the terminal prototypes. For the less stringent termination criteria (ε = 0.1 and 0.01), different terminal prototypes were sometimes obtained on different runs. For ε = 0.001, this effect was nearly (but not always) eliminated. Most of the runs using ε = 0.001 were completed in less than 300 iterations through Iris.

Unsupervised nearest prototype designs for Iris that seek $\hat{c}$= 3 prototypes report resubstitution errors ranging from 5 to 20. Table 4.8 exhibits the terminal prototypes found by each algorithm for c = 6, as well as the resultant 1-nmp error rates they produce when used in (4.7) on all of Iris. Each of the three physical clusters is represented by two prototypes for both LVQ and GLVQ-F, and the overall error rate produced by these two classifiers is 9.33% - 14 mistakes, not really much better than any unsupervised design at $\hat{c}$= 3, and not as good as the supervised sample means design.

Table 4.8 Typical prototypes, confusion matrices and 1-nmp resubstitution error rates for c = 6 prototypes (Iris data)

LVQ labels	LVQ prototypes	GLVQ-F labels	GLVQ-F (m=2) prototypes
1	4.69 3.12 1.39 0.20	1	4.75 3.15 1.43 0.20
1	5.23 3.65 1.50 0.28	1	5.24 3.69 1.50 0.27
2	5.52 2.61 3.90 1.20	2	5.60 2.65 4.04 1.24
2	6.21 2.84 4.75 1.57	2	6.18 2.87 4.73 1.56
3	6.53 3.06 5.49 2.18	3	6.54 3.05 5.47 2.11
3	7.47 3.12 6.31 2.02	3	7.44 3.07 6.27 2.05
	$C=\begin{pmatrix}50 & 0 & 0\\ 0 & 50 & 0\\ 0 & 14 & 36\end{pmatrix}$		$C=\begin{pmatrix}50 & 0 & 0\\ 0 & 50 & 0\\ 0 & 14 & 36\end{pmatrix}$
	Error rate = 9.33 %		Error rate = 9.33 %

The third and fourth features of the prototypes in Table 4.8 are plotted in Figure 4.13 against a background created by roughly estimating the convex hull of each physical class in these two dimensions by eye. Some of the prototypes are hard to see because their coordinates are very close in these two dimensions. The LVQ and GLVQ-F prototypes that seem to lie on the boundary between classes 2 and 3 are highlighted by enclosing these points with the jagged star ✸. These prototypes are the ones that incur most of the misclassifications that are committed by the LVQ and GLVQ-F 1-nmp classifiers.

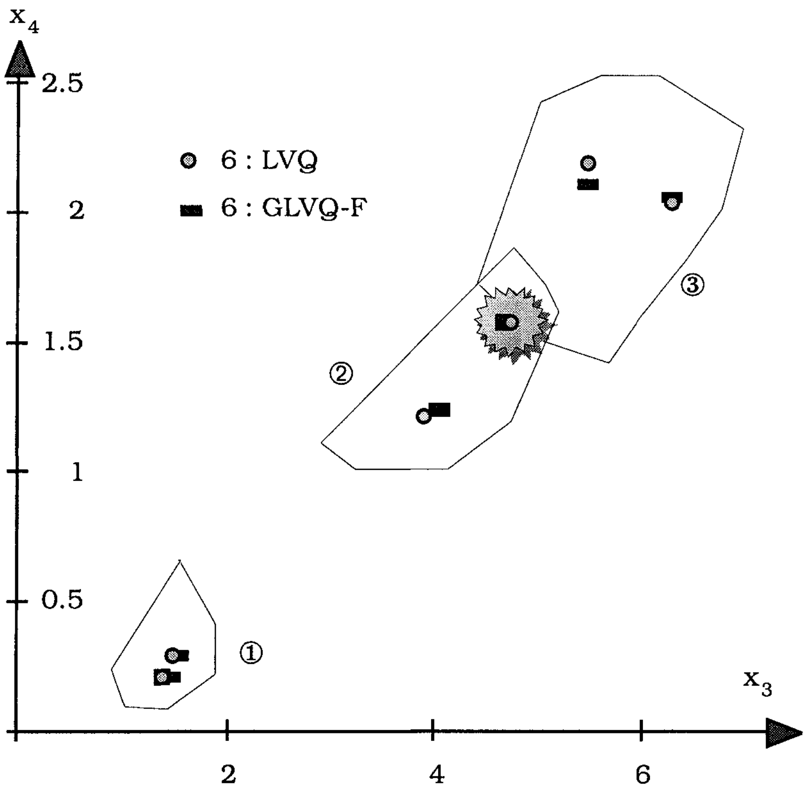


Figure 4.13 Terminal prototypes in Iris$_{34}$ at c = 6

Table 4.9 lists the same information as Table 4.8 for typical runs made at c = 7. There is a sharp drop in the error rate for both the LVQ and GLVQ-F 1-nmp designs. Be careful to note that the seventh prototype is not "added" to the previous six; rather, seven new prototypes are found by each algorithm. The error rates in Table 4.9 are very low for designs that do not use the labels during training. Note that LVQ and GLVQ-F continue to use 2 prototypes for each of classes 1 and 2, and add a third representative for class 3 at c = 7. Thus, neither LVQ nor GLVQ-F provides an efficient representation of the data because only one prototype is needed to represent the 50 class 1 points with no resubstitution errors. This point is brought out in Bezdek et al. (1998b), where the so-called "dog-rabbit" prototype generation algorithm is used to achieve this somewhat more desirable result.

Table 4.9 Typical prototypes, confusion matrices and 1-nmp resubstitution error rates for c = 7 prototypes (Iris data)

LVQ Labels	LVQ prototypes	GLVQ-F Labels	GLVQ-F (m=2) prototypes
1	4.68 3.11 1.39 0.20	1	4.74 3.15 1.43 0.20
1	5.23 3.65 1.50 0.28	1	5.24 3.69 1.50 0.27
2	5.53 2.62 3.93 1.21	2	5.57 2.61 3.96 1.21
2	6.42 2.89 4.59 1.43	2	6.26 2.92 4.54 1.43
3	6.57 3.09 5.52 2.18	3	6.62 3.09 5.56 2.16
3	7.47 3.12 6.31 2.02	3	7.50 3.05 6.35 2.06
3	5.99 2.75 5.02 1.79	3	6.04 2.79 4.95 1.76
	$C=\begin{pmatrix}50 & 0 & 0\\ 0 & 47 & 3\\ 0 & 1 & 49\end{pmatrix}$		$C=\begin{pmatrix}50 & 0 & 0\\ 0 & 46 & 4\\ 0 & 1 & 49\end{pmatrix}$
	Error rate = 2.66 %		Error rate = 3.33 %

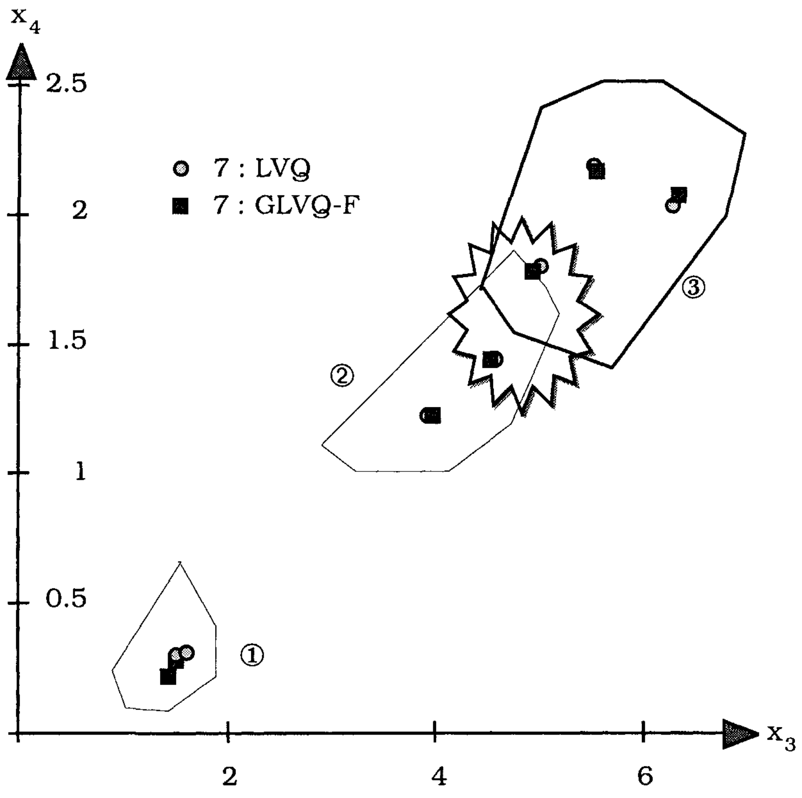


Figure 4.14 Terminal prototypes in Iris$_{34}$ at c = 7

Figure 4.14 shows that the crucial "boundary" prototypes from LVQ and GLVQ-F in the c = 6 case have roughly "divided" into two sets of new prototypes, enclosed again by the jagged star. These two pairs of prototypes have moved away from the apparent boundary of the lower left part of the convex hull of class 3. Both new pairs move further into the convex hulls of their respective classes.

When these two CL algorithms are instructed to seek c = 8 prototypes, the resubstitution error rate for both designs typically remains at 2.66%, and at c = 9 the results are quite similar. These results suggest that the replacement of Iris with 8 or 9 prototypes found by either LVQ or GLVQ-F results in a 1-nmp design that is quite superior (as measured by the resubstitution error rate) to the labeled 1-np design based on the $\hat{c} = 3$ subsample means $\overline{\mathbf{V}}$. It is reasonable to assume that this trend would also hold for apparent error rates computed with test data reserved from Iris - i.e., that the 1-nmp designs would generalize better than classifiers based on 1-np designs - reasonable, but certainly not guaranteed.

How few prototypes are needed by the 1-nmp rule to achieve good results? And conversely, at what point does prototype representation become counter-productive? Table 4.10 shows the best case results (as number of resubstitution errors) reported in Bezdek et al. (1998b) using each algorithm at various values of c.

Table 4.10 Best case resubstitution errors

c →	3	4	5	6	7	8	9	15	30
LVQ	17	24	14	14	3	4	4	4	4
GLVQ-F	16	20	19	14	5	3	4	4	4

Observe that on passing from $\hat{c} = 3$ to c = 4, even the best case error rate *increased*, followed by a decrease on passing from c = 4 to c = 5. Table 4.10 shows that the Iris data can be fairly well represented in the sense of minimal resubstitution error by 7 or 8 labeled prototypes (see Kuncheva and Bezdek (1998) for a non fuzzy design that yields zero resubstitution errors using 12 prototypes). At the other extreme, increasing c past c = 7 has little effect on the best case results. Taken together, these observations suggest that Iris (and more generally, any labeled data set) has upper and lower bounds in terms of high quality representation by multiple prototypes for classifier design. There seems to be little hope, however, of discovering this on a better than case-by-case basis.

Finally, some comments on the sensitivity of each CL model to changes in its control parameters. Bezdek et al. (1998b) did not experiment with changes in m for GLVQ-F. Certainly this parameter affects terminal prototypes. However, we doubt that small changes in m will cause radical changes in the results given above. The initial learning rate α_0 was varied from 0.4 to 0.6 in both LVQ and GLVQ-F without noticeable changes in typical results.

G. The soft competition scheme (SCS)

Yair et al. (1992) proposed two vector quantization models, a *stochastic relaxation scheme* (SRS) and a *soft competition scheme* (SCS). Like GLVQ-F, algorithms for these two models eliminate the need to define an update neighborhood by extending the update to all c nodes; and they use learning rates that are functions of the c distances $\left\{\|\mathbf{x}-\mathbf{v}_i\|\right\}$. Our discussion here is limited to the SCS model and algorithm.

SCS is a deterministic algorithm (the algorithm is deterministic because its steps are not stochastically controlled, but it does use probabilities as part of the learning rates). In SCS all c prototypes are simultaneously updated by a scheme which directs them - like LVQ - towards the current training vector. The step size of each update is scaled by the probability of that prototype being the winner. At time (iterate) t, the probability of the i-th prototype winning is defined as

$$p_{ik,t} = \frac{e^{-\beta_t \|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|^2}}{\sum_{j=1}^{c} e^{-\beta_t \|\mathbf{x}_k - \mathbf{v}_{j,t-1}\|^2}} \quad , \tag{4.22}$$

where $\lim_{t\to\infty}\{\beta_t\} = \infty$. The probability $p_{ik,t}$ is one factor in the SCS update equation. The choice for β_t specified by Yair et al. is $\beta_t = \hat{\gamma}^{t/c}/T_0$. T_0 is regarded as an initial "temperature", and $\hat{\gamma}$ is a constant which Yair et al. stipulate should be greater than 1. The quantity $(1/\beta_t)$ is regarded as the temperature T at time t, so as $t \to \infty$, $\beta_t \to \infty$, and $T \to 0$. Hence, this procedure is analogous to simulated annealing.

Next, let $n_{i,t} = n_{i,t-1} + p_{ik,t}$ (*approximately* the total number of times that $\mathbf{v}_i$ has been updated). This parameter is reset to 1 whenever iteration counter t is a perfect square. Yair et al. use this to define the other factor of their learning rates as

$$\eta_{ik,t} = \left(\frac{1}{n_{i,t}}\right) = \left(\frac{1}{n_{i,t-1} + p_{ik,t}}\right) \quad . \tag{4.23}$$

The overall learning rate for SCS that is substituted into (4.12c) is the product of these two factors,

$$\alpha_{ik,t}^{SCS} = \eta_{ik,t} \cdot p_{ik,t} \quad . \tag{4.24}$$

Table 4.11 gives the implementation of SCS that was used by Bezdek and Pal (1995) for the results presented in Example 4.3.

Table 4.11 The SCS algorithm (Yair et al., 1992)

Store	(Un)labeled Object Data $X_{tr} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$
	$U_{tr} \in M_{hcn}$ = Labels of vectors in X_{tr} (if available)
Pick	☺ number of nodes : $1 < c < n$
	☺ max. # of iterations : T
	☺ distance measure : $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|$
	☺ termination measure : $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|$
	☺ termination criterion : ε
	☺ $\hat{\gamma} > 1$
	☺ T_0 = initial temperature
Get	☺ initial prototypes: $\mathbf{V}_0 \in \Re^{cp}$
Do	See pseudocode below

```
t ← 1; $E_0$ = high value
DO UNTIL (t >T or $E_{t-1} \le \varepsilon$)
$\beta_t = \hat{\gamma}^{t/c} / T_0$
For k = 1 to n
    $\mathbf{x} \in X,\ \mathbf{x}_k \leftarrow \mathbf{x},\ X \leftarrow X - \{\mathbf{x}_k\}$
    For i = 1 to c
        $p_{ik,t} = e^{-\beta_t \|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|^2} \Big/ \sum_{j=1}^{c} e^{-\beta_t \|\mathbf{x}_k - \mathbf{v}_{j,t-1}\|^2}$
        If (t = a perfect square) $n_{i,t} = 1$
        Else $n_{i,t} = n_{i,t-1} + p_{ik,t}$
        $\eta_{ik,t} = (1 / n_{i,t})$
        $\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \eta_{ik,t} \cdot p_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1})$
    Next i
Next k
$t \leftarrow t + 1$
END UNTIL
$\mathbf{V} \leftarrow \mathbf{V}_{t-1}$
```

We call attention to the handling of $n_{i,t}$ in Table 4.11, which necessitates further modifications to the standard CL model set out in Table 4.4, but reports what Yair et al. used in their original paper. Since SCS is an unsupervised method, optional blocks B and C of Table 4.4 are applicable to this scheme too. SCS starts with a low value of β_t (i.e., with approximately uniform $\{p_{ik,t}\}$), and then β_t slowly increases with time. As a result, at the beginning of the procedure no prototype is strongly attracted to a particular class. With time (i.e., as the number of iterations increases) prototypes become more strongly separated from each other as $p_{ik,t}$ begins to peak around the Euclidean winner, but at the same time $\eta_{ik,t} \to 0$. Thus in the limit (as iterate t goes to infinity) SCS behaves like a winner-take all (LVQ type) competition.

The c numbers $\{p_{ik,t}\}$ are probabilities, so they satisfy $0 \le p_{ik,t} \le 1$ and $\sum_{i=1}^{c} p_{ik,t} = 1$. Consequently, $\mathbf{p}_t(\mathbf{x}_k) = (p_{1k,t}, p_{2k,t}, \ldots, p_{ck,t})^T$ is a probabilistic label vector for $\mathbf{x}_k$, $\mathbf{p}_t(\mathbf{x}_k) \in N_{fc}$. Since $\eta_{ik,t} = \left(\frac{1}{n_{i,t}}\right) \le 1$, the sum of the learning rates for a fixed input vector $\mathbf{x}_k$ at any iterate t satisfies the constraint $0 < \sum_{i=1}^{c} \alpha_{ik,t}^{SCS} \le 1$. Bezdek and Pal (1995) showed that there is a strong relationship between SCS and mixtures of normal distributions as discussed in Section 2.2.C. Bezdek and Pal made two simplifying assumptions about the mixture of normals obtained by substituting (2.18a) into (2.17). For each class i, $1 \le i \le c$, they assumed that

$$\pi_i = 1/c \qquad ; \text{and} \tag{4.25a}$$

$$\Sigma_i = \sigma^2 I \qquad . \tag{4.25b}$$

In other words, all classes are equally likely and all classes have a population covariance matrix which is a scalar multiple of the identity. Then $\Sigma_i^{-1} = \frac{1}{\sigma^2} I$ and $\sqrt{\det(\Sigma_i)} = \sigma$ for every class, so the Mahalanobis norm becomes a multiple of the Euclidean norm, $\|\mathbf{x} - \boldsymbol{\mu}_i\|^2_{\Sigma_i^{-1}} = \frac{1}{\sigma^2}\|\mathbf{x} - \boldsymbol{\mu}_i\|^2$. For this special case Bayes rule at (2.19) takes the form

$$\pi(i|\mathbf{x}) = \frac{\left(\frac{1}{c}\right)e^{-\left(\frac{1}{2\sigma^2}\|\mathbf{x}-\mu_i\|^2\right)} \Big/ \left((2\pi)^{p/2}\sigma\right)}{\left[\sum_{j=1}^{c}\left(\frac{1}{c}\right) e^{-\left(\frac{1}{2\sigma^2}\|\mathbf{x}-\mu_j\|^2\right)}\right] \Big/ \left((2\pi)^{p/2}\sigma\right)}$$

$$= \frac{e^{-\left(\frac{1}{2\sigma^2}\|\mathbf{x}-\mu_i\|^2\right)}}{\left[\sum_{j=1}^{c} e^{-\left(\frac{1}{2\sigma^2}\|\mathbf{x}-\mu_j\|^2\right)}\right]} \quad . \tag{4.26}$$

For a given $\mathbf{x}_k$ this becomes

$$\pi(i|\mathbf{x}_k) = e^{-\left(\frac{1}{2\sigma^2}\|\mathbf{x}_k-\mu_i\|^2\right)} \Big/ \sum_{j=1}^{c} e^{-\left(\frac{1}{2\sigma^2}\|\mathbf{x}_k-\mu_j\|^2\right)} \quad . \tag{4.27}$$

If we define $\beta_t = 1/2\sigma^2$ and $\mathbf{v}_{i,t-1} = \boldsymbol{\mu}_i$ for i = 1 to c, then $p_{ik,t}$ and $\pi(i|\mathbf{x}_k)$ are identical. Thus the component $p_{ik,t}$ of the SCS learning rate used by Yair et al. can be interpreted as an estimate of the posterior probability of $\mathbf{x}_k$ being from class i under the assumptions in Section 2.2.C and (4.25). However, $\beta_t = 1/2\sigma^2$ does not ensure $\lim_{t\to\infty}\{\beta_t\} = \infty$. To achieve this Yair et al. use t in the definition for β_t, i.e., $\beta_t = (\hat{\gamma}^{t/c} / T_0)$. Thus, at time t we take $\sigma_t^2 = (T_0\,\hat{\gamma}^{-t/c} / 2)$. Then $p_{ik,t}$ and $\pi(i|\mathbf{x}_k)$ are still identical, and $\beta_t = 1/2\sigma_t^2 \Rightarrow \lim_{t\to\infty}\{\beta_t\} = \infty$.

In summary, $p_{ik,t}$ can be interpreted as the posterior probability that $\mathbf{x}_k$ is from class i when all classes are equally likely, and class i is modeled as a p-variate normal distribution with parameters $\left(\boldsymbol{\mu}_{i,t} = \mathbf{v}_{i,t},\ \Sigma_{i,t} = (T_0\,\hat{\gamma}^{-t/c} / 2)I\right)$. Example 4.3 will compare SCS to FLVQ, the next CL model we discuss.

H. Fuzzy learning vector quantization (FLVQ)

A possible connection between batch FCM and sequential LVQ was first discussed by Huntsberger and Ajjimarangsee (1990), who suggested fuzzification of LVQ by replacing the learning rates $\{\alpha_{ik,t}\}$ in (4.12c) with the fuzzy membership values $\{u_{ik,t}\}$ computed with FCM formula (2.7a). While this approach was innovative, it was to some extent unmotivated. Moreover, their method still required choosing m, and it seemed to improperly mix the objectives of LVQ (vector quantization) and FCM (clustering).

Tsao et al. (1993) proposed a batch prototype generator model that required the use of fuzzy partitions that was initially called a *fuzzy Kohonen clustering network* (FKCN). Like fuzzy ISODATA, this initial name seemed inappropriate, so the model and algorithms for it have subsequently become known as FLVQ (Bezdek and Pal, 1995). FLVQ has three objectives: (i) to overcome the two problems we identified for LVQ (which nodes to update and how to use the non-winner prototypes in the determination of learning rates); (ii) to circumvent (to some extent) the problem of how to choose m for FCM-AO; and (iii) to provide a substantial link between the batch c-means and sequential LVQ families of prototype generators.

As noted in Section 2.3, the choice of m for the FCM model is very important. When m is small (close to 1), (2.7a) tends to produce almost crisp label vectors. If prototype updates in equation (4.12c) use learning rates based on (2.7a), and u_{ik} is close to 1, the update for node i may be very large compared to the other updates because of the column constraint $\sum_{i=1}^{c} u_{ik} = 1$. If, additionally, the current prototypes have an unfavorable geometry compared to the central tendencies of clusters in the data, some prototypes may move rapidly towards a cluster, while others may move but little. This effect is illustrated in Figure 4.15 for the data set $X = X_1 \cup X_2$.

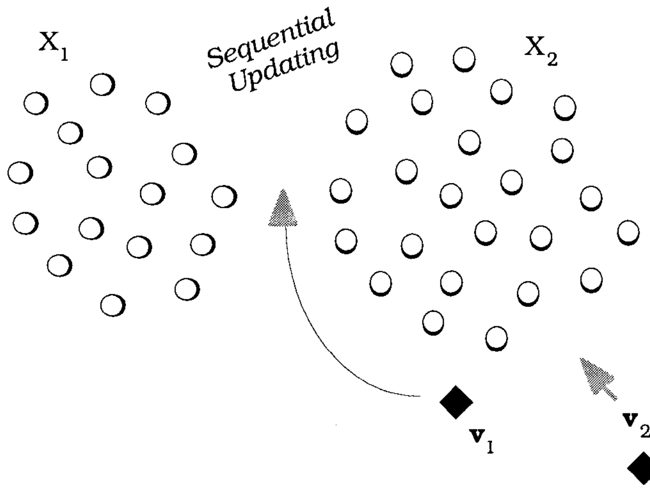


Figure 4.15 A low value of m may produce bad prototypes

In Figure 4.15 prototype $\mathbf{v}_1$ is closer to every point in X than $\mathbf{v}_2$ is. The result of this is that for *any* m at c = 2, the class 1 memberships $\{u_{1k}\}$ of every point in X computed with (2.7a) will be higher than the class 2 memberships $\{u_{2k}\}$. Since $u_{1k} + u_{2k} = 1$ for all k, the two rows of membership matrices produced with (2.7a) for any m will look like this:

$$U(m) = \begin{bmatrix} \leftarrow\cdots & (>0.5) & \cdots\rightarrow \\ \leftarrow\cdots & (<0.5) & \cdots\rightarrow \end{bmatrix} \xrightarrow{m \xrightarrow{+} 1} U(1) = \begin{bmatrix} \leftarrow\cdots & (\rightarrow 1) & \cdots\rightarrow \\ \leftarrow\cdots & (\rightarrow 0) & \cdots\rightarrow \end{bmatrix}$$

So, when m is close to 1, memberships of points in both X_1 and X_2 in class 1 will be close to 1. The effect of this is that the sequential updating strategy (4.12a) with learning rates based on (2.7a) will force prototype $\mathbf{v}_1$ in Figure 4.15 to migrate towards the grand mean $\bar{\mathbf{v}}$ of X, and $\mathbf{v}_2$ will not change much.

On the other hand, if m is large (say > 7) all of the u_{ik}'s will be nearly 1/c. In this case both prototypes in Figure 4.15 will be pulled towards the data very slowly because $(u_{ik,t})^m \approx 1/c^m$. So when m is large, for any competitive learning scheme whose update rate is a monotonic function of the $\{u_{ik,t}\}$, every prototype will be updated to almost the same very small extent (e.g., with c = 3 and m = 7, every $u_{ik,t} \approx 0.0004$).

Thus, if the memberships at (2.7a) are to be used in (4.12c), neither low nor high values of m seem desirable. However, if we start with a high value of m, and then slowly reduce it *during iteration*, this undesirable situation is avoided. Motivated by this, Tsao et al. (1993) defined the batch *fuzzy learning vector quantization* (FLVQ) algorithm via the heuristic learning rates

$$\alpha_{ik,t}^{FLVQ} = u_{ik,t}^{m_t} = \left(\sum_{j=1}^{c} \left(\|\mathbf{x}_k - \mathbf{v}_{i,t}\|_A \Big/ \|\mathbf{x}_k - \mathbf{v}_{j,t}\|_A \right)^{\frac{2}{m_t - 1}} \right)^{-m_t} \quad \forall\ i, k; \quad (4.28a)$$

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \sum_{k=1}^{n} \alpha_{ik,t}^{FLVQ}(\mathbf{x}_k - \mathbf{v}_{i,t-1}) \Big/ \sum_{s=1}^{n} \alpha_{is,t}^{FLVQ} \quad \forall\ i \quad \text{, where} \quad (4.28b)$$

$$m_t = m_0 + t[(m_f - m_0)/T] = m_0 + t\Delta m\ ;\ m_f, m_0 > 1;\ t=1,\ldots,T. \quad (4.28c)$$

Equation (4.28b) can be rewritten as $\mathbf{v}_{i,t} = \sum_{k=1}^{n} \alpha_{ik,t}^{FLVQ} \mathbf{x}_k \Big/ \sum_{s=1}^{n} \alpha_{is,t}^{FLVQ}$. Comparing this to (2.7b), equation (4.28c) asserts that when $m_0 = m_f = m$ is fixed, FLVQ *is* FCM-AO. Since m_t in (4.25c) is variable, we can have three families of FLVQ algorithms, depending on the choice of the initial (m_0) and final (m_f) values of m. For $t \in \{1, 2, \ldots, T\}$,

$$m_0 > m_f \Rightarrow \{m_t\} \downarrow m_f : \text{Descending FLVQ} = \downarrow\text{FLVQ} \tag{4.29a}$$

$$m_0 < m_f \Rightarrow \{m_t\} \uparrow m_f : \text{Ascending FLVQ} = \uparrow\text{FLVQ} \tag{4.29b}$$

$$m_0 = m_f \Rightarrow m_t \equiv m_0 \equiv m : \text{FLVQ} \equiv \text{FCM} \tag{4.29c}$$

We have included a discussion of $\uparrow$FLVQ here for completeness, but its properties as functions of m_t seem opposite to the intuitively desirable properties shared by SCS and $\downarrow$FLVQ. Here we concentrate on and describe in Table 4.12 the implementation of $\downarrow$FLVQ based on equations (4.28) and (4.29a), which is used in Example 4.3, and with modifications as set out in Baraldi et al. (1998), Example 4.26.

Table 4.12 Descending FLVQ ($\downarrow$FLVQ), Tsao et al. 1993

Store	(Un)labeled Object Data $X_{tr} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$ $U_{tr} \in M_{hcn}$ = Labels of vectors in X_{tr} (if available)
Pick	☺ number of nodes : $1 < c < n$ ☺ max. # of iterations : T ☺ distance measure : $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|_A$ ☺ termination measure : $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|$ ☺ termination criterion : ε ☺ $7 > m_0 > m_f > 1.1$
Get	☺ initial prototypes: $\mathbf{V}_0 \in \Re^{cp}$
Do	$t \leftarrow 1; E_0$ = high value DO UNTIL ($t > T$ or $E_{t-1} \le \varepsilon$) $m_t = m_0 + t[(m_f - m_0)/T]$ For k = 1 to n $\alpha_{ik,t}^{FLVQ} = \left(\sum_{j=1}^{c} \left(\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|_A \Big/ \|\mathbf{x}_k - \mathbf{v}_{j,t-1}\|_A \right)^{\frac{2}{m_t - 1}} \right)^{-m_t}$ Next k For i = 1 to c $\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \sum_{k=1}^{n} \alpha_{ik,t}^{FLVQ}(\mathbf{x}_k - \mathbf{v}_{i,t-1}) \Big/ \sum_{s=1}^{n} \alpha_{is,t}^{FLVQ}$ Next i $t \leftarrow t + 1$ END UNTIL $\mathbf{V} \leftarrow \mathbf{V}_{t-1}, U \leftarrow U_{t-1}$

As with LVQ and SCS, $\downarrow$FLVQ prototypes can be used with equation (2.6a) to produce a crisp partition of X. Notice, however, that at termination a fuzzy partition is also available, and it will be part of an optimal pair $(U_t, \mathbf{V}_t)$ for the FCM objective function J_{m_t} *at the terminal value of* m_t. In either case, the final prototypes can be used to define a 1-np or 1-nmp classifier. Our implementation of $\downarrow$FLVQ is necessarily batch, and this preserves its relationship to FCM-AO.

Another difference worth noting is that unlike FCM-AO, $\downarrow$FLVQ does not optimize a fixed objective function. All we can say about this is that since $\downarrow$FLVQ uses equations (4.28) at each iteration with $m=m_t$, every full step of $\downarrow$FLVQ finds a pair $(U_t, \mathbf{V}_t)$ that are necessary for a local extrema of J_{m_t}.

Observe the constraints $7 > m_0 > m_f > 1.1$ in our specification of $\downarrow$FLVQ. These empirically chosen limits may prevent numerical instability - in other words, stay away from 1 and infinity (see Baraldi et al., 1998 for more discussion on this aspect of $\downarrow$FLVQ).

The vector $\mathbf{u}_t(\mathbf{x}_k) = (u_{1k,t}, u_{2k,t}, \ldots, u_{ck,t})^T$ is a fuzzy label vector for $\mathbf{x}_k$, $\mathbf{u}_t(\mathbf{x}_k) \in N_{fc}$. This means that the sum of the $\downarrow$FLVQ learning rates for input vector $\mathbf{x}_k$ at any iterate t satisfies the same constraint as the SCS learning rates: $0 < \sum_{i=1}^{c} \alpha_{ik,t}^{FLVQ} \le 1$.

To understand how m_t acts to control the distribution and values of the learning rates $\{\alpha_{ik,t}^{FLVQ}\}$ in FLVQ, we discuss $\downarrow$FLVQ in more detail. The general situation can be understood by examining the learning rates at (4.28a) for fixed c, $\{\mathbf{v}_{i,t}\}$ and m_t. In this case,

$$\alpha_{ik,t}^{FLVQ} = \kappa^{-m_t}\left(\left\|\mathbf{x}_k - \mathbf{v}_{i,t}\right\|_A^{-2m_t/(m_t-1)}\right) \quad , \qquad (4.30)$$

where $\kappa = \sum_{j=1}^{c}\left(1 / \left\|\mathbf{x}_k - \mathbf{v}_{j,t}\right\|_A\right)^{2/(m_t-1)}$ is a positive constant. Equation (4.30) shows that the contribution of $\mathbf{x}_k$ to the next update of the node weights is inversely proportional to their distances from it, so the winner for this k is the $\mathbf{v}_{i,t-1}$ closest to $\mathbf{x}_k$. Larger values of m_t lead to fuzzier values of $u_{ik,t}$ (values closer to 1/c), and

$\sum u_{ik,t} = 1 \Rightarrow \sum \alpha_{ik,t}^{FLVQ} \leq 1$. So, in the initial stages of $\downarrow$ FLVQ large values of m_t (near m_0) yield updates with lower individual learning rates.

In the initial stages of SCS (for low values of t) $p_{ik,t} \approx 1/c$, and since the counters $\{n_{i,t}\}$ all start at 1, at the beginning of the SCS learning process each prototype is (more or less) updated to the same extent. In other words $\alpha_{ik,t}^{SCS} = (\eta_{i,t} \cdot p_{ik,t}) \approx (\eta_{j,t} \cdot p_{jk,t}) = \alpha_{jk,t}^{SCS}$ for all i and j at low values of t. What happens for $\downarrow$FLVQ? In this case we start with a high value of $m = m_0$. For high values of m, $u_{ik,t} \approx 1/c\ \forall i$, and as a result $\alpha_{ik,t}^{FLVQ} = (u_{ik,t})^{m_t} \approx \alpha_{jk,t}^{FLVQ} = (u_{jk,t})^{m_t}$ for all i and j at low values of t. Thus, in $\downarrow$ FLVQ all c prototypes will have about the same importance at the beginning of iteration, with learning rates at each $\mathbf{x}_k$ that are roughly uniformly distributed across the c nodes during updates. Thus, $\downarrow$FLVQ and SCS start with similar learning rates.

As iteration continues $p_{ik,t}$ for SCS and $u_{ik,t}$ for $\downarrow$FLVQ both tend to peak at the winner. For SCS, $p_{ik,t} \to 1$ when node i is the winner, but $\eta_{ik,t} \to 0$, so if the iteration is allowed to continue indefinitely the overall SCS learning rate $\eta_{ik,t} \cdot p_{ik,t} \to 0$ *almost everywhere* - that is, except on a set of measure zero in $\Re$ (recall that $\eta_{ik,t} = 1$ is reset at all the perfect squares in $\Re$). On the other hand, $u_{ik,t} \to 1$ for $\downarrow$ FLVQ when node i is the winner but since $m_t \to 1$, the overall learning rate for this method also goes to 1, $\alpha_{ik,t}^{FLVQ} \to 1$. As $m_t \searrow m_f$ (m_t gets closer to 1), more and more of the update is given to the winner node. That is, the lateral distribution of learning rates is a function of t, which in $\downarrow$ FLVQ sharpens at the winner node as $m_t \searrow m_f$. Indeed, the learning rate characteristics of $\downarrow$FLVQ are roughly opposite to the usual behavior imposed on them by other competitive learning schemes. In LVQ and SCS *all c* learning rates at $\mathbf{x}_k$ decrease towards 0 (everywhere for LVQ, and almost everywhere for SCS) as t increases (this imbues them with stability and improves the chance they will satisfy the termination condition), but in $\downarrow$FLVQ, the winner learning rate tends to increase towards 1 during learning, while the other c-1 rates tend towards zero. So, SCS behaves more like LVQ as iteration proceeds than $\downarrow$ FLVQ does. Nonetheless, $\downarrow$ FLVQ seems to terminate rapidly in the literature that illustrates its use.

Example 4.3 We abbreviate some results given by Bezdek and Pal (1995) to illustrate and compare LVQ, SCS and FLVQ by again using Anderson's (1935) Iris data. Two initializations, shown in Table 4.13, are used: I_1 is the set $\bar{\mathbf{V}}$ of subsample means; and I_2 is computed with (4.21).

Table 4.13 Two initializations for the numerical experiments

Init. I_1 = (Means)					Init. I_2 via (4.21)			
5.01	3.43	1.46	0.25	$\leftarrow \mathbf{v}_{1,0} \rightarrow$	4.30	2.00	1.00	0.10
5.94	2.77	4.26	1.33	$\leftarrow \mathbf{v}_{2,0} \rightarrow$	6.10	3.20	3.95	1.30
6.59	2.97	5.55	2.03	$\leftarrow \mathbf{v}_{3,0} \rightarrow$	7.90	4.40	6.90	2.50

None of the algorithms used class information (that is, are supervised) during learning. Table 4.14 shows the results of 1-np classification (with δ the Euclidean metric) of Iris using the (relabeled) terminal centroids recommended by LVQ, SCS and FLVQ.

Table 4.14 Sample Mean, LVQ, SCS and FLVQ 1-np classifiers on the Iris data when initialized with I_1

Initial Prototypes I_1				Confusion Matrix		
5.01	3.43	1.46	0.25	50	0	0
5.94	2.77	4.26	1.33	0	46	4
6.59	2.97	5.55	2.03	0	7	43
Final Prototypes : LVQ $T=50$, $\alpha_0=0.6$				Confusion Matrix		
5.00	3.42	1.46	0.25	50	0	0
5.87	2.74	4.37	1.41	0	47	3
6.81	3.08	5.68	2.08	0	13	37
Final Prototypes : SCS $T=50$, $\hat{\gamma}=1.3$, $T_0=40$				Confusion Matrix		
5.01	3.42	1.46	0.25	50	0	0
5.88	2.74	4.370	1.41	0	47	3
6.78	3.05	5.63	2.03	0	13	37
Final Prototypes : $\downarrow$ FLVQ $T=50$, $m_0=5$, $m_f=1.5$				Confusion Matrix		
5.01	3.42	1.47	0.25	50	0	0
5.88	2.75	4.37	1.41	0	47	3
6.82	3.06	5.70	2.06	0	14	36

The confusion matrix associated with $\mathbf{D}_{\mathbf{V},\mathbf{E},\delta}$ when $\mathbf{V}_f = \mathbf{V}_0 = I_1$ shows that the sample means yield a 1-np classifier that commits 11 errors; 4 class 2 points are labeled class 3; and 7 class three points are labeled class 2. All three algorithms produce very similar prototypes. The confusion matrices for the LVQ and SCS based 1-np designs are identical, showing 16 resubstitution errors. FLVQ is very nearly the same, committing one more error than LVQ and SCS on a class 3 data point.

SCS seems very sensitive to the choice of and interaction between $\hat{\gamma}$ and T_0. Table 4.15 studies the effect on SCS outputs to the parameters $\hat{\gamma}$ and T_0.

Table 4.15 Some outputs of the SCS 1-np Classifier on Iris

	Init.		$\hat{\gamma}=1.30$, $T_0=40$			Confusion	Matrix	
		5.006	3.425	1.465	0.247	50	0	0
A	I_1	5.884	2.743	4.370	1.414	0	47	3
		6.776	3.047	5.634	2.031	0	13	37
			$\hat{\gamma}=1.15$, $T_0=40$			Confusion	Matrix	
		5.843	3.057	3.758	1.199	50	0	0
B	I_1	5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
			$\hat{\gamma}=1.30$, $T_0=40$			Confusion	Matrix	
		5.006	3.425	1.465	0.247	50	0	0
C	I_2	5.884	2.743	4.370	1.414	0	47	3
		6.776	3.047	5.634	2.031	0	13	37
			$\hat{\gamma}=1.15$, $T_0=40$			Confusion	Matrix	
		5.843	3.057	3.758	1.199	50	0	0
D	I_2	5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
			$\hat{\gamma}=1.30$, $T_0=60$			Confusion	Matrix	
		5.008	3.378	1.548	0.284	50	0	0
E	I_2	6.272	2.884	4.945	1.690	3	0	47
		6.292	2.884	4.945	1.690	0	0	50
			$\hat{\gamma}=1.30$, $T_0=70$			Confusion	Matrix	
		5.843	3.057	3.758	1.199	50	0	0
F	I_2	5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0

All runs used T=50; rows A are repeated from Table 4.14. First compare A, B, C and D, all of which have T_0 =40. Changing $\hat{\gamma}$ from 1.30 to 1.15 using either I_1 or I_2 has the dramatic result of forcing all three SCS centroids to terminate at $\bar{\mathbf{v}} = (5.843, 3.057, 3.758, 1.199)^T$ - the grand mean of Iris. This has the very predictable bad effect on the 1-np design based on these prototypes of it committing 100 mistakes in both cases.

Next, compare sets C and F in Table 4.15 to see that it is not just a change of $\hat{\gamma}$ that has this effect on SCS, for in this case you will see that the same result occurs with $\hat{\gamma}$ fixed at 1.30 but T_0 increased from 40 to 70. Finally, look at sets C, E and F for I_2 and $\hat{\gamma}$=1.30 fixed. Intermediate between the good result at T_0=40 and the worst result at T_0=70 is the case T_0= 60, for which SCS terminates with a good estimate of the first centroid, but identical vectors for the second and third prototypes, resulting in a 1-np error rate of 50 mistakes. Table 4.15 - and many other experiments with other values for $\hat{\gamma}$ and T_0 not reported here - suggest that SCS is very sensitive to choices for these two parameters.

Another set of runs (not shown here) for all three algorithms that used the same parameters but which were started at initialization I_2 yielded prototypes that were identical (to three decimal places) to those shown in Table 4.14. This does not establish that these three algorithms are insensitive to initialization, but it gives us some confidence that the Iris data are (in the eyes of these algorithms) rather well structured. The important point is that there *are* combinations of initializations and algorithmic parameters for all three algorithms that produce very similar and predictable results. This is usually the case for competing algorithms - given enough time, most models for a particular class of problems can be made to yield pretty similar results.

I. The relationship between c-Means and CL schemes

In (2.7a) and (2.7b) the weighting exponent m for J_m is fixed, but in (4.28a) it is a variable. Since m is replaced by a parameter whose value depends on the number of iterations that have elapsed, m_t plays a role that is somewhat analogous to $\alpha_{ik,t}^{LVQ}$ in LVQ. To see this, remember that $\sum_{i=1}^{c} u_{ik,t} = 1$ for each $\mathbf{x}_k$ in X. Consequently, the learning rates in (4.28a) that are applied to all c nodes via (4.28b) for each $\mathbf{x}_k$ are dependent on each other, and themselves must satisfy the condition $\sum_{i=1}^{c} \alpha_{ik,t}^{FLVQ} \le 1$. The effect of controlling the learning rates this way is best understood by considering a simple example. Suppose c=5 and m_t=4 at some iterate. Two label vectors for $\mathbf{x}_k$ for the five nodes, and the resultant learning rate distribution vectors they induce via (4.28a) are shown below:

$$\mathbf{u}(\mathbf{x}_k) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \boldsymbol{\alpha}_k^{FLVQ} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ for any } m_t \qquad ; \text{and} \qquad (4.31a)$$

$$\mathbf{u}(\mathbf{x}_k) = \begin{pmatrix} 0.1 \\ 0.6 \\ 0.0 \\ 0.2 \\ 0.1 \end{pmatrix} \Rightarrow \boldsymbol{\alpha}_k^{FLVQ} = \begin{pmatrix} 0.0001 \\ 0.1296 \\ 0.0000 \\ 0.0016 \\ 0.0001 \end{pmatrix}, \ (m_t = 2 \text{ is illustrated}). \qquad (4.31b)$$

In (4.31a) node 2 is the crisp winner since it receives all of the membership of this data point in the five clusters. From (4.25a) it follows that for *any* value of m_t the learning rates applied to this data point will also be crisp, and will be the same as the labels used to compute them, as shown in (4.31a). Thus, when a single node *can* win *all of the membership*, none of the non-winner nodes are allowed to influence the update in (4.28b) *for that data point*. In this special case, FLVQ reverts to an LVQ - like strategy - but only for data points that have crisp memberships.

On the other hand, if the distribution of memberships for $\mathbf{x}_k$ is truly fuzzy, as in (4.31b), exponentiation of the membership values by m_t has a noticeable effect on the role played by each node in the update scheme. The winner node in (4.31b) in the sense of maximum membership (which is, as previously noted, also the minimum distance prototype) is still node 2. But in this second case, non-winner nodes with non-zero memberships will also participate in the determination of how much to change their corresponding weight vectors for that data point. Finally, if m_0=m_f then clearly FCM=FLVQ.

If all n membership columns in U from the FCM formula (2.7a) were crisp, (4.28b) would become a batch version, LVQ-style update, with

$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \sum_{\mathbf{x}_k \in i} (\mathbf{x}_k - \mathbf{v}_{i,t-1}) / n_{i,t}$, where $n_{i,t}$ is the number of points in the i-th crisp cluster of X at iterate t. The previous estimate for $\mathbf{v}_{i,t-1}$ can be eliminated from this last equation by distributing the sum over the minus sign, leaving the HCM update formula (2.6b). Suppose we replace equation (4.12c) with this batch update formula and *require* the calculation of U_{LVQ} with the nearest prototype rule (2.6a) or (4.2) at each pass through X (remember that LVQ does not do so). Call this *extended batch LVQ* (EBLVQ). Then FLVQ reduces to EBLVQ whenever U is crisp, and further, EBLVQ is precisely HCM. In this sense FLVQ is a true generalization of both LVQ and HCM that integrates their models in perhaps the strongest possible way.

A somewhat more formal analysis of the relationship between FLVQ and FCM is elaborated in Karayiannis and Bezdek (1997). Karayiannis (1997a) provides a fairly comprehensive survey of learning vector quantization that includes not only FLVQ, but a number of more general formulations that have interesting connections to generalizations of all three c-means families.

J. The mountain "clustering" method (MCM)

Yager and Filev (1994a) developed a prototype generation algorithm for *unlabeled* data that is very different in spirit than all of the previous methods discussed in this section. In their scheme a very large finite set of candidate prototypes are specified and fixed, and the MCM objective function is then used to select c good prototypes from the fixed set of candidates. In short, prototypes are not initialized and then iteratively updated, but simply chosen iteratively from a (very large and fixed) discrete set.

MCM begins by specifying a lattice of coordinates that capture the *unlabeled* data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$. Without loss of generality we describe a simplified version of MCM that uses an integer lattice. We construct the lattice by first enlarging the hyperbox hb($\mathbf{m}$,$\mathbf{M}$) using the floors and ceilings of the features instead of the given values in equations (4.20). Thus, with $\lfloor x_{jk} \rfloor$ and $\lceil x_{jk} \rceil$ denoting the integer floor and ceiling of x_{jk}, respectively, we compute hb($\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil$), which is the smallest hyperbox with corners having integer coordinates that contains X as a proper subset.

For $1 \le j \le p$, the j-th edge of hb($\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil$) is composed of, say, r_j integers that run from the floor of the minimum, $\lfloor m_j \rfloor$, to the ceiling of the maximum, $\lceil M_j \rceil$. The lattice Lhb($\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil$) $\doteq$ Lhb of *integer grid points* (or nodes) in $\mathrm{hb}(\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil)$ comprises the set of candidate prototypes for the MCM model. We will use our usual notation for the point prototypes in this set, i. e., $\mathbf{v}_i \in \mathrm{Lhb}$, and ask you to remember that their coordinates are integers in this subsection only.

Next, calculate the $n \cdot r_1 \cdot r_2 \cdot \cdots \cdot r_p$ distances $\{\delta(\mathbf{v}_i, \mathbf{x}_k)\}$. Yager and Filev (1994a) discuss using only Minkowski metrics (1.11) for this, but it is clear that inner product metrics in the family at (1.6) are equally applicable. Unlike, say, any of the c-means models, the MCM objective function is not fixed. Instead, the model begins with an initial objective function $J_{MCM,1}$, and then uses the current set of values in subsequent iterations to define a new objective function $J_{MCM,t}$ at each $t > 1$, very much like the objective function J_{m_t} used by FLVQ. The initializing objective function is

$$J_{MCM,1}(\mathbf{v}_j; X) = \sum_{k=1}^{n} e^{-\alpha\delta(\mathbf{v}_j, \mathbf{x}_k)} \, , \; \mathbf{v}_j \in \mathrm{Lhb} \qquad , \qquad (4.32)$$

where α is a positive constant. If we regard $e^{-\alpha\delta(\mathbf{v}_j, \mathbf{x}_k)}$ as the "potential" at $\mathbf{v}_j$ due to $\mathbf{x}_k$, then $J_{MCM}(\mathbf{v}_j; X)$ measures the total potential at $\mathbf{v}_j$ due to the data. Thus, the total potential $J_{MCM}(\mathbf{v}_j; X)$ will be high when many data points are concentrated near $\mathbf{v}_j$. Yager and Filev thus argue that maxima of (4.32) identify good prototypes.

Put another way, for a fixed $\mathbf{v}_j \in \mathrm{Lhb}$, the maximum (minimum) value of J_{MCM} occurs at the minimum (maximum) value of $\delta(\mathbf{v}_j, \mathbf{x}_k)$ over $1 \le k \le n$. Since J_{MCM} sums up the n values $\left\{e^{-\alpha\delta(\mathbf{v}_j, \mathbf{x}_k)}\right\}$ at node $\mathbf{v}_j$, J_{MCM} will be proportional to the density of points in X in the neighborhood of $\mathbf{v}_j$. A plot of the values $\{J_{MCM}(\mathbf{v}_j; X)\}$ over $\mathbf{v}_j \in \mathrm{Lhb}$ should, for compact well separated clusters at least, be a digital surface with (mountain) peaks at nodes where the density of the data is highest - i.e., where there are clusters. Hence the term "mountain function" for (4.32).

Maximization of $J_{MCM,t}$ over $\mathbf{v}_j \in \mathrm{Lhb}$ is accomplished by simply enumerating its values and finding the largest one, ties being resolved arbitrarily. We let the set of initial *mountain function values* (MFVs) be

$$\mathrm{MFV}_1 = \left\{ J_{\mathrm{MCM},1}(\mathbf{v}_j;X) : \mathbf{v}_j \in \mathrm{Lhb}, 1 \le j \le \prod_{j=1}^{p} r_j \right\} \quad . \tag{4.33}$$

If $\hat{\mathbf{v}}_1 = \underbrace{\arg\max}_{j}\left\{J_{\mathrm{MCM},1}(\mathbf{v}_j;X)\right\}$, lattice point $\hat{\mathbf{v}}_1$ is taken as the first prototype. The next step in MCM is to "destroy" the peak at $\hat{\mathbf{v}}_1$, redefining the mountain function by subtracting from each $J_{\mathrm{MCM},1}(\mathbf{v}_j;X)$ a fractional amount of $J_{\mathrm{MCM},1}(\hat{\mathbf{v}}_1;X)$ that is also inversely proportional to the distance $\delta(\hat{\mathbf{v}}_1,\mathbf{v}_j)$. This results in a new set of values MFV_{t+1} of the modified objective function $J_{\mathrm{MCM},t}$, which after t $\ge$1 steps, take the form

$$J_{\mathrm{MCM},t+1}(\mathbf{v}_j;X) = J_{\mathrm{MCM},t}(\mathbf{v}_j;X) - \left(e^{-\beta\delta(\hat{\mathbf{v}}_t,\mathbf{v}_j)}\right)\left(J_{\mathrm{MCM},t}(\hat{\mathbf{v}}_t;X)\right), \tag{4.34}$$

for $\mathbf{v}_j \in \mathrm{Lhb}, 1 \le j \le \prod_{j=1}^{p} r_j$, where β is a second user-defined positive constant and $\hat{\mathbf{v}}_t$ is the t-th prototype. Maximization over MFV_2 produces a second winning node, say $\hat{\mathbf{v}}_2$, and $\hat{\mathbf{v}}_1$ is a candidate to also become $\hat{\mathbf{v}}_2$, an occurrence which is called "node reuse" by Barone et al. (1995). In any case, $\hat{\mathbf{v}}_2$ is the second MCM prototype, etc. Equation (4.34) thus defines an iterative procedure that continues to select nodes from the lattice as prototypes for the data until a user-defined termination criterion is met. Yager and Filev (1994a) recommend termination when the ratio of successive maximum values of the mountain function is small, i.e., at the first i for which

$$\frac{J_{\mathrm{MCM},t}(\hat{\mathbf{v}}_{i,t};X)}{J_{\mathrm{MCM},t-1}(\hat{\mathbf{v}}_{i-1,t-1};X)} < \varepsilon \quad , \tag{4.35}$$

for some termination threshold $\varepsilon > 0$. At this point MCM has produced the set $\mathbf{V}_{\mathrm{MCM}} = \{\hat{\mathbf{v}}_1,\ldots,\hat{\mathbf{v}}_t\} \subset \mathrm{Lhb}$, which are taken as prototypes for t (as yet undefined) clusters in X.

This method is simple, and like all algorithms, has some parameters to pick. As mentioned above, it may happen that MCM uses the same node more than once, since the amount subtracted from each mountain value in (4.34) depends on β, and for the wrong choice, may not be enough to flatten a particularly strong peak. Barone et al. (1995) provide an in depth analysis and empirical recommendations for choosing α, β and ε, and also discuss the issue of peak reusability. Our simplified description of MCM uses an integral grid size, but the lattice of prototypes could be either finer or courser than this. Barone et al. (1995) consider the issue of grid size, and also discuss the choice of a metric for the distance calculations. Table 4.16 summarizes the MCM method of prototype generation.

Table 4.16 MCM prototype generation (Yager and Filev, 1994a)

Store	Unlabeled Object Data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$
Pick	☞ positive constants α and β ☞ distance measure : $\lVert \mathbf{x}_k - \mathbf{v}_{i,t-1} \rVert$ ☞ termination measure : $E_t = J_{MCM,t}(\hat{\mathbf{v}}_{i,t}; X) / J_{MCM,t-1}(\hat{\mathbf{v}}_{i,t-1}; X)$ ☞ termination criterion : ε
Get	☞ Lattice $Lhb(\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil) \doteq Lhb$
Do	E_0 = high value $\hat{\mathbf{v}}_1 = \underbrace{\arg\max}_{\mathbf{v}_j \in Lhb} \left\{ J_{MCM,1}(\mathbf{v}_j; X) \right\}$ $t \leftarrow 1$ DO UNTIL $(E_t \le \varepsilon)$: $\hat{\mathbf{v}}_{t+1} = \underbrace{\arg\max}_{\mathbf{v}_j \in Lhb} \left\{ J_{MCM,t+1}(\mathbf{v}_j; X) \right\}$ $t \leftarrow t+1$ END UNTIL $\mathbf{V}_{MCM} = \{\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_t\} \subset Lhb$

If no peak is reused before MCM terminates, then c = t, that is, the number of distinct prototypes corresponds to the last value of t in Table 4.16. On the other hand, when one or more peaks is reused, the number of distinct prototypes determined by MCM is, say, $c \le t$. In either case, MCM starts with c = 1 prototype, much like a divisive hierarchical clustering method, and continues to add (possibly non-distinct) prototypes until its termination criterion is met. At first glance, this seems to bypass the cluster validity problem. However, the number of prototypes determined by MCM depends on α, β and ε, so validation is still a problem - just not an explicit one. Barone et al. do discuss cluster validity, and suggest validating the number of prototypes selected by a novel application of singular value decomposition applied to the $t \times p$ matrix $\mathbf{V}_{MCM}$. They recommend looking for one or more "breaks" in the list of singular values (very similar in spirit to Hubert's knees in Chapter 2), and basing the final estimate of c on this procedure.

The clustering part of MCM amounts to using $\mathbf{V}_{\mathrm{MCM}}$ to compute, for example, the crisp nearest prototype labels of X. Some computational experiments report finding good clusters this way, but it is easy to construct data for which this method fools the user badly. This disclaimer aside, MCM has been used some for one important application, and that is as a simple and often successful way to initialize other clustering and/or prototype generator algorithms. Indeed, Barone et al. (1995) advocate this themselves, and offer several examples to support their claim that terminal MCM prototypes are often very similar to those found by other methods.

Example 4.4 (Barone et al., 1995). Table 4.17 juxtaposes the terminal prototypes found by MCM and FCM on the data set Iris_{34} shown in Figure 4.12. The first column in Table 4.17 also shows the symbols used for the 2D means shown in Figure 4.12

Table 4.17 Terminal MCM and FCM prototypes for Iris_{34}

	Means $\bar{\mathbf{v}}$		$\mathbf{V}_{\mathrm{FCM}}$		$\hat{\mathbf{V}}_{\mathrm{MCM}}$	
✹ $\bar{\mathbf{v}}_{34,1}$	1.46	0.25	1.46	0.25	1.66	0.37
★ $\bar{\mathbf{v}}_{34,2}$	4.26	1.33	4.28	1.35	4.28	1.43
✚ $\bar{\mathbf{v}}_{34,3}$	5.55	2.03	5.62	2.05	5.59	2.23

Barone et al. used the Euclidean norm for both algorithms, and set c = 3 for FCM. They state that α was set at 4 for MCM, but do not specify β and ε, or any of the other processing parameters for FCM that give the results in Table 4.17. Since the MCM values in Table 4.17 are non integral, we know that the lattice used by MCM for these calculations was considerably finer (at least fine enough to have grid points with coordinates to two decimal places) than the unit lattice $\mathrm{Lhb}(\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil)$ used in our specification of MCM.

The conclusion we draw from Table 4.17 is that, given the right choices for MCM, it can produce prototypes that are reasonable initializers for FCM. Notice that the MCM estimate of $\bar{\mathbf{v}}_{34,1}$ seems to be the worst of the three, but the 50 points which it represents are very compact and well separated from the remaining 100 points in Iris_{34} (cf. Figure 4.12).

Perhaps the biggest and certainly most evident problem with MCM is computational complexity. If p is more than two or three, and/or the range of the data set X in any of its p dimensions is large, the lattice Lhb used in our description of MCM will be very large indeed, because $|\text{Lhb}(\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil)| = r_1 \cdot r_2 \cdot \cdots \cdot r_p$. For the two dimensional data set Iris_{34}, this amounts to (700)(300)=210,000 initial prototypes to cover the lattice $\text{Lhb}((0,0)^T, (7,3)^T)$. In a non-specific setting, suppose X contains data points in 10 dimensions - a not uncommonly large number of features. If each of the 10 axes is subdivided by 10, the unit lattice $\text{Lhb}(\lfloor \mathbf{m} \rfloor, \lceil \mathbf{M} \rceil)$ will have 10^{10} candidate prototypes - too many to make MCM computationally tractable.

Chiu (1994, 1995, 1997) proposed a modification of MCM wherein the lattice of candidate grid points is abandoned, and replaced with X, the unlabeled input data. Chiu called his modification of MCM the *subtractive clustering method* (SCM), and it is not sufficiently novel or different from MCM to warrant a separate discussion here. (We will, however, discuss SCM again in Example 4.18.)

Since the candidate prototypes in SCM now coincide with the data, there are only n of them, and the complexity issue would *seem* to resolved. However, Davé and Krishnapuram (1997) have shown that the complexity of SCM is still $O(n^2)$, while the complexity of FCM is $O(n)$. They further discuss the relationship between SCM, PCM and other clustering algorithms, including the potential function approach (Tou and Gonzalez, 1974).

Velthuizen et al. (1997) discussed a different set of modifications to MCM, and called the resultant algorithm the *modified mountain method* (M3). Noting that MCM is useful only if "good" values are chosen for the MCM parameters α and β, they suggest computing α based on a sample statistic of X. Letting $S = \sum_{k=1}^{n} (\mathbf{x}_k - \bar{\mathbf{v}})^T(\mathbf{x}_k - \bar{\mathbf{v}}) / n$ be the sample covariance matrix with $\bar{\mathbf{v}} = \sum_{k=1}^{n} \mathbf{x}_k / n$ the grand mean of n input points X in $\Re^p$, Velthuizen et al. suggest computing α as

$$\alpha = \left(\frac{pc}{rn\sqrt{\text{trace}(S)}} \right) \quad \text{, where} \tag{4.36}$$

$$r = n^{-1/(p+4)} \cdot \left(\frac{2^{p+2}\Gamma\left(\frac{p+2}{2}\right)}{(p+2)^{(p/2)+1}} \right)^{\frac{1}{p+4}} \quad . \tag{4.37}$$

Unlike MCM, the M3 model fixes c, the number of prototypes to seek, in (4.36). Velthuizen et al. also present a method for eliminating the sensitivity of MCM to β. The essence of this part of M3 is to pick a "reasonable" β – presumably by trial and error ($\beta = 0.06$ in Velthuizen et al.), isolate a neighborhood of the current winner prototype $\hat{\mathbf{v}}_t$ by finding the 5 nearest prototypes to it, and then introducing a finer local subgrid just in some enlargement of this neighborhood, over which the distribution of the data in the neighborhood is then fit with a multivariate normal distribution (you have to wonder a little about a fit to 5 points). Finally, $J_{\mathrm{MCM},t}(\hat{\mathbf{v}}_t;X)$ in (4.34) is replaced by the value of the Gaussian density just found in the neighborhood of $\hat{\mathbf{v}}_t$. The authors assert that this modification overcomes the sensitivity of MCM to parameter β.

The application domain of interest to Velthuizen et al. is *magnetic resonance* (MR) image segmentation. Let $T1_{ij}$, $T2_{ij}$ and ρ_{ij} denote, respectively, the spin lattice relaxation, transverse relaxation, and proton density of pixel (i,j) in an MR slice (three images at the same location in time and space) of size $m \times n$. If we aggregate these 3 numbers into a *pixel vector* $\mathbf{x}_{ij} = (T1_{ij}, T2_{ij}, \rho_{ij})$, the data set $X = \{\mathbf{x}_{11}, \mathbf{x}_{12}, \ldots, \mathbf{x}_{ij}, \ldots, \mathbf{x}_{mn}\}$ is in $\Re^3$; we will meet this 3D pixel vector data in several other examples in Chapters 4 and 5. The basic algorithm used by Velthuizen et al. proceeds as follows. Let X stand for a set of pixel feature vectors derived from any MR image, and denote the prototypes found by M3 as $\mathbf{V}_{M3}$ to distinguish them from $\mathbf{V}_{MCM}$. Then

[M3.1] run M3 on (unlabeled) X to find $\mathbf{V}_{M3}$;

[M3.2] construct U, a crisp 1-np labeling of X with $\mathbf{D}_{\mathbf{V}_{M3},\mathbf{E},\delta}$ with equation (4.2): the label assigned to pixel vector $\mathbf{x}_{ij}$ is the algorithmic label (index) of the closest prototype;

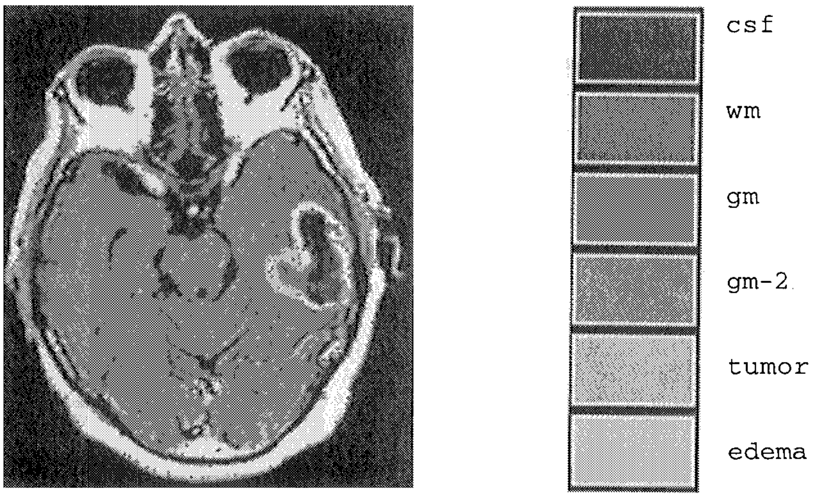
[M3.3] physically relabel each cluster in U as a tissue class by matching the pixels in each algorithmic cluster to one of the ground truth tissue clusters. Assign the algorithmic cluster to the tissue class that enjoys maximum pixel matching (this is a different relabeling method than the one given in Section 4.3.B);

[M3.4] artificially color the labeled image.

Example 4.5 (Velthuizen et al., 1997) Velthuizen et al. (1997) evaluated segmentations of 13 MR images using two types of ground truth. Three of the test images had manual ground truth (GT1) for c = 10 tissue classes derived by visual inspection and marking of each image by a trained radiologist. Segmentations were produced by four methods: a supervised *7-nearest neighbor* (k-nn, see Section 4.4) rule, which was used to construct type GT2 ground truth for the other 10 images; and unsupervised M3, unsupervised FCM($\mathbf{V}_o$) and unsupervised FCM($\mathbf{V}_{M3}$). Comparisons were made visually and quantitatively.

Segmentation of an MR image by FCM was done with two initializations: a "standard" initialization $\mathbf{V}_o$ (cf. (9) in Velthuizen et al., 1997); and with $\mathbf{V}_{M3}$. We write FCM(**V**) to indicate FCM initialized with **V**. FCM generates a terminal fuzzy c-partition U_{FCM} of X which is hardened using equation (1.15), and finally, steps [M3.3] and [M3.4] are performed on the resultant crisp partition.

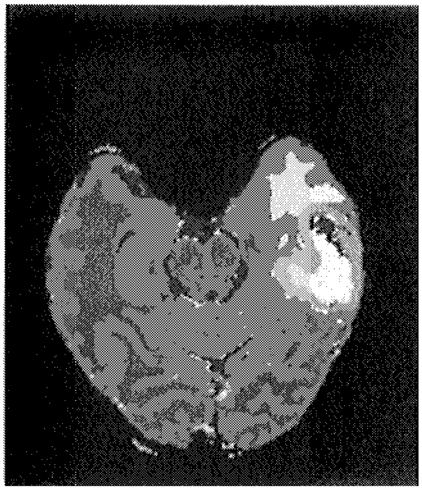
Figure 4.16 shows the T1 (weighted) input data for a patient that has a brain tumor. Figure 4.16(b) is the color key for the images: csf = *cerebro spinal fluid*; wm = *white matter*; gm = *gray matter*; gm-2 = *(falsely labeled) gray matter*. *Edema* is an abnormal accumulation of tissue fluid resulting in swelling.

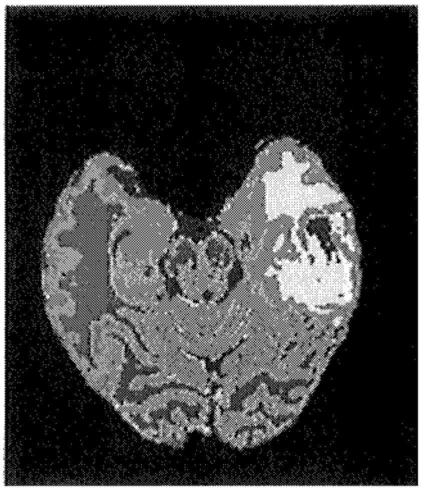


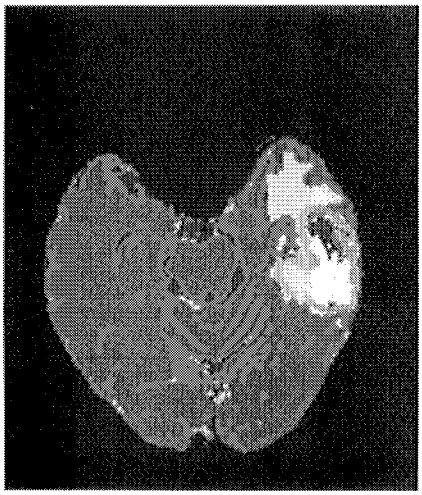
(a) T1 Weighted MR Image **(b) Color Legend**

Figure 4.16 MR segmentations (Velthuizen et al., 1997)

A supervised k-nn segmentation is shown in Figure 4.16(c). This image results from an operator choosing labeled subsets of pixels from each tissue class, and then using the standard k-nn rule to label the remaining pixels. This is repeated until a panel of radiologists agree that the k-nn segmentation is good enough to be used as type GT2 ground truth. Ten of the thirteen images discussed in this study used this method (GT2) as a basis for comparing the results of the three algorithms (unsupervised M3, unsupervised FCM($\mathbf{V}_o$) and unsupervised FCM($\mathbf{V}_{M3}$).


(c) 7-nn (type GT2, c = 5)


(d) FCM($\mathbf{V}_o$)


(e) M3

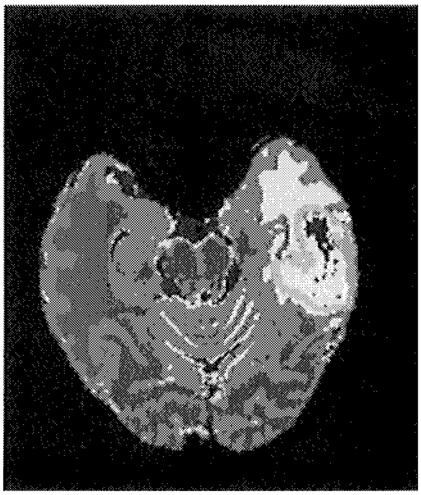

(f) FCM($\mathbf{V}_{M3}$)

Figure 4.16 (con't.) MR segmentations (Velthuizen et al., 1997)

Figure 4.16(d) shows a segmentation achieved by FCM($\mathbf{V}_o$). The tumor is not detected. Instead, FCM($\mathbf{V}_o$) finds two false gray matter regions that do not correspond to anatomical tissues. The M3 segmentation in Figure 4.16(e) is much better - it finds many of the tumor pixels and does not have false gray matter tissue regions. Panel (4.16f) shows a segmentation resulting from the initialization of FCM with the output of M3. This view should be compared to Figure 4.16(c). It's hard to see on a printed copy, but there is excellent correspondence between the tumor regions in these two views.
Table 4.18, adapted from Velthuizen et al. (1997), shows the *average* performance on pathological tissues for segmentations of the thirteen images made by unsupervised M3, unsupervised FCM($\mathbf{V}_o$) and unsupervised FCM($\mathbf{V}_{M3}$).

Table 4.18 Average true and false positive pixel counts (in %) for pathological tissues (Velthuizen et al., 1997, Table 1)

	False Positives			True Positives		
	FCM($\mathbf{V}_o$)	M3	FCM($\mathbf{V}_{M3}$)	FCM($\mathbf{V}_o$)	M3	FCM($\mathbf{V}_{M3}$)
Tumor	10.3	5.6	5.2	59.4	66.1	75.5
Edema	5.9	5.9	8.7	75.9	77.9	81.2

When FCM is initialized with $\mathbf{V}_o$, segmentation is not as good as M3 - it has nearly 5% more false positives and about 7% less true positives in tumor. In edema, the recognition rates are about the same. When FCM is initialized with $\mathbf{V}_{M3}$, there is substantial improvement in the true positive rate for both tissue classes and a slight decrease in edema false positives.

MCM, SCM and M3 are not really clustering methods - they generate prototypes in the feature space. Sometimes they find good clusters, but like LVQ and SCS, partitions of the data are not involved in the iterative procedure that produces the prototypes. Nonetheless, the examples of this subsection suggest that MCM, SCM and M3 are useful for initializing clustering algorithms such as the c-means families.

4.4 Nearest neighbor classifiers

Another widely used classifier design is the *k-nearest neighbor* (k-nn) rule, which *requires* labeled samples from each class. Figure 4.17 displays the geometry of this scheme. All that is needed is to choose k, the *number* of nearest neighbors to find in the neighborhood of any unlabeled vector **z** in $\Re^p$; and some measure of distance between pairs of vectors in $\Re^p$, usually Euclidean distance. The metric δ defines the *shape* of the capture neighborhood for the k nearest neighbors to **z**. The easiest voting scheme to justify and implement is to accept a simple majority of the votes for any class represented by points in the k-nn neighborhood. In this case, k is usually taken as an odd integer, precluding ties in the c = 2 class case.

The labeled data shown in Figure 4.17 consist of 11 objects, each of which has one of the c = 3 crisp labels shown in the upper portion of the figure. With the Euclidean norm and k = 6 nearest neighbors having c = 3 class labels, the point **z** will be labeled (and subsequently colored) as a class 2 point, because 3 of its nearest 6 Euclidean neighbors (the ones inside the circular disk centered at **z**) have this crisp label.

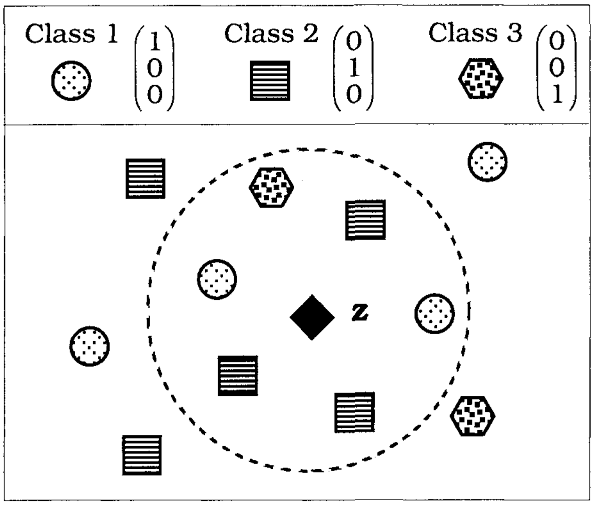

Class 1
$\begin{pmatrix}1\\0\\0\end{pmatrix}$
Class 2
$\begin{pmatrix}0\\1\\0\end{pmatrix}$
Class 3
$\begin{pmatrix}0\\0\\1\end{pmatrix}$
z

Figure 4.17 Geometric idea of the crisp k-nn rule classifier

Let $\{\mathbf{x}_{(1)},\ldots,\mathbf{x}_{(k)}\}$ be the nearest neighbors of **z**, arranged in order of ascending distance, i.e., $\delta(\mathbf{z},\mathbf{x}_{(1)}) \le,\ldots,\le \delta(\mathbf{z},\mathbf{x}_{(k)})$; and let $n_{(i)}(\mathbf{z})$ be the number of neighbors of **z** with label $\mathbf{e}_{(i)}$, i = 1,...,c. Then the *crisp k-nn rule classifier* $\mathbf{D}_{\text{hnn};k,\delta}$ can be written formally as

$$\text{Decide } \mathbf{z} \in (i) \Leftrightarrow \mathbf{D}_{\text{hnn};k,\delta}(\mathbf{z}) = \mathbf{e}_{(i)} \Leftrightarrow n_{(i)}(\mathbf{z}) = \underbrace{\max}_{(k)}\{n_{(k)}(\mathbf{z})\}. \tag{4.38}$$

In (4.38) ties are broken arbitrarily. Generalization of the crisp k-nn rule vote begins with an alternative way to compute (4.38). First we consider the weighted sum

$$\hat{\mathbf{D}}_{nn;k,\delta}(\mathbf{z}) = \left(\frac{\sum_{i=1}^{c} n_{(i)}(\mathbf{z})\mathbf{e}_{(i)}}{\sum_{j=1}^{c} n_{(j)}(\mathbf{z})} \right) \quad , \tag{4.39}$$

Since $\sum_{j=1}^{c} n_{(j)}(\mathbf{z}) = k$, $\hat{\mathbf{D}}_{nn;k,\delta}(\mathbf{z})$ is a convex combination of crisp label vectors, so it is a point in N_{fc}. In our view formula (4.39) always produces fuzzy label vectors, but statistics aficionados will disagree, interpreting (4.39) as a probabilistic label vector because Cover and Hart (1967) proved that these labels converge to Bayesian labels. Regardless of your bias, $\hat{\mathbf{D}}_{nn;k,\delta}$ is, formally anyway, either a probabilistic or fuzzy classifier function, even though the labels and reasoning leading to it are crisp. For example, equation (4.39) for the situation in Figure 4.17 yields

$$\hat{\mathbf{D}}_{nn;k,\delta}(\mathbf{z}) = \frac{2\begin{pmatrix}1\\0\\0\end{pmatrix} + 3\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\0\\1\end{pmatrix}}{6} = \begin{pmatrix}0.33\\0.50\\0.17\end{pmatrix} \quad . \tag{4.40}$$

Notice that $n_{(i)}(\mathbf{z}) = \underbrace{\max}_{(k)}\{n_{(k)}(\mathbf{z})\} = 3$ is just the coefficient that multiplies the crisp class 2 label vector, the majority class in the neighborhood of **z** shown in Figure 4.17. The decision rendered by the crisp k-nn rule can be realized by applying **H** at (1.15) to the result in (4.40). Thus,

$$\mathbf{D}_{hnn;k,\delta}(\mathbf{z}) = \mathbf{H}(\hat{\mathbf{D}}_{nn;k,\delta}(\mathbf{z})) \tag{4.38'}$$

is equivalent to (4.38). Formula (4.40) is not needed to crisply label **z**; it is simply convenient for computer implementation of the crisp k-nn rule (convenient, but at the cost of more computation, so if you are only interested in the crisp k-nn rule, this is not the most efficient way to implement it). However, the construction at (4.40) shows how to arrive at a truly fuzzy k-nn design. Suppose the six neighbors of **z** shown in Figure 4.17 had fuzzy label vectors as follows:

$$\overbrace{\begin{matrix}\mathbf{x}_1 & \mathbf{x}_2\\ \begin{pmatrix}0.9\\0.0\\0.1\end{pmatrix} & \begin{pmatrix}0.9\\0.1\\0.0\end{pmatrix}\end{matrix}}^{\textit{class 1}} \quad \overbrace{\begin{matrix}\mathbf{x}_3 & \mathbf{x}_4 & \mathbf{x}_5\\ \begin{pmatrix}0.3\\0.6\\0.1\end{pmatrix} & \begin{pmatrix}0.03\\0.95\\0.02\end{pmatrix} & \begin{pmatrix}0.2\\0.8\\0.0\end{pmatrix}\end{matrix}}^{\textit{class 2}} \quad \overbrace{\begin{matrix}\mathbf{x}_6\\ \begin{pmatrix}0.3\\0.0\\0.7\end{pmatrix}\end{matrix}}^{\textit{class 3}} \quad . \tag{4.41}$$

The generalization of simple majority voting in the crisp case is to assign **z** to the class in which it has the highest membership. Using formula (4.40) with these labels results in

$$\mathbf{D}_{\text{fnn};k,\delta}(\mathbf{z}) = \frac{\begin{pmatrix}0.9\\0.0\\0.1\end{pmatrix}+\begin{pmatrix}0.9\\0.1\\0.0\end{pmatrix}+\begin{pmatrix}0.3\\0.6\\0.1\end{pmatrix}+\begin{pmatrix}0.03\\0.95\\0.02\end{pmatrix}+\begin{pmatrix}0.2\\0.8\\0.0\end{pmatrix}+\begin{pmatrix}0.3\\0.0\\0.7\end{pmatrix}}{6} = \begin{pmatrix}0.44\\0.41\\0.15\end{pmatrix}, \quad (4.42)$$

and $\mathbf{D}_{\text{fnn};k,\delta}$ is a fuzzy label vector produced by fuzzy labels. Consequently, $\mathbf{D}_{\text{fnn};k,\delta}$ is a *fuzzy k-nn rule classifier.* There are many other ways to generalize (4.38), so this cannot be interpreted as *the* fuzzy k-nn rule - rather, it is one of many possible designs. Applying **H** at (1.15) to the label vector in (4.42) results in this fuzzy 6-nn rule assigning **z** the crisp label for class 1 instead of class 2, the label produced by the crisp 6-nn rule for the data in Figure 4.17.

If any label vector in (4.41) were possibilistic, the calculation at (4.42) would result in a possibilistic label vector for **z**. Consequently, we have described fuzzy ($\mathbf{D}_{\text{fnn};k,\delta}$) and possibilistic ($\mathbf{D}_{\text{pnn};k,\delta}$) versions of $\mathbf{D}_{\text{hnn};k,\delta}$. Since there are many other implementations that can also be called fuzzy and possibilistic k-nn designs, we call the algorithms summarized in Table 4.19 a set of "basic" k=nn rules.

Table 4.19 Basic crisp, fuzzy and possibilistic k-nn rules

Store	Labeled data $X = X_{tr} \subset \Re^p$, $\lvert X\rvert = n$ Label matrix U_{tr} of X_{tr}, $\mathbf{U}_{tr,j} \in N_{pc}$, $j = 1,\ldots,n$
Pick	☛ k = # of nn's to find ☛ $\delta: \Re^p \times \Re^p \mapsto \Re^+$ = metric on $\Re^p$
Given	To label : **z** in $\Re^p$
Compute	The distances $\{\delta_j \equiv \delta(\mathbf{z},\mathbf{x}_j): j = 1,\ldots,n\}$
Rank	$\underbrace{\delta_{(1)} \le \delta_{(2)} \le \ldots \le \delta_{(k)}}_{\text{k-nn indices}} \le \delta_{(k+1)} \le \ldots \le \delta_{(n)}$
Compute	$\mathbf{D}_{\text{pnn};k,\delta}(\mathbf{z}) = \frac{1}{k}\left(\sum_{i=1}^{k} \mathbf{U}_{tr,(i)}\right) \in N_{pc}$ (4.43)
Optional (Harden)	$\mathbf{D}_{H(\text{pnn});k,\delta}(\mathbf{z}) = \mathbf{H}\left(\mathbf{D}_{\text{pnn};k,\delta}(\mathbf{z})\right) \in N_{hc}$

Example 4.6 We illustrate the basic k-nn rules with the apples and pears data listed in Table 4.1 and plotted in Figure 4.6. Figure 4.18 shows a shaded disk with radius $\|\mathbf{x}_{13} - \mathbf{z}\|_2 = 0.46$ centered at $\mathbf{z}$ using Euclidean distance for δ. The disk captures three neighbors - $\mathbf{x}_{11}$, $\mathbf{x}_{13}$ and $\mathbf{x}_{14}$ - labeled pears and two neighbors - $\mathbf{x}_2$ and $\mathbf{x}_8$ - labeled apples in Table 4.1. The crisp 5-nn rule labels $\mathbf{z}$ a pear,

$$\hat{\mathbf{D}}_{\mathrm{nn};5,\delta_2}(\mathbf{z}) = \frac{2\begin{pmatrix}1\\0\end{pmatrix} + 3\begin{pmatrix}0\\1\end{pmatrix}}{5} = \begin{pmatrix}0.4\\0.6\end{pmatrix} \Rightarrow \mathbf{D}_{\mathrm{hnn};5,\delta_2}(\mathbf{z}) = \mathbf{H}\left[\begin{pmatrix}0.4\\0.6\end{pmatrix}\right] = \begin{pmatrix}0\\1\end{pmatrix}.$$

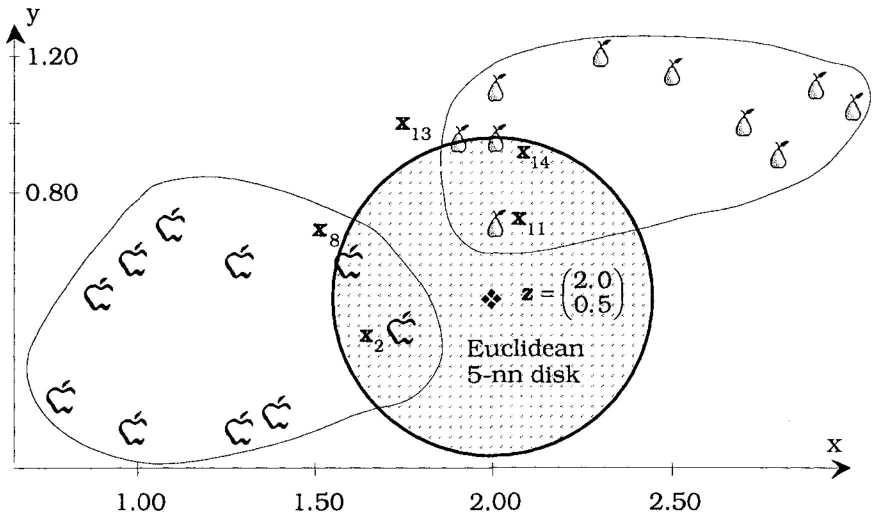


Figure 4.18 A crisp k-nn rule on the apples and pears data

To see that k and δ affect the crisp decision, Table 4.20 shows the distances from the point $\mathbf{z} = (2.0, 0.5)^T$ in Figure 4.6 to each of the 20 data points listed in Table 4.1 (whose coordinates and labels are repeated here for convenience).

Distances from $\mathbf{z}$ to each of its five nearest neighbors in the three norms are shown in Table 4.21, where $\mathbf{x}_{(j)}$ is the j-th ranked nearest neighbor to $\mathbf{z}$ and $U(\mathbf{x}_{(j)})$ is the crisp label for $\mathbf{x}_{(j)}$. Whenever there is a tie, the label assigned to $\mathbf{z}$ is arbitrary. There are *two* possible kinds of ties: label ties (U-ties), and distance ties (δ-Ties).

Table 4.20 Distances from z to the 20 points in Table 4.1

e_i	$\mathbf{x}_i$	x_i	y_i	$\delta_1(\mathbf{z}, \mathbf{x})$	$\delta_2(\mathbf{z}, \mathbf{x})$	$\delta_\infty(\mathbf{z}, \mathbf{x})$
🍎	1	1.00	0.60	1.10	1.00	1.00
🍎	2	1.75	0.40	0.35	0.27	0.25
🍎	3	1.30	0.10	1.10	0.81	0.70
🍎	4	0.80	0.20	1.50	1.24	1.20
🍎	5	1.10	0.70	1.10	0.92	0.90
🍎	6	1.30	0.60	0.80	0.71	0.70
🍎	7	0.90	0.50	1.10	1.10	1.10
🍎	8	1.60	0.60	0.50	0.41	0.40
🍎	9	1.40	0.15	0.95	0.69	0.60
🍎	10	1.00	0.10	1.40	1.08	1.00
🍐	11	2.00	0.70	0.20	0.20	0.20
🍐	12	2.00	1.10	0.60	0.60	0.60
🍐	13	1.90	0.95	0.55	0.46	0.45
🍐	14	2.00	0.95	0.45	0.45	0.45
🍐	15	2.30	1.20	1.00	0.76	0.70
🍐	16	2.50	1.15	1.15	0.82	0.65
🍐	17	2.70	1.00	1.20	0.86	0.70
🍐	18	2.90	1.10	1.50	1.08	0.90
🍐	19	2.80	0.90	1.20	0.89	0.80
🍐	20	3.00	1.05	1.55	1.14	1.00

Table 4.21 Crisp labels with $\mathbf{D}_{\text{hnn};k,\delta}(\mathbf{z})$ as a function of k and δ

k	$\mathbf{x}_{(k)}$	$U(\mathbf{x}_{(k)})$	$\delta_1(\mathbf{z}, \mathbf{x}_{(k)})$	$U(\mathbf{z})$
1	$\mathbf{x}_{11}$	🍐	0.20	🍐
2	$\mathbf{x}_{2}$	🍎	0.35	U-Tie
3	$\mathbf{x}_{14}$	🍐	0.45	🍐
4	$\mathbf{x}_{8}$	🍎	0.50	U-Tie
5	$\mathbf{x}_{13}$	🍐	0.55	🍐
k	$\mathbf{x}_{(k)}$	$U(\mathbf{x}_{(k)})$	$\delta_2(\mathbf{z}, \mathbf{x}_{(k)})$	$U(\mathbf{z})$
1	$\mathbf{x}_{11}$	🍐	0.20	🍐
2	$\mathbf{x}_{2}$	🍎	0.27	U-Tie
3	$\mathbf{x}_{8}$	🍎	0.41	🍎
4	$\mathbf{x}_{14}$	🍐	0.45	U-Tie
5	$\mathbf{x}_{13}$	🍐	0.46	🍐
k	$\mathbf{x}_{(k)}$	$U(\mathbf{x}_{(k)})$	$\delta_\infty(\mathbf{z}, \mathbf{x}_{(k)})$	$L(\mathbf{z})$
1	$\mathbf{x}_{11}$	🍐	0.20	🍐
2	$\mathbf{x}_{2}$	🍎	0.25	U-Tie
3	$\mathbf{x}_{8}$	🍎	0.40	🍐
4	$\mathbf{x}_{14}$	🍐	0.45	δ-Tie
5	$\mathbf{x}_{13}$	🍐	0.45	🍐

The nearest neighbor, $\mathbf{x}_{(1)} = \mathbf{x}_{11}$ is closest to $\mathbf{z}$ in all three distances, and labels it a pear. All three rules yield a label tie using k = 2, so either label may be assigned to $\mathbf{z}$ by these three classifiers, depending on the outcome of the tie-breaking rule employed. For k = 3 the 1 and sup norm distances label $\mathbf{z}$ a pear, but the 2 norm labels it an apple. At k = 4 the sup norm has a distance tie between $\mathbf{x}_{(4)}$ and $\mathbf{x}_{(5)}$, but both points are labeled pear so the decision is pear regardless of how the tie is resolved. This illustrates that the label assigned by $\mathbf{D}_{\text{hnn};k,\delta}$ depends on both k and δ; the five nearest neighbors are *not* ranked in the same order by all three distances.

If we apply FCM and PCM to these data (c = m = 2, both norms Euclidean, ε = 0.01, initialization random for HCM and FCM, PCM initialized with the terminal prototypes from FCM, PCM weights $w_1 = 0.15$, $w_2 = 0.16$), we get terminal label vectors for each of the 20 points that are fuzzy or possibilistic, respectively. Using the FCM labels for the five nearest neighbors to $\mathbf{z}$ yields

$$\mathbf{D}^{\text{FCM}}_{\text{fnn};5,\delta_2}(\mathbf{z}) = \frac{\begin{pmatrix}0.33\\0.67\end{pmatrix}+\begin{pmatrix}0.77\\0.23\end{pmatrix}+\begin{pmatrix}0.41\\0.59\end{pmatrix}+\begin{pmatrix}0.10\\0.90\end{pmatrix}+\begin{pmatrix}0.84\\0.16\end{pmatrix}}{5} = \begin{pmatrix}0.49\\0.51\end{pmatrix}, \text{ so}$$

$$\mathbf{H}\begin{pmatrix}0.49\\0.51\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} = \mathbf{e}_2 \Rightarrow \mathbf{z} = \text{pear}.$$

PCM labels for these five points would result in the same decision here but this is not always the case. The 20-nn rules based on all 20 crisp (given or HCM, which produces the given labels), FCM and PCM labels yield

$$\mathbf{H}\left(\mathbf{D}^{\text{HCM}}_{\text{hnn};20,\delta_2}(\mathbf{z}) = \frac{1}{20}\left[\sum_{j=1}^{20}\mathbf{U}_{\text{HCM}(j)}\right]\right) = \mathbf{H}\begin{pmatrix}0.50\\0.50\end{pmatrix} \Rightarrow \text{tie} \quad ; \tag{4.44a}$$

$$\mathbf{H}\left(\mathbf{D}^{\text{FCM}}_{\text{fnn};20,\delta_2}(\mathbf{z}) = \frac{1}{20}\left[\sum_{j=1}^{20}\mathbf{U}_{\text{FCM}(j)}\right]\right) = \mathbf{H}\begin{pmatrix}0.52\\0.48\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} \Rightarrow \mathbf{z} = \text{apple} \tag{4.44b}$$

$$\mathbf{H}\left(\mathbf{D}^{\text{PCM}}_{\text{pnn};20,\delta_2}(\mathbf{z}) = \frac{1}{20}\left[\sum_{j=1}^{20}\mathbf{U}_{\text{PCM}(j)}\right]\right) = \mathbf{H}\begin{pmatrix}0.33\\0.31\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} \Rightarrow \mathbf{z} = \text{apple} \tag{4.44c}$$

Equations (4.44) show that using all 20 nn's to $\mathbf{z}$ results, for all three classifiers, in the label switching from pear for k = 5 to apple for k = 20 (up to resolution of the tie in (4.44a)). An important point is that if all 20 labels are used, the rule-based on crisp labels is ambiguous, while the fuzzy and possibilistic based rules both label **z** an apple.

Terminology Even though the final outputs in (4.44b) and (4.44c) are crisp, some writers refer to the overall crisp decision as the result of a fuzzy or possibilistic *k*-nn rule. More properly, perhaps, a *fuzzy k-nn rule* is an algorithm that produces the fuzzy labels which are subsequently hardened. Similarly, we regard the *input* or *argument* of **H** in (4.44c) as the output of a *possibilistic k-nn rule.*

Example 4.7 As an example of the utility of the k-nn rule on a real problem, we revisit the segmentation of MR imagery to demonstrate that excellent classification results can be obtained from a very simple algorithm such as the crisp k-nn rule.

Views (a)-(c) of Figure 4.19 show three MRIs of the T1, T2 and ρ slices, respectively, of a patient who has a tumor in the upper right-central portion of the brain. The circular dot in the lower right side of these views is for color registration.

Figure 4.19(d) is a segmentation of the images made by having a human operator select training sets $(X_{tr,i})$ of pixels in each of i=1,...,7 tissue regions, and then assigning each vector in the training subsets a crisp tissue label. An operator usually labels a very few pixels from each tissue class (on the order of 100 pixels per class).

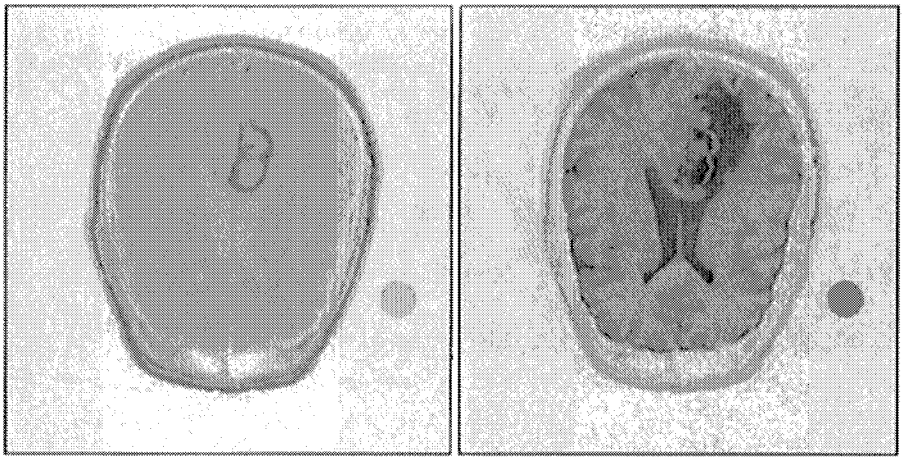
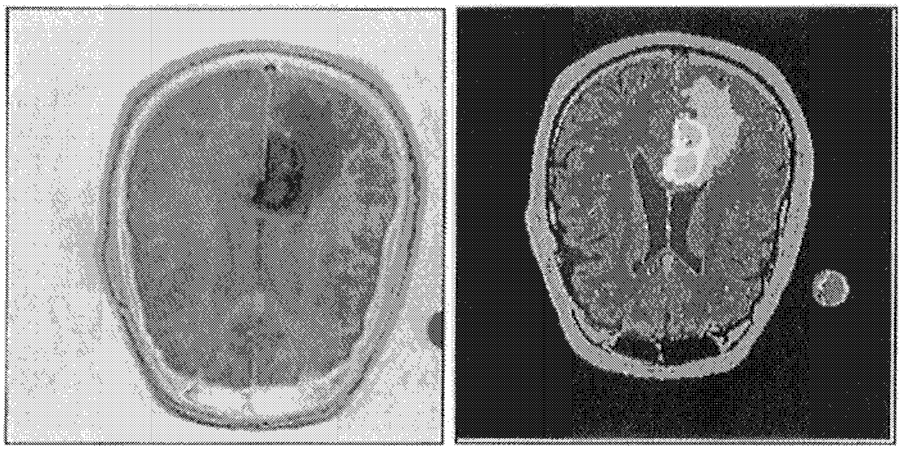

(a) MR Data - T1 Image **(b) MR Data - T2 Image**

Figure 4.19 Segmentation of an MR Image with the crisp 5-nn rule



(c) MR Data - ρ Image **(d) Segmentation via 5-nn rule**

Figure 4.19 (con't.) Segmentation of an MR Image with the crisp 5-nn rule

The remainder of the pixels were then labeled using the crisp 5-nn rule (4.38). The tumor, comprising the regions of lightest gray and white in Figure 4.19(d), is visually well defined in this segmentation of the three dimensional data. It may seem surprising that the k-nn rule yields good segmentations with a relatively small subset of labeled data, but this method is often rated as the best of many supervised techniques for image segmentation (Bezdek et al., 1997a).

There are many generalizations of the k-nn rules (Dasarathy, 1990). One important class of extensions is the *inverse distance weighted* (IDW) k-nn algorithms which modify $\mathbf{D}_{pnn;k,\delta}$ by replacing the fixed weights of 1/k in Table 4.19, equation (4.43), which simply average the k-nn labels, with (normalized) weights that are inversely proportional to the distances of the k neighbors to **z**. The idea is that neighbors that are closer in feature space (as measured by the distance metric) should exert more influence in the generation of the membership vector for the point being labeled. The *IDW k-nn algorithm* (Keller et al., 1985) is typical of this type of extension and uses the following equation instead of the one given in (4.43),

$$\mathbf{D}^{IDW,m}_{pnn;k,\delta}(\mathbf{z}) = \frac{\sum_{j=1}^{k}\left(\delta(\mathbf{z},\mathbf{x}_{(j)})\right)^{\frac{-2}{m-1}}\mathbf{U}_{tr,(j)}}{\sum_{s=1}^{k}\left(\delta(\mathbf{z},\mathbf{x}_{(s)})\right)^{\frac{-2}{m-1}}} \in N_{pc} \quad . \tag{4.45}$$

Equation (4.45) contains a fuzzifier value, $m \geq 1$, as in the FCM, PCM and GLVQ-F models, and is the basis of the crisp, fuzzy and possibilistic IDW k-nn rules. Like (4.43), equation (4.45) will produce a fuzzy label vector even when the training data have crisp labels. The choice m = 3 simplifies (4.45) considerably. For example, suppose the 6 nearest neighbors to $\mathbf{z} = (0,0)^T$ in Figure 4.17 are $\mathbf{x}_1 = (2,0)^T$, $\mathbf{x}_2 = (-1.5,1)^T$, $\mathbf{x}_3 = (1,2)^T$, $\mathbf{x}_4 = (1,-2)^T$, $\mathbf{x}_5 = (-1,-1)^T$ and $\mathbf{x}_6 = (-0.5,3)^T$. Using (4.41) and $\mathbf{D}^{\mathrm{IDW},3}_{\mathrm{fnn};k,\delta}$ instead of $\mathbf{D}_{\mathrm{fnn};6,\delta_2}$ in (4.42) with m = 3, we calculate

$$\frac{0.5\begin{pmatrix}0.9\\0.0\\0.1\end{pmatrix}+0.56\begin{pmatrix}0.9\\0.1\\0.0\end{pmatrix}+0.42\left[\begin{pmatrix}0.3\\0.6\\0.1\end{pmatrix}+\begin{pmatrix}0.03\\0.95\\0.02\end{pmatrix}\right]+0.71\begin{pmatrix}0.2\\0.8\\0.0\end{pmatrix}+0.33\begin{pmatrix}0.3\\0.0\\0.7\end{pmatrix}}{0.5+0.56+0.42+0.42+0.71+0.33},$$

$$\text{so } \mathbf{D}^{\mathrm{IDW},3}_{\mathrm{fnn};k,\delta}(\mathbf{z}) = \begin{pmatrix}0.45\\0.44\\0.11\end{pmatrix} . \tag{4.46}$$

As in (4.42), the inverse weighted distance fuzzy 6-nn rule will assign **z** to class 1 upon hardening of (4.46). The class memberships in (4.46) for classes 1 and 2 are a little closer than they are in (4.42), so the IDW rule indicates that **z** is in a region of uncertainty between classes 1 and 2 a little more strongly than simple averaging does.

Example 4.8 To illustrate the difference between the basic and IDW versions of the k-nn rules, we repeat some of the calculations of Example 4.6 using (4.45) instead of (4.43) for various m's and δ's.

Table 4.22 $\mathbf{D}^{\mathrm{IDW},3}_{\mathrm{hnn};k,\delta}$ on the apples-pears data using crisp labels

$\mathbf{H}(\mathbf{D}^{\mathrm{IDW},3}_{\mathrm{hnn};k,\delta_1})$ with m = 3 and δ = 1-norm

k	$\mathbf{x}_{(k)}$	$U(\mathbf{x}_{(k)})$	$\delta_1^{-1}(\mathbf{z},\mathbf{x}_{(k)})$	u(🍐, **z**)	u(🍎, **z**)	U(**z**)
1	$\mathbf{x}_{11}$	🍐	5.00	1.00	0.00	🍐
2	$\mathbf{x}_{2}$	🍎	2.86	0.64	0.36	🍐
3	$\mathbf{x}_{14}$	🍐	2.22	0.72	0.28	🍐
4	$\mathbf{x}_{8}$	🍎	2.00	0.60	0.40	🍐
5	$\mathbf{x}_{13}$	🍐	1.82	0.65	0.35	🍐

Table 4.22 (con't.) $\mathbf{D}^{\mathrm{IDW},3}_{\mathrm{hnn};k,\delta}$ on the apples-pears data using crisp labels

$\mathbf{H}(\mathbf{D}^{\mathrm{IDW},3}_{\mathrm{hnn};k,\delta_2})$ with m = 3 and δ = 2-norm

k	$\mathbf{x}_{(k)}$	$U(\mathbf{x}_{(k)})$	$\delta_2^{-1}(\mathbf{z},\mathbf{x}_{(k)})$	u(🍐, **z**)	u(🍎, **z**)	U(**z**)
1	$\mathbf{x}_{11}$	🍐	5.00	1.00	0.00	🍐
2	$\mathbf{x}_{2}$	🍎	3.70	0.57	0.43	🍐
3	$\mathbf{x}_{8}$	🍎	2.44	0.45	0.55	🍎
4	$\mathbf{x}_{14}$	🍐	2.22	0.54	0.46	🍐
5	$\mathbf{x}_{13}$	🍐	2.17	0.60	0.40	🍐

$\mathbf{H}(\mathbf{D}^{\mathrm{IDW},3}_{\mathrm{fnn};k,\delta_\infty})$ with m = 3 and δ = sup-norm

k	$\mathbf{x}_{(k)}$	$U(\mathbf{x}_{(k)})$	$\delta_\infty^{-1}(\mathbf{z},\mathbf{x}_{(k)})$	u(🍐, **z**)	u(🍎, **z**)	U(**z**)
1	$\mathbf{x}_{11}$	🍐	5.00	1.00	0.00	🍐
2	$\mathbf{x}_{2}$	🍎	4.00	0.55	0.45	🍐
3	$\mathbf{x}_{8}$	🍎	2.50	0.43	0.57	🍎
4	$\mathbf{x}_{14}$	🍐	2.22	0.53	0.47	🍐
5	$\mathbf{x}_{13}$	🍐	2.22	0.59	0.41	🍐

Table 4.22 shows the results of applying the IDW k-nn rules to the apples and pears data with the Euclidean norm for m = 3. Hardening in the last column of the tables in this example is done via (1.15), although the memberships themselves could be used in later processing. Notice that the ties that were recorded in Table 4.21 for the crisp k-nn rule using the same three norms disappear because the memberships induced by inverse distance weighting are distinct, even in the situation where the training data has crisp labels. With the 2-norm, the resultant class memberships are closer, even though the final crisp label for **z** is the same as in the crisp k-nn.

The results of using IDW k-nn classification with the 2-norm fixed for two values of m other than m = 3 are displayed in Table 4.23 to give an idea of the effect of this parameter in the membership calculations. Compare the two blocks in Table 4.23 with the center block in Table 4.22 to see changes as m goes from 2 to 3 to 5. For a relatively small value of m, i.e., m = 2, all hardened class assignments are to class 🍐, even in the 3-nn case where two of the closest neighbors (a majority!) are from the 🍎 class. Table 4.23 shows what is expected - that smaller values of m tend to magnify the effect of the closer points, whereas larger values of m produce the opposite result.

Table 4.23 $D^{IDW,m}_{fnn;k,\delta_2}$ on the apples-pears data using crisp labels

$H(D^{IDW,2}_{fnn;k,\delta_2})$ with m = 2 and δ = 2-norm

k	$x_{(k)}$	$U(x_{(k)})$	$\delta_2^{-2}(z, x_{(k)})$	u(🍐, z)	u(🍎, z)	U(z)
1	x_{11}	🍐	25.00	1.00	0.00	🍐
2	x_2	🍎	13.69	0.65	0.35	🍐
3	x_8	🍎	5.95	0.56	0.44	🍐
4	x_{14}	🍐	4.93	0.60	0.40	🍐
5	x_{13}	🍐	4.71	0.80	0.20	🍐

$H(D^{IDW,5}_{fnn;k,\delta_2})$ with m = 5 and δ = 2-norm

k	$x_{(k)}$	$U(x_{(k)})$	$\delta_2^{-0.5}(z, x_{(k)})$	u(🍐, z)	u(🍎, z)	U(z)
1	x_{11}	🍐	2.24	1.00	0.00	🍐
2	x_2	🍎	1.92	0.54	0.46	🍐
3	x_8	🍎	1.56	0.39	0.61	🍎
4	x_{14}	🍐	1.49	0.52	0.48	🍐
5	x_{13}	🍐	1.47	0.60	0.40	🍐

Finally, there is no reason that the training data must have crisp labels. In fact, the use of fuzzy or possibilistic labels for the training data is perhaps the real advantage of the soft k-nn rules. Table 4.24 gives a simple example of this where the neighbors have been assigned (subjectively, by us) the fuzzy labels in columns 3 and 4.

Table 4.24 Using fuzzy labels : $D^{IDW,3}_{fnn;k,\delta_2}$ with m = 3 and δ = 2-norm

k	$x_{(k)}$	u(🍐, $x_{(k)}$)	u(🍎, $x_{(k)}$)	$\delta_2^{-1}(z, x_{(k)})$	u(🍐, z)	u(🍎, z)	U(z)
1	x_{11}	0.70	0.30	5.00	1.00	0.00	🍐
2	x_2	0.20	0.80	3.70	0.49	0.51	🍎
3	x_8	0.25	0.75	2.44	0.44	0.56	🍎
4	x_{14}	0.75	0.25	2.22	0.49	0.51	🍎
5	x_{13}	0.70	0.30	2.17	0.52	0.48	🍐

For the 2-nn and 4-nn cases in Table 4.24, the final crisp label for **z** switches from 🍐 to 🍎. This is due to the stronger memberships for $\mathbf{x}_2$ and $\mathbf{x}_8$ in the 🍎 class and the "weakening" of the effect of the closest neighbor, $\mathbf{x}_{11}$, in the 🍐 class. While this example is contrived, it points out that if meaningful fuzzy labels can be assigned to the training data of a nearest neighbor classifier, the resultant fuzzy labels for unknown points will reflect that partial commitment.

Section 4.7.I will describe one way that soft labels can be generated for training data. Of course, you can run any clustering algorithm such as FCM or PCM on the data and simply ignore the given crisp labels. This yields fuzzy or possibilistic labels for the points in X, but it is arguable whether this is a plausible strategy. Some feel that rejecting given crisp labels constitutes a loss of known information, while others support the idea that the structure of the data itself (as discovered by a "reliable" clustering algorithm) is more important in determining useful labels for k-nn designs. Both camps have good points.

4.5 The Fuzzy Integral

The fuzzy integral is a numeric-based approach which has been used for both pattern classification and image segmentation (Keller et al., 1986, Tahani and Keller, 1990, Keller and Krishnapuram, 1994, Keller et al., 1994a, Grabisch and Nicolas, 1994). It uses a hierarchical network of evidence sources to arrive at a confidence value for a particular hypothesis or decision. A distinguishing characteristic of the fuzzy integral is that it utilizes information concerning the worth or importance of the sources in the decision making process.

The fuzzy integral is a nonlinear approach to combining multiple sources of uncertain information as often happens in automated pattern recognition. In these applications the integral is evaluated over a set of information sources (sensors, algorithms, features, etc.) and the function being integrated supplies a confidence value for the object under consideration in a particular hypothesis or class from the standpoint of each individual source of information.

The fuzzy integral relies on the concept of a fuzzy measure (Sugeno (1977), Dubois and Prade (1982), Wang and Klir (1992)) which generalizes the concept of a probability measure. A *fuzzy measure* (FM) over a set X with power set $\wp(X)$ is a function $g:\wp(X) \mapsto [0,1]$ such that $\forall\, A, B, A_i \in \wp(X)$,

$$g(\emptyset) = 0; g(X) = 1 \qquad ; \qquad \text{(FM1)}$$

$$g(B) \geq g(A) \text{ if } B \supset A \qquad ; \qquad \text{(FM2)}$$

If $\{A_i\}_{i=1}^{\infty}$ is monotonic, then $\underbrace{\lim}_{i\to\infty}\{g(A_i)\} = g(\bigcup_{i=1}^{\infty} A_i)$. (FM3)

When X is finite, (FM3) holds trivially. A particularly useful class of fuzzy measures is due to Sugeno (1977). A fuzzy measure g_λ is called a *Sugeno* or *λ-fuzzy measure* if it satisfies (FM1-FM3) and the following additional property for some $\lambda > -1$:

$$\text{If } A \cap B = \varnothing, \text{ then } g_\lambda(A \cup B) = g_\lambda(A) + g_\lambda(B) + \lambda g_\lambda(A) g_\lambda(B). \tag{4.47}$$

If $\lambda = 0$ in (4.47) then g_λ is a *probability measure.* Suppose X is a finite set of information sources, $X = \{x_1, \dots, x_n\}$, and let $g^i = g_\lambda(\{x_i\})$. The values $g^1, g^2, \dots, g^n$, are called the *fuzzy densities* associated with X.

These densities are interpreted as the importance of the individual information sources. The measure of a set A of information sources is interpreted as the importance of that subset of sources toward answering a particular question (such as class membership).

Using these definitions we can show that $g_\lambda(A)$ can be constructed from the fuzzy densities of the elements of A for any subset A of X. Given the set of densities, the value of λ can be easily found as the unique root greater than -1 of the simple polynomial in (4.48) obtained from (4.47) and the fact that g(X) = 1 (Sugeno, 1977),

$$\lambda + 1 = \prod_{i=1}^{n} (1 + \lambda g^i) \quad . \tag{4.48}$$

Thus, estimating the densities is a core problem when using Sugeno (and some other classes of) fuzzy measures.

Sugeno measures are a large subset of all fuzzy measures. All belief and plausibility measures (Shafer, 1976) are Sugeno measures. Sugeno measures are useful because (4.47) provides a way to calculate the measure of a union of two sets from a pair of component measures. Other classes of fuzzy measures exhibit a similar computational advantage. For example, the traditional possibility measure has the defining property that $g_{poss}(A \cup B) = g_{poss}(A) \vee g_{poss}(B)$. A similar fuzzy measure can be defined with any T co-norm.

Still, there are many fuzzy measures which do not fit into nice classes, but which are useful in pattern recognition. The trick is to find a way to choose a measure that is "optimal" for a given problem.

Grabisch and Nicolas (1994) give some methods for learning useful general fuzzy measures from training data.

Let $Z = \{z_1, \ldots, z_n\}$ denote the objects to be classified. For each class hypothesis c_k, let $h_k : Z \times X \mapsto [0,1]$. The value $h_k(z_j, x_i)$ is called the *partial evaluation* or support for object z_j in class k from the perspective of information source x_i. When the context is clear, we suppress the object name and class label from the partial support function.

The information sources $X = \{x_1, \ldots, x_n\}$ could be a set of individual feature types or simple classifiers. The fuzzy measure, g, supplies the expected worth of each subset of sources for a classification hypothesis. The Sugeno fuzzy integral $S_g(h)$ of a function h over X with respect to g is defined using α-cuts of h, $h_\alpha = \{x : h(x) \geq \alpha\}$ as (Sugeno, 1977),

$$S_g(h) = \int_X h(x) \circ g = \underbrace{\sup}_{0 \leq \alpha \leq 1} \{\alpha \wedge g(h_\alpha)\} \quad . \tag{4.49}$$

In applications to pattern recognition, the computational cost of computing the confidence value $S_g(h)$ can be reduced significantly since the set of information sources is finite. If $X = \{x_1, \ldots, x_n\}$ is arranged so that $h(x_1) \geq h(x_2) \geq \ldots \geq h(x_n)$, then Sugeno (1977) showed that

$$S_g(h) = \vee_i [h(x_i) \wedge g(X_i)] \quad , \tag{4.50}$$

where $X_i = \{x_1, \ldots, x_i\}$ for $i = 1, \ldots, n$. This reduces the number of subsets needed to evaluate the fuzzy integral for each function h from 2^n to just n. Also, for a Sugeno measure g_λ, the values $\{g_\lambda(X_i)\}$ can be determined recursively from the densities as

$$g_\lambda(X_1) = g_\lambda(\{x_1\}) = g^1 \quad ; \tag{4.51a}$$

$$g_\lambda(X_i) = g_\lambda(X_{i-1} \cup \{x_i\}) = g_\lambda(X_{i-1}) + g^i + \lambda g_\lambda(X_{i-1}) \cdot g^i . \tag{4.51b}$$

Sorting the function h adds some complexity to the evaluation. For a general fuzzy measure, it is still possible to use look-up table methods to extract the appropriate n subsets to compute the integral. The reader is referred to Dubois and Prade (1982), Sugeno (1977), Grabisch et al. (1992), Wang and Klir (1992), and Grabisch et al. (1995, 1998) for more extensive theoretical background on fuzzy measures and the fuzzy integral.

The definition given by Sugeno (1977) for the fuzzy integral is not a proper extension of Lebesgue integration, in the sense that the Lebesgue integral is not recovered when the measure is additive. To avoid this drawback, Murofushi and Sugeno (1991) proposed the Choquet integral as an alternative, referring to a functional defined by Choquet in a different context. Let h be a function on X with values in [0,1] and g be a fuzzy measure. The Choquet integral $C_g(h)$ is

$$C_g(h) = \int_X h(x) \circ g = \int_0^1 g(h_\alpha)d\alpha \quad . \tag{4.52}$$

where again $h_\alpha = \{x : h(x) \geq \alpha\}$. If X is discrete, X = $\{x_1, \ldots, x_n\}$ and arranged so that $h(x_1) \geq h(x_2) \geq \ldots \geq h(x_n)$, then $C_g(h)$ can be computed as

$$C_g(h) = \sum_{i=1}^{n} g(X_i) \cdot \left[h(x_i) - h(x_{i+1})\right] \quad , \tag{4.53}$$

where $h(x_{n+1})$ is defined to be 0, and $X_i = \{x_1, \ldots, x_i\}$ for i = 1,..., n. It is also informative to write the discrete Choquet integral as a (nonlinear) weighted sum of these values in which the weights depend on their order. For i = 1, 2,..., n, assume $g(X_0) = 0$ and define

$$\omega_i(g) = g(X_i) - g(X_{i-1}) \quad . \tag{4.54}$$

Combining (4.53) and (4.54),

$$C_g(h) = \sum_{i=1}^{n} h(x_i) \cdot \omega_i(g) \quad . \tag{4.55}$$

In the general case, the sum in (4.55) is a nonlinear function of h because the ordering of the arguments depends upon the relative sizes of the values of the function h. This ordering can determine the values of the weights $\{\omega_i(g)\}$, and which products, $h(x_i) \cdot \omega_i(g)$, will be formed. As for the Sugeno integral, calculating the Choquet integral for a λ-fuzzy measure requires only the fuzzy densities. Assigning densities (on the entire fuzzy measure) appropriately is crucial for successful application of the fuzzy integral to pattern recognition.

Example 4.9 To display the mechanics of S_g and C_g, we compute the integrals for an object, z, whose class confidence (the h function) from the standpoint of 4 sources of information (perhaps features or other classifiers) is given in Table 4.25. Also listed are the densities assigned to each source for a λ-fuzzy measure.

Table 4.25 Class confidences and densities for fuzzy integrals

Source x_i	Confidence $h(x_i)$	Density g^i
1	0.9	0.2
2	0.7	0.2
3	0.4	0.2
4	0.3	0.2

Notice that $X = \{x_1, x_2, x_3, x_4\}$ is already sorted by decreasing h values. Even though all densities are equal, the fuzzy measure g is not a probability measure since the sum of the densities is less than 1. Solving equation (4.48) (by, for example, Newton's method) for λ gives $\lambda = 0.746$, and so, the 4 values of the measure that are needed to compute either fuzzy integral are generated by (4.51) and given in Table 4.26.

Table 4.26 Measure values to compute S_g and C_g for the data given in Table 4.25

Source Set X_i	Measure $g(X_i)$
X_1	0.200
X_2	0.430
X_3	0.695
X_4	1.000

For these values the two fuzzy integrals are

$S_g(h) = \vee(0.9 \wedge 0.2, 0.7 \wedge 0.43, 0.4 \wedge 0.695, 0.3 \wedge 1.0) = 0.43$; and

$C_g(h) = (0.9-0.7)(0.2) + (0.7-0.4)(0.43) + (0.4-0.3)(0.695) + (0.3-0.0)(1.0)$
$= 0.54$.

In comparison with probability theory, the fuzzy integral corresponds to the concept of expectation. Fuzzy integral values provide a different measure of certainty in a classification than posterior probabilities. Since the integral evaluation need not sum to one, lack of evidence and negative evidence can be distinguished. Dempster-Shafer belief theory (Shafer, 1976), can also distinguish between lack of evidence and negative evidence. A conceptual difference between the fuzzy integral and a Dempster-Shafer classifier is in the frame of discernment. For the fuzzy integral, the frame of discernment contains the knowledge sources related to the hypothesis under consideration, whereas with belief theory, the frame of discernment contains all of the possible hypotheses. The fuzzy integral can assess the importance of all groups of knowledge sources towards answering the questions as well as the degree to which each knowledge source supports the hypothesis.

We can view the action of a single fuzzy integral as a local filter on a set of values. For example, if the h function is just the scaled gray level in an image window, then applying the fuzzy integral to the window and replacing the gray level of the center pixel with the integral value induces a filter on the image. (Note that while we discuss filters on image windows - we can't help it because we like image processing - this discussion holds for any sequence of data values, for example, in signal processing). Selection of different fuzzy measures yields different types of filters. Several examples of fuzzy integral filters are given in the literature (Grabisch, 1994, Grabisch and Schmitt, 1995, Hocaoglu et al., 1997, Keller et al., 1998, Hocaoglu and Gader, 1998). We note a few for the Choquet integral.

Assume that all neighborhoods are of size n (neighborhoods are usually square regions of odd length centered at a point ξ). Here, n represents the total number of points in the neighborhood). If the measure g is an additive measure with all densities equal to 1/n, then the filter is the simple local average. Suppose n = 2k+1. If the measure, g_ξ is defined for any subset A of the window to be

$$g_\xi(A) = \begin{cases} 1 & \text{if } |A| \ge k \\ 0 & \text{else} \end{cases} \quad , \tag{4.56}$$

then the Choquet integral is the median filter. This is easy to see using equation (4.55) because ω_i will be nonzero for only one value of the index, which is the index required to "pick off" the median of the input values. In fact, replacing k with any i between 1 and 2k+1 in the above definition yields the i-th order statistic (including the maximum for i = 1 and the minimum for i = 2k+1). More generally, all stack filters (a class which includes the median filter and all other order statistic filters) can be represented by Choquet integral filters (Shi et al., 1998).

Choquet integral filters can also represent combinations of *linear order statistic* (LOS) filters defined by the convex sum

$$\text{LOS}_g(h) = \sum_{i=1}^{n} \omega_i h(x_i) \quad , \tag{4.57}$$

where the weights satisfy $\sum_{i=1}^{n} \omega_i = 1$ and the function values are sorted in descending order. This operator can be seen as a Choquet fuzzy integral filter by defining the measure g according to

$$\text{If } |A| = i, \text{ then } g(A) = \sum_{j=1}^{i} \omega_j \quad . \tag{4.58}$$

These filters can also be referred to as *ordered weight average* (OWA) filters since they implement the operator given that name by Yager (Yager, 1988). They have also been referred to as generalized order filters by Grabisch. They are useful for implementing robust estimators (Huber, 1981, Rousseeuw and Leroy, 1987), such as the alpha-trimmed mean (Shi et al., 1998).

Example 4.10 This example shows the use of a Choquet integral noise filter in an *automatic target recognition* application (Hocaoglu et al. 1997, Keller et al., 1998). (OK, this really belongs in Chapter 5, but hey, it seems like a good place to demonstrate the use of the fuzzy integral as a data filter.) Figure 4.20 shows a portion of a LADAR range image where the scaling has been performed artificially to give a clear picture of the convoy located in the middle. The figure shows 6 of the 9 targets in the image. The white rectangles enclosing the vehicles in the convoy in the image were inserted manually.

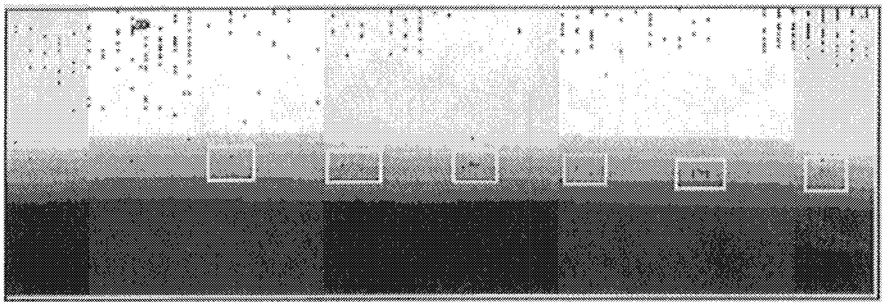
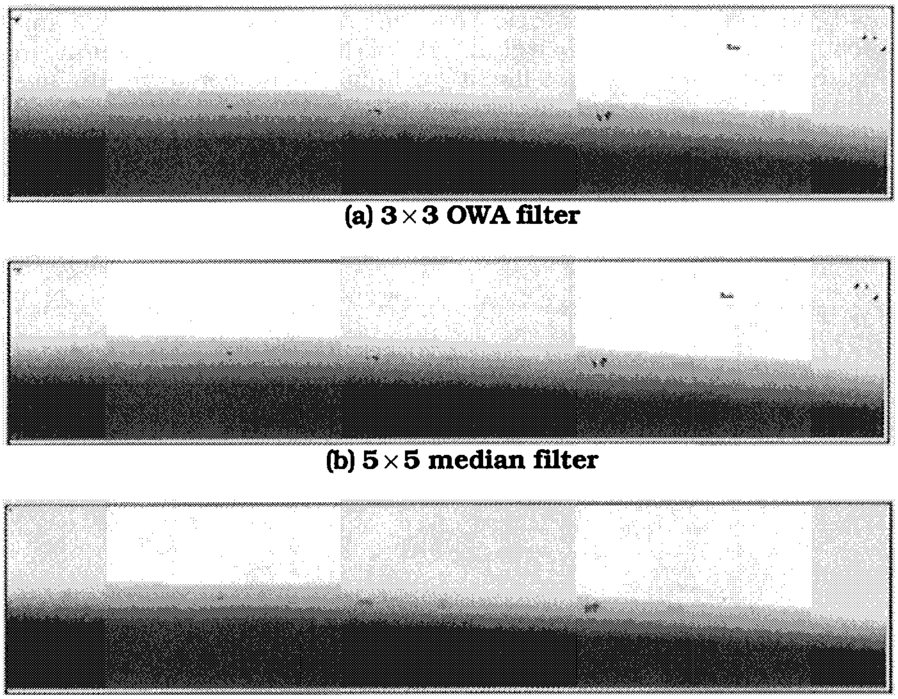


Figure 4.20 A (nonlinearly scaled) LADAR range image

Notice the noisy background caused by sensor dropout as well as other phenomena (although it may be somewhat hard to see the full extent of the noise in these small images). The original image was processed by three filters: (a) a standard 5 ×5 median filter, (b) a 3 × 3 OWA filter with weights 0, 0, 0, 0.25, 0.5, 0.25, 0, 0, 0, and (c) a 5 × 5 Choquet integral filter based on a λ-fuzzy measure with the densities described in Hocaoglu et al. (1997). Basically, the density for a given pixel, i.e., for the singleton set containing the pixel, in a window measures how similar this pixel's range value is with its neighbors. In all cases, the center pixel's range value is replaced by the value obtained from the filter. The Choquet integral filter preserved edge structure better than the other filters while smoothing the background more. (Note: a 3×3 OWA was used because the 5×5 OWA filter "looked" the same as the 5×5 median). Detailed discussion is provided in Hocaoglu et al. (1997).

Figure 4.21 shows the output of the three filters on the LADAR range image (nonlinearly scaled for display purposes to show the convoy). It's hard to see, but there is texture in the background for the OWA and median filters. The background in Choquet-filtered image is flat. The Choquet filter managed to remove some of the noise pixels that otherwise caused 3 false alarms. In any case, these examples illustrate the wide range of behaviors that can be obtained with Choquet integral filters by choosing different measures and classes of measures.


(a) 3×3 OWA filter

(b) 5×5 median filter

(c) 5×5 Choquet filter

Figure 4.21 Application of filters to LADAR range image

The fuzzy integral can be used in pattern recognition problems as follows. Information sources are identified. These sources could be individual features, pattern classifiers, context information, etc. A fuzzy measure is generated subjectively or estimated from training data for each pattern class. Generation of the measures is the training phase for the fuzzy integral approach. Given a pattern to be classified, an evidence function $h_i(x_j)$ is evaluated for each information source x_j and each class i. The functions $\{h_i\}$ are then integrated with respect to their corresponding class fuzzy measures resulting in one confidence value for each class. These confidence values are used to make a final classification decision, e.g., assign the pattern to the class with the highest confidence. The fuzzy integral approach is summarized in Table 4. 27.

Table 4.27 A fuzzy integral-based classifier

Given	☛ A set $X = \{x_1, \ldots, x_n\}$ of information sources ☛ To label : object **z** ☛ For $1 \le i \le c$, a function $h_i : X \to [0,1]$ which evaluates the strength of object **z** in class i with respect to x_j
Get	Densities $\{g_i^j : 1 \le i \le c;\ 1 \le j \le n\}$ for measures $\{g_i\}$, or the entire measures $\{g_i\}$.
Find	$h_i(x_j)$ for each source j and each class i for object **z**
Sort	X: $\{h_i(x_1) \ge h_i(x_2) \ge \ldots \ge h_i(x_n) : 1 \le i \le c\}$
Compute	$f_{g_i}(h_i) = \begin{cases} S_{g_i}(h_i) = \bigvee_{j=1}^{n} [h_i(x_j) \wedge g_i(X_j)] \quad \forall i\ ; \quad \text{or} \quad (4.59) \\ C_{g_i}(h_i) = \sum_{j=1}^{n} \left[h_i(x_j) - h_i(x_{j+1}) \right] \cdot g(X_j) \quad \forall i \quad (4.60) \end{cases}$
Array	$\mathbf{D}_{f_g}(\mathbf{z}) = (f_{g_1}(h_1(\mathbf{z})), \ldots, f_{g_c}(h_c(\mathbf{z})))^T \in N_{pc}$ (4.61) where $f = S$ or C and $g = (g_1, \ldots, g_c)$
Optional (Harden)	$\mathbf{H}(\mathbf{D}_{f_g}(\mathbf{z})) = \mathbf{e}_i \Leftrightarrow f_{g_i}(h_i(\mathbf{z})) \ge f_{g_j}(h_j(\mathbf{z})) \ \forall j \ne i$ (4.62)

Notice that the calculation in (4.61) results in a *possibilistic* label vector for **z** using either the Sugeno or Choquet fuzzy integral, so $\mathbf{D}_{f_g}$ is, in our terminology, a possibilistic classifier that depends on either the Sugeno or Choquet fuzzy integral. And when the option to harden is used, the resultant classifier at (4.62) is crisp. As with the k-nn rules, other authors sometimes call (4.61) a fuzzy classifier, but we feel that our terminology is technically correct.

Example 4.11 Tahani and Keller (1990) describe a fuzzy integral-based classifier that they developed for *automatic target recognition* (ATR). The classifier was developed and tested using *forward looking infrared* (FLIR) images containing two tanks and an *armored personnel carrier* (APC), Three sequences of 100 frames each were used for training. In each sequence the vehicles appeared at a different aspect angle to the sensor (0^o, 45^o, 90^o). In the fourth sequence the APC "circled" one of the tanks, moving in and out of a ravine and finally coming toward the sensor. This sequence was used to perform the comparison tests. The images were preprocessed to extract "object of interest" windows.

Classification level integration using S_{g_i} with λ-fuzzy measures was performed on four statistical features calculated from the windows, that is, the sources $\{x_1, x_2, x_3, x_4\}$ represent the {"mean", "variance", "skewness", "kurtosis"} of image neighborhoods. To get the partial evaluation, $h_k(x_i)$, (for k= tanks, armored personnel carriers = APCs), for each feature, the FCM algorithm with c = 2 was used on the training data. Normalized inverse distances to the terminal cluster centers produced memberships for the test objects.

The fuzzy densities - the degree of importance of each feature - were assigned by how well each feature separated the two classes (tank and APC) on the training data. These are shown, along with λ values, in Table 4.28.

Table 4.28 Computed Densities and λ values

	g^1	g^2	g^3	g^4	λ
Tank	0.16	0.23	0.19	0.22	0.760
APC	0.15	0.24	0.18	0.23	0.764

Table 4.29 compares the output results for three classifiers; the Sugeno fuzzy integral design $\mathbf{D}_{S_g}$; the standard Bayes classifier $\mathbf{D}_b$; and $\mathbf{D}_{DS}$, a classifier that uses Dempster-Shafer theory for integration of information (Wootton et al., 1988).

Table 4.29 Classification results for three classifiers

	Fuzzy Integral $\mathbf{D}_{S_g}$		**Bayes Classifier $\mathbf{D}_b$**		**Dempster-Shafer $\mathbf{D}_{DS}$**	
	Tank	APC	Tank	APC	Tank	APC
Tank	175	1	176	0	176	0
APC	17	49	22	44	22	44
	92.6% right		90.9% right		86.4% right	

Each 2×2 block of cells in Table 4.29 is the confusion matrix obtained by the classifier when applied to the test data (the fourth image sequence), hardened in the usual way. In this test at least, the classifier designed with fuzzy integrals did slightly better than the two probabilistic designs. In Tahani and Keller (1990) and Keller et al. (1994a) it was demonstrated, on the above data, that the fuzzy integral had the ability to fuse the outputs of three classifiers: a Bayes recognizer, a nearest prototype design based on exemplars from fuzzy c-means clustering, and a feature-level fuzzy integral. The densities were chosen heuristically based on individual classifier performance on a training set. The integration process was able to "correct" mistakes made by one of the classifiers, while maintaining the correct classifications for those objects where there was no confusion in the algorithmic outputs. In Keller et al. (1994a), the value of the fuzzy integral to fuse outputs of several neural network classifiers was nicely demonstrated on a very difficult handwritten character recognition problem. We will return to the issue of classifier fusion, or multistage classifiers, in Section 4.9.

The behavior of the fuzzy integral in a real problem is heavily dependent on the densities, or more generally, on the individual fuzzy measures. Therefore, estimation of the densities or the measures is very important. In some applications of the fuzzy integral the densities can be supplied subjectively by an expert. This subjective assignment approach may be the only method to assess the worth of non-numeric sources of information, such as context or "intelligence" reports. In most pattern recognition problems, it is preferable to estimate the densities directly from training data.

Given a set of n information sources, we either need to specify 2^n values directly (one for each subset), or for "nice" classes of measures, such as Sugeno λ-measures or possibility measures, we need only to generate n fuzzy densities. In some instances the measure can represent subjective information only. Hence, heuristic methods have been used to specify either the full measure or the densities. They can be directly produced by human experts, or can be inferred from training data in numerous ways. No general theory applies here. For example, Keller et al. (1986) and Qiu and Keller (1987) used the relative amount of overlap between the histograms of a feature for the various classes (on training data) to generate densities. Chiang and Gader (1997) used the percentage of cases for which input feature values contributed towards correct decisions on training data for each hypothesis.

In many applications, the number of knowledge sources is considerably less than the number of hypotheses, or classes. For example, in handwritten word recognition (Gader et al., 1995c, Gader et al., 1996a) the number of classes (i.e., words) is "essentially infinite", and so, for any test image, the potential classes must be dynamically assigned.

Example 4.12 This example, taken from Gader et al. (1996a) and

Gader and Mohamed (1996) shows the use of the fuzzy integral as a match function in a dynamic programming-based word recognition application. The details can be found in Gader et al. (1996a) and its references. Here, we only wish to demonstrate how the fuzzy integral can improve word recognition, a domain where the class labels change dynamically with each object (image). Let I be a word image and let $L = \{W_1, W_2, \ldots, W_q\}$ be a set of possible words or strings for the particular image. The top of Figure 4.22 shows an image of the actual word "Richmond". The set L represents the dictionary or lexicon of all possible words. One version of the word recognition problem is to find the word in L that matches I better than all other words in L. In the baseline dynamic programming algorithm (see Gader et al., 1996a), a match between a string $W \in L$ and I is computed by maximizing the average match between segmentations of I and the individual characters of the word W. In the fuzzy integral algorithm, it is used to compute the match score.

Let $W = c_1c_2 \cdots c_n$ where c_i is the ith character in W. The basic idea is as follows: We assign a density to each character class represented in the string W, $c_i \rightarrow g^i$, using some method. Given these densities, we can generate a λ-fuzzy measure g. Thus, each string has a measure associated with it. Assume we have a segmentation of the word image I into n segments. Basic character recognition algorithms (neural networks, usually) provide confidence values that the ith segment represents the ith character in the string. Denote these confidence values by $h(x_1), h(x_2), \cdots h(x_n)$. The baseline system computed the match between the segmentation and the string by averaging these confidence values. Alternatively, they integrated these confidence values with respect to the measure g to arrive at a different match score.

Figure 4.22 illustrates the basic process. The word image is broken into small pieces (no bigger than one character) called primitives. Then the primitives are joined together to get the "best" match to each word in the lexicon using dynamic programming. The match of the image of the actual word "Richmond" to the strings "Richmond" and "Edmund" are shown near the bottom of the figure. For the two segmentations, the character confidence values are shown below each segment. The average of the character confidence scores (on a scale from 0 to 1, but note that the displayed values in the figure are multiplied by 100) in the correct match is 0.57, whereas for the incorrect match it is 0.58. For each segmentation, a Choquet integral was computed using all densities equal to $1/(1.4n)$ where n is the string length. The parameter 1.4 was found through analysis of a set of training data. In this case, the Choquet integral assigned a score of 0.54 to the correct match and a score of 0.52 to the incorrect match.

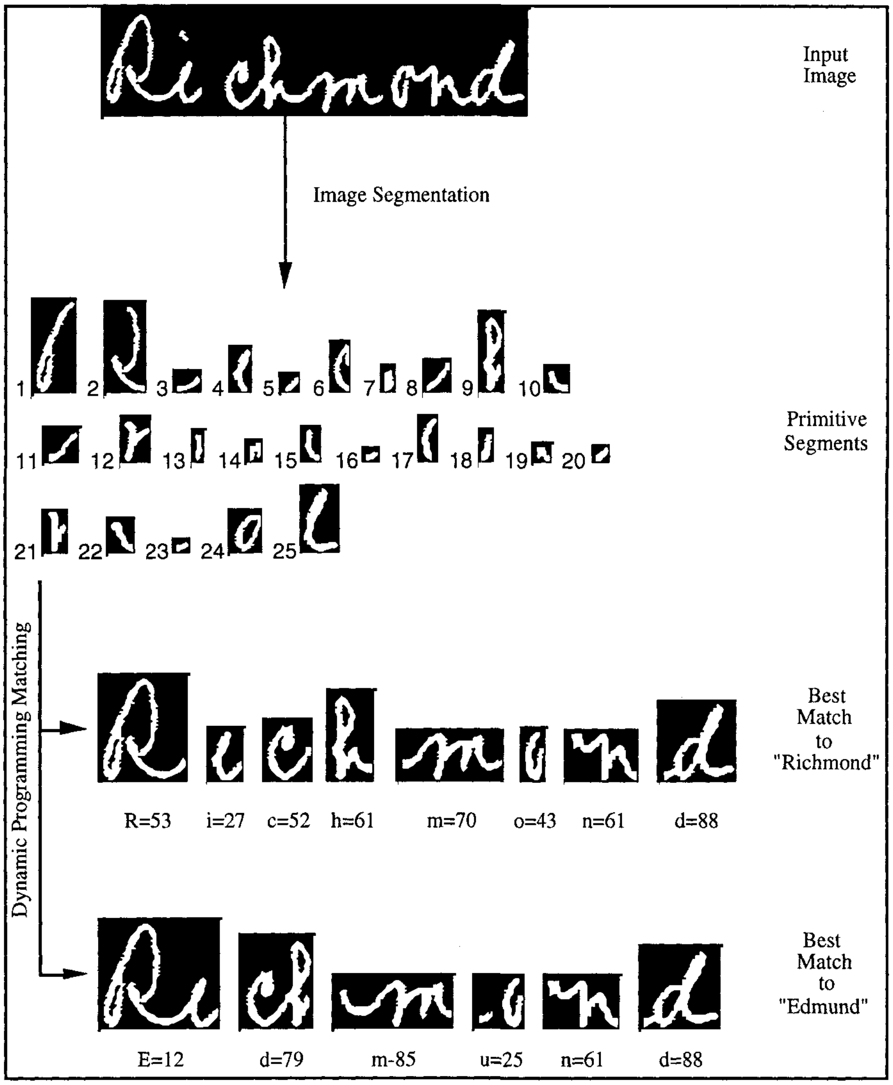


Figure 4.22 Dynamic programming approach to word recognition : (numbers below each letter are scaled by 100 for display)

All testing was performed on images of handwritten words taken from the standard SUNY CDROM data set described in Hull(1994) Specifically, they used the 317 word "bd city name" test data set, and presented results for the lexicon set with average length 100.

The segmentation algorithm (Gader et al. 1995) was initially developed on *National Institute of Standards and Technology* (NIST) character data. Later, the algorithm was adapted to images of handwritten words obtained from the United States Postal Service through the *Environmental Research Institute of Michigan* (ERIM) using a data set referred to as the *bha data* (Gader et al., 1995, Gillies et al., 1993). The character recognition and compatibility neural networks were trained on characters and pairs of characters extracted from the bha data.

Table 4.30 shows the increase in recognition rates obtained by setting all densities equal to the same value, $g^i = \frac{1}{s \cdot n}$, where s is a parameter and n is the length of the string W. This method reduces to the baseline system (averaging) for s = 1 but produced better results than the existing system for larger values of s.

Table 4.30 Recognition Rates obtain from initial experimentation

Baseline System	Equal densities (s=1.4)
83.9%	86.1%

One interesting property of the fuzzy integral is that it seems to be less susceptible to single outliers in the partial evaluation functions than many other methods. To illustrate, consider the match of the image of the word "plain" to the strings "Erin" and "Plain" as summarized below. In the match of plain to "Erin", the match of a wrong group of primitives within the image to "i" is very high (bigger than any other character match). This big value dominates the averaging method, causing the wrong classification, but is compensated for by the Choquet integral.

MATCH OF Plain TO STRING Erin	
Character confidence function	h: 0.06 0.35 0.84 0.23
Old Match Score	0.554
New Integral Match Score	0.309

MATCH OF Plain TO STRING Plain	
Character confidence function	h: 0.29 0.30 0.21 0.67 0.36
Old Match Score	0.542
New Integral Match Score	0.337

When desired outputs of the integrals are available for each class for a large enough training set, Grabisch has shown that the entire measure for each class can be learned via an optimization problem (Grabisch et al., 1995, Grabisch and Nicolas, 1994) using quadratic programming. This methodology is quite useful but requires a least squares objective function in order to derive a quadratic program. If the number of information sources is large, this optimization may be computationally prohibitive, and in noisy applications, least squares is known to be non-robust. However, Chen et al. (1997) used the quadratic programming methodology to define optimal measures for computing word recognition confidence from character confidence values in handwritten word recognition with excellent results.

The process of determining the densities for fuzzy integrals can also be thought of as a random search activity when training data is available. Theoretically, an exhaustive search will always find the best density set. But when the number of classifiers is large, this approach is impractical. Yan and Keller(1996) suggested a modified random search and a form of simulated annealing, both motivated by heuristics, to find densities for possibility integrals used in image segmentation.

Genetic algorithms provide an efficient alternative to exhaustive search. They have been utilized by some researchers to obtain various parameters of neural-fuzzy pattern recognition systems, including density values for fuzzy measures (Wang et al., 1997, 1998, Pham and Yan, 1997). Densities corresponding to multiple classifiers are coded as chromosomes in the genetic algorithm, and the classification rate is used as the objective function to be maximized. They combine "survival of the fittest" of strings and special ways of information exchange between generations of strings to form a search algorithm that is neither gradient search nor a simple random walk. In a genetic algorithm, each possible solution is coded as a binary string and a set of candidate solutions called a population is maintained. A genetic algorithm uses the three operators: reproduction, crossover, and mutation, operating in cycles (generations), returning the string with best fitness. One advantage of this type of search algorithm is that the densities for all classes are updated at each step, allowing for better comparison of fuzzy integrals values. See Geyer-Schulz (1998) for a complete treatment of crisp and fuzzy genetic algorithms.

Keller and Osborn (1996) described a novel fuzzy density training algorithm (for Sugeno fuzzy measures) which was similar to training algorithms employed in neural network research. It was based on a "reward/punishment" scheme to adjust the fuzzy densities for each class. Initially the densities for each class start out at the same value, for example, 1/n. For a given labeled object instance, the integrals are calculated for each classification hypothesis. If the largest integral value does not correspond to the correct classification, training must be done. The offending fuzzy integrals are punished by decreasing the densities that directly contributed to their integral values while the correct class has its contributing densities increased. This tended to raise the integral value of the correct class integral and lower the value of those that were misclassifying the input. This process is continued until all objects in a training set were correctly classified. This approach was used to train fuzzy integral classifiers in a target recognition application (Keller and Osborn, 1991).

In the methods discussed above, you need to compute membership values (confidences) in different classes from observed feature data. Several methods can be used for this purpose (Section 4.7.I). You are probably getting tired of hearing this, but for any fuzzy classifier to work, fuzzy sets must be generated. This is equivalent to estimating conditional probability density functions and prior probabilities in statistical classifier design.

There are several extensions of the given fuzzy integral pattern recognition algorithm both in terms of the class of fuzzy measures utilized and in the formulation of the equation to generate the values (i.e., generalizing equation (4.49). The reader is referred to Keller et al. (1994a) for a discussion and real examples of these extensions (as well as a good bibliography of fuzzy integral approaches to pattern recognition).

4.6 Fuzzy Rule-Based Classifiers

Fuzzy rule-based systems have gained a wide degree of acceptance in control, where output signals are almost always continuous. In pattern recognition, rule-based systems are less evident, since crisp classifiers are discretely valued. One advantage of using a fuzzy rule-based classifier, however, is that the labels can be soft during the operation of the rule-base, and hardened as the last step of classification.

There are many, many ways that rules can be extracted from data (Weiss and Kulikowski, 1991, Lin and Lee, 1996, Jang et al., 1997, Nguyen and Sugeno, 1998). We will discuss rule extraction methods based on decision trees, clustering and heuristics in this section, and on neural networks in Section 4.7. Our intention is to begin in a gentle, non-traditional way, with some simple examples based on crisp decision trees. We hope this will pave your way towards understanding some useful connections between three apparently disparate fields of classifier design : neural network classifiers, machine learning (classification trees), and (fuzzy) rule-based systems.

Like neural networks and fuzzy systems, decision trees can be used for approximation and classification. Since this is a book on pattern recognition, our interest is in the use of trees as classifiers, which in our context are sometimes called *classification trees.* Many writers and readers are used to the more general term decision tree, but we will use these two terms interchangeably unless there is a need to be more specific.

A. Crisp decision trees

Decision trees (DTs) are a simple and intuitively natural way to introduce the idea of rule-based network approaches to classifier design. Let $\mathbf{D}_{DT}$ be a decision tree classifier, $\mathbf{z}$ be a point to be labeled, and let $\mathbf{D}(\mathbf{z})$ represent any of the classifiers we have studied so far ("single stage" classifiers). Advocates of decision trees list the following as advantages of the decision tree approach:

1. $\mathbf{D}_{DT}$ approximates global, complex decision regions by constructing the union of simpler local ones.

2. Calculation of $\mathbf{D}(\mathbf{z})$ involves all c classes, whereas $\mathbf{D}_{DT}(\mathbf{z})$ is often obtained using a subset of the c classes, so $\mathbf{D}_{DT}$ may be faster than $\mathbf{D}$.

3. Calculation of $\mathbf{D}(\mathbf{z})$ uses all p input features for all decisions regardless of their actual values, whereas the features used in computing $\mathbf{D}_{DT}(\mathbf{z})$ may be used in various combinations - different nodes in decision trees may use different feature subsets to get good discrimination between the classes that arrive at particular nodes.

Proponents of decision trees also concede some disadvantages:

1. Overlapping data classes (for example, from mixture distributions) can cause the tree to have many leaves (and, as we shall see, this means many rules), thus increasing memory requirements. And when the decision tree is soft (fuzzy, probabilistic or possibilistic), this can lead to large evaluation time during operation.

2. Decision trees typically overfit the training data, so a good pruning algorithm is needed to make the tree generalize well.

3. Classic decision trees have a parallel axis bias (this can be overcome by "oblique code 1", Murthy et al, 1994).

4. Decision trees grow larger with more training data, but their accuracy on test data rarely shows a concomitant increase.

5. Decision trees generally don't afford incremental learning (but ID5 is incremental, Utgoff, 1989).

We can summarize these two lists succinctly: as with all other classifier designs, you take the good with the bad. As usual, the real question is how to find a good classification tree? Unlike previous classifiers we have studied, $\mathbf{D}_{DT}$ involves more than just choosing a family of classifier functions and training $\mathbf{D}$ by some method for parametric estimation. Finding an (error rate) optimal $\mathbf{D}_{DT}$ is not so easy. The design process, on the other hand, is fairly standardized: build it, prune it, and test it. First we develop some terminology, then we discuss strategies for building a crisp decision tree, and finally, we review most of the work that has been done towards fuzzifying crisp structures.

A *tree* T = (V, E) is a directed graph (or *digraph*) which has a *root node* $v_1 \in V$ with the property that there is a unique *path* $p(v_1, v)$ from v_1 to every other node $v \in V$, and no path from v_1 to itself. In this section we denote nodes of T as V={v_j}, v standing for vertex. v_1 is the only node without a parent. Terminal vertices, the only nodes without children, are called *leaves*. Non-terminal nodes are also called *internal nodes*, denoted as V_I; and we denote the leaves by V_L. Thus, $V = V_L \cup V_I$, and $V_L \cap V_I = \emptyset$. T is *binary* when each non-terminal node $v \in V_I$ has two exit edges (or equivalently, two children, usually called the left and right children of v). Thus, internal nodes can have one or more children but only one parent - figure *that* out!.

Learning the structure of a decision tree, or equivalently, the rules it represents, is called *rule induction* (from our viewpoint, this means training $\mathbf{D}_{DT}$). A classification tree $\mathbf{D}_{DT}$ *covers* the given cases (inputs) in crisply labeled *input-output* (IO) data set X if and only if the rules it corresponds to are consistent in the pattern recognition sense, i.e., the resubstitution error rate $E_{\mathbf{D}_{DT}}(X|X) = 0$. Any finite data set can be covered by at most n consistent crisp rules (i.e., by a classification tree with n pure leaves). In general, the number of leaves required to consistently cover c classes is $|V_L| \geq c$.The smallest tree that covers the training data is desirable, but often does not provide good generalization.

Figure 4.23 shows a crisp decision tree classifier $\mathbf{D}_{DT}$ whose job is to decide which of three crisp labels, chicken ($\mathbf{e}_1$), crab ($\mathbf{e}_2$) or fish ($\mathbf{e}_3$), a particular object should be given.

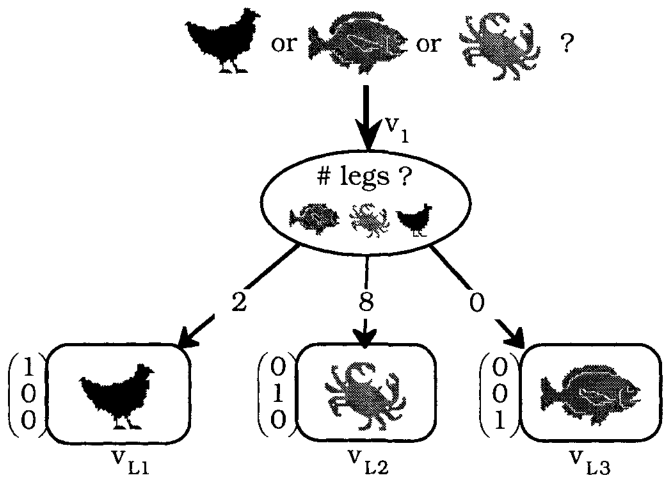

or
or
?
v_1
# legs ?
2
8
0
$\begin{pmatrix}1\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\1\end{pmatrix}$
v_{L1}
v_{L2}
v_{L3}

Figure 4.23 A decision tree that covers three classes

We can't just drop a fish into the computer and ask for an answer (OK, some computers *are* pretty fishy, as are many of our comments). As usual, we have two choices for representation of objects: *numerical features* (either continuously valued features such as weight, length; or discretely valued features such as number of fins, etc.); or categorical attributes (color, skin texture, etc.). In Figure 4.23 the only numerical feature needed to make correct classifications is the integer n_L, the number of legs: $n_{\text{chicken}} = 2$, $n_{\text{crab}} = 8$ (we don't count the claws of the crab as legs - they are hands), $n_{\text{fish}} = 0$. When the computer considers a question such as " # legs?", it must make a computation or comparison to answer the question. This happens at all the internal nodes, and at none of the leaves. The set of leaves $T_L = \{v_{L1}, v_{L2}, v_{L3}\}$ in Figure 4.23 provide a crisp 3-partition of the data, with label vectors as shown to the left of each leaf.

Classification is accomplished just as you see it: each internal node in the tree poses one question (here the root is the only internal node), and, based on the response, the object traverses T from root to some leaf. When a leaf consists of objects from just one crisp class, we say it is *pure*; and when all the leaves are pure, we say T is a pure classification tree. Following tradition, the leaf nodes are indicated by rectangles (well, ours are *almost* rectangles - apropos don't you think?), while the internal nodes are shown as ellipses. Also notice that the root node v_1 processes examples of all three classes - this is why they are shown "in" the node in Figure 4.23, but they don't reside there - they are just passing through. In this example, there are 3 classes and 3 terminal nodes, but usually, each class will have several leaves that bear the same crisp label.

Up to this point object and relational data have been continuously or discretely real valued, and in all cases the measurements (numerical features) can be ordered (this property results in the alternate name *ordinal data* for these two kinds of data). Decision trees can also deal with *categorical (nominal) data* - i.e., data whose features or attributes take values that have no ordering. Many sets of objects can be described nicely by categorical data.

Categorical attributes (nominal variables) are in some sense similar to the semantics of fuzzy descriptions in rule-based systems, where linguistic variables can take linguistic values. Categorical variables are not associated with membership functions, while linguistic variables take values that are in turn represented by fuzzy sets, that is, by (typically continuous) membership functions. In the domain of fuzzy systems models, we call words such as scaly, feathery, hard *linguistic values*, say $\{\underset{i}{\ell}\}$, of a linguistic variable $\mathcal{L}$ = skin texture. Linguistic variables in the fuzzy sets context are somewhat more general than categorical variables. First, the membership function that represents a linguistic value serves to modify the extent to which a particular observation should be considered to exhibit the linguistic attribute. Second, most linguistic values (e.g., low, medium, high) of linguistic variables (e.g. speed) have an unspecified but semantically clear *ordering*, as, for example, low is less than medium which is less than high. We will write vectors that have q linguistic variables for entries as $\mathcal{L} = (\mathcal{L}_1, \ldots, \mathcal{L}_q)^T \in \mathcal{L}^q$. The number of possible values that can be taken by a categorical or linguistic variable is called the *granularity* of the variable. For example, when texture is smooth, scaly, feathery, hard or leathery, its granularity is 5, whether each of these words is represented by a crisp or soft membership function.

The three objects in Figure 4.23 could be equally well described with a nominal variable such as skin texture as they are by counting the number of legs: the skin of chickens is feathery, of crabs is hard, and of fish is scaly, smooth or leathery. The tree built using the numerical feature "number of legs" and one built using the unordered categorical attribute "skin texture" will be identical in this simple example. The question shown at the root in Figure 4.23 becomes "is skin scaly, hard or feathery? The three crisp labels, chicken = feathery = $(\mathbf{e}_1)$, crab = hard = $(\mathbf{e}_2)$ or fish = scaly = $(\mathbf{e}_3)$ still apply, and the tree covers the three given cases, now described by values of a categorical variable.

B. Rules from crisp decision trees

The simple example in Figure 4.23 introduces the idea of using a DT for classification. What is ostensibly different with this approach from those previously discussed is the *representation* of the classifier function. If we designate the set of objects as X, the crisp decision tree classifier $\mathbf{D}_{DT}: X \to N_{h3}$ in Figure 4.23 can be represented by three crisp rules :

$$\text{If } (n_L = 2) \text{ Then } \mathbf{D}_{DT}(n_L) = \mathbf{e}_1 \quad ; \tag{4.63a}$$

$$\text{If } (n_L = 8) \text{ Then } \mathbf{D}_{DT}(n_L) = \mathbf{e}_2 \quad ; \tag{4.63b}$$

$$\text{If } (n_L = 0) \text{ Then } \mathbf{D}_{DT}(n_L) = \mathbf{e}_3 \quad . \tag{4.63c}$$

Unlike our previous classifiers, there is no functionally compact way to represent $\mathbf{D}_{DT}$. Moreover, rule-based systems like (4.63) will almost always be embodied as computer programs. This is our first example of a "learning" model that leads to a computational representation of $\mathbf{D}$. To emphasize the structure of classification trees as rule-based functions, we will denote the *set* of rules in a rule-base as $\mathcal{R} = \{R_1, \ldots, R_M\}$, and the *output* of $\mathcal{R}$ for input $\mathbf{x} \in \Re^p$ as the vector $\boldsymbol{\mathcal{R}}(\mathbf{x})$. This emphasizes that the rules are just a computational representation of a transformation, $\boldsymbol{\mathcal{R}}: \Re^p \mapsto \Re^q$.

Why use rules at all? The classifiers discussed so far make decisions based on mathematical models that have little or no "physical" meaning to most users. Generally, decisions rendered by a computer are based on reasoning that is not readily apparent (even to the designer of the system!). This can lead to a lack of confidence by humans in decisions made by the computer. Arguably, one of the primary advantages of rule-based classifiers is their ability to provide humans with understandable *explanations* of label assignments. Certainly system (4.63) satisfies this criterion : each rule is easily understood by us.

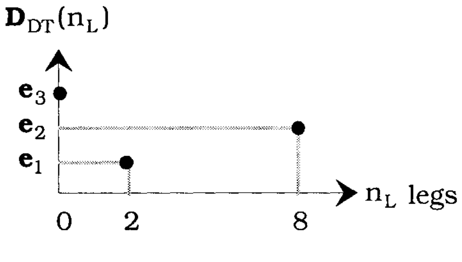


Figure 4.24 Geometric representation of (4.63)

Another nice aspect of rule-based classifiers is that they have a simple geometric interpretation. System (4.63) is illustrated in Figure 4.24, which depicts the functional action of each rule in terms of its numerical input (n_L) and output $\mathbf{e}_i$, represented here simply as integer i. The 3 discrete points in the plane represent these three rules completely.

The system in Figure 4.23 based on the numerical input "number of legs" has no rule for inputs that are not 0, 2 or 8. That is, it has no generalization capability at all. For example, some crabs come out of the water with only 7 legs, and the tree in Figure 4.23 will fail to classify crabs with this misfortune. Moreover, if a human was submitted to the tree in Figure 4.23, she or he would be classified as a chicken. The alternative tree based on skin texture would perform equally badly: turtles (hard skin) would be classified as crabs, and humans, perhaps, as fish. This is a problem that is particularly acute for crisp decision tree classifiers - it is not hard to train them to have zero resubstitution error rates, but they often generalize badly.

We can ameliorate this problem in the numerical case by erecting crisp membership functions along the horizontal axes that capture the training inputs in continuous intervals. The domains shown in Figure 4.25 are fish = [0,1), chickens = [1,3] and crabs = [7,9]; the ends of the intervals are called *cutpoints*. Geometrically this creates three crisp *rule patches*, as shown in Figure 4.25. (Actually they are not patches, since they have no vertical extent, the outputs being singletons; we show them with finite heights in Figure 4.25 so you can see them.)

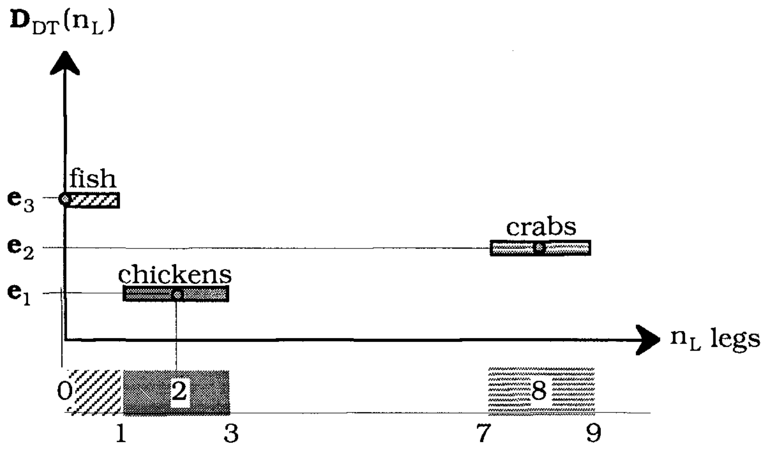


Figure 4.25 Crisp rule patches associated with (4.63)

The usual way to generalize a tree for continuous variables is to simply place a cutpoint at the midpoint of each pair of adjacent, distinct values taken by any continuous attribute in the training data. The updated version of Quinlan's classic tree-building algorithm (ID3, Quinlan, 1983) for continuously valued inputs called C4.5 (Quinlan, 1993) uses the feature values in the data as cutpoints. We have done it a little differently in Figure 4.25 so you can see the general idea, because several fuzzy generalizations of ID3 depart from the midpoint strategy used by C4.5. In any case, we refer to extensions of this kind that imbed the n discrete, observed values of feature i, i = 1,...,p, in some real interval (often the interval $[m_i, M_i]$ shown in equation (4.20)) as *cutpoint quantization.*

Now any input between 1 and 3, for example, would evoke the response "label 1" = (most like a) chicken, and 7-legged crabs will be classified correctly. This may seem nonsensical for discrete inputs, but it makes this important point: when we cover the training data with crisp rule patches, the patches allow us to *have* outputs for non-training inputs - i.e., the patches provide generalization capability to $\mathbf{D}_{DT}$. When the input variables are continuous, this makes a lot of sense.

Suppose we add humans (crisp label = $\mathbf{e}_4$) to the three classes in Figure 4.23. Since humans have 2 legs, the rules in (4.63) no longer cover the four classes - we need another feature. Let x = number of legs, y = number of hands for the object represented by $\mathbf{x} = (x, y)^T$, and count the claws of crabs as hands. Thus $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 8 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix}$ are, respectively, the discrete-valued, numerical feature vectors for all representatives of the four classes chickens, crabs, fish and humans. One decision tree that covers these four classes is depicted in Figure 4.26.

During training, we try to pick questions at the internal nodes in Figure 4.26 so that they act like a set of sieves, separating more and more training cases as we move down through the tree, until all the examples in each terminal node are in a single class. Here, as in Figure 4.23, the number of leaves equals the number of classes, but again, this is coincidental (in fact, unusual). Unlike our previous classifiers, the DT in Figure 4.26 uses the input features hierarchically (one at a time), rather than jointly. This is not a general property of all decision trees. All of the internal nodes in some decision tree classifiers process the entire input vector.

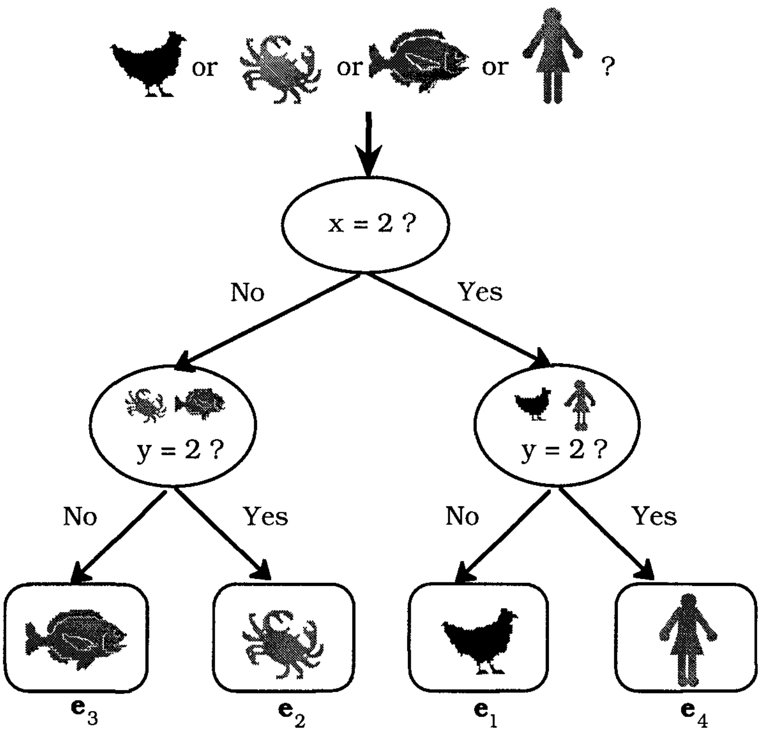


Figure 4.26 A binary decision tree that covers four classes

Crisp rules corresponding to the tree in Figure 4.26 are:

If $(x = 2 \text{ and } y \neq 2)$	Then	$\mathbf{D}_{BDT}(\mathbf{x}) = \mathbf{e}_1$	;	(4.64a)
If $(x = 2 \text{ and } y = 2)$	Then	$\mathbf{D}_{BDT}(\mathbf{x}) = \mathbf{e}_4$	.	(4.64b)
If $(x \neq 2 \text{ and } y = 2)$	Then	$\mathbf{D}_{BDT}(\mathbf{x}) = \mathbf{e}_2$	;	(4.64c)
If $(x \neq 2 \text{ and } y \neq 2)$	Then	$\mathbf{D}_{BDT}(\mathbf{x}) = \mathbf{e}_3$	.	(4.64d)

Figure 4.27 shows a different solution to the problem in Figure 4.26. Which tree, Figure 4.26 or Figure 4.27, is "best"? Both represent zero error rate solutions, but their *topology* is slightly different. Thus, the tree in Figure 4.26 has three internal nodes, while there are but two in the tree in Figure 4.27. Figures 4.26 and 4.27 illustrate that even in the simplest cases there is usually more than one covering decision tree, and data of any appreciable size will often have many. In machine learning, the node splitting principle chosen for building the tree produces a covering tree that is optimal with respect to the training criterion; and then most of the effort is placed on pruning the tree so that it generalizes well.

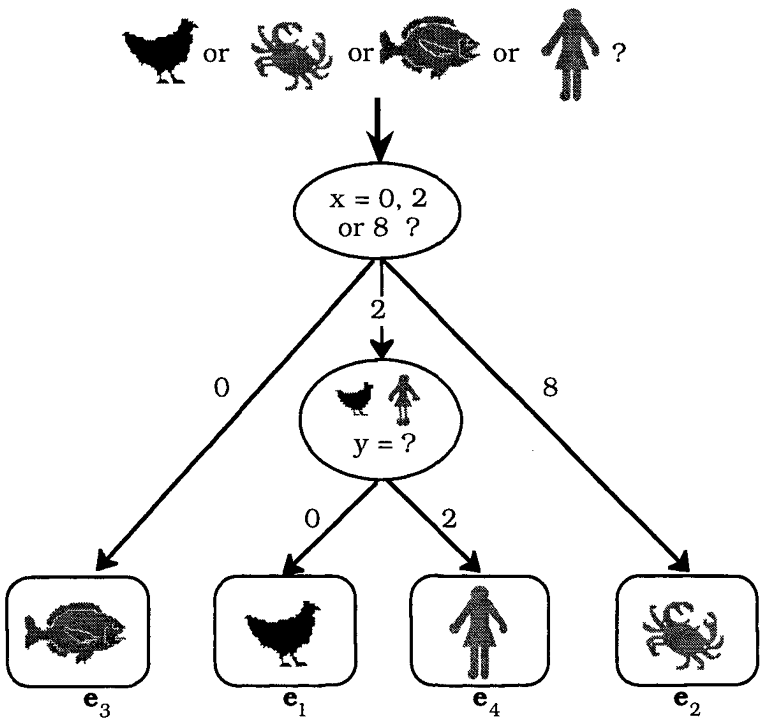


Figure 4.27 An alternate solution to the tree in Figure 4.26

When a crisp decision tree uses only rules in disjunctive normal form and the variables are continuously valued, $\mathbf{D}_{DT}$ is a piecewise linear classifier whose decision boundaries are hyperplanes that are parallel to the coordinate axes (hence the parallel axis bias in classical decision tree learning). In the special case in Figure 4.24, the three rules are points on vertical hyperplanes passing through 0, 2 and 8, because values on the horizontal axis are discrete.

If the constraint at each internal node is an inequality on a *continuously valued* feature, then a set of covering rules represents an IO relationship corresponding to capturing the training data in crisp rule patches or hyperboxes with sides parallel to the coordinate axes. This situation is depicted in Figure 4.28, which shows crisp rule patches capturing training data for two linearly separable classes ($\mathbf{e}_1$ = ducks and $\mathbf{e}_2$ = llamas).

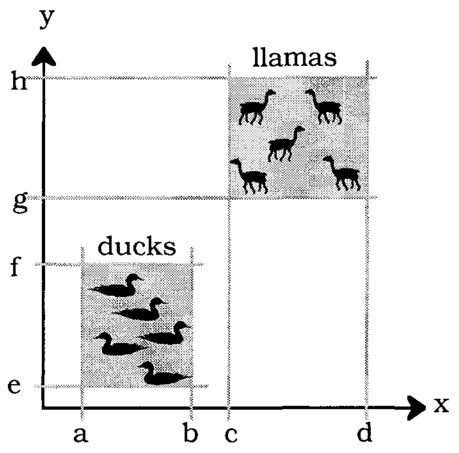


Figure 4.28 Geometry of rules on continuous numerical domains

The geometric interpretation of crisp rule patches is strictly correct for numerical inputs. For categorical inputs, we can imagine clusters in the input space corresponding to each categorical value (ducks are feathery, llamas are furry (fuzzy?), but there is no way to construct a graphical representation. The covering rules for the data in Figure 4.28 are

$$\text{If } (a \le x \le b \text{ and } e \le y \le f) \text{ Then } \mathbf{D}_{DT}(\mathbf{x}) = \mathbf{e}_1 \quad \text{, and} \tag{4.65a}$$

$$\text{If } (c \le x \le d \text{ and } g \le y \le h) \text{ Then } \mathbf{D}_{DT}(\mathbf{x}) = \mathbf{e}_2 \quad . \tag{4.65b}$$

Although Murty's (1994) oblique code 1 can sometimes capture a lot of training data with a few crisp rules, this is generally not the case unless the data are linearly separable. On the other hand, n distinct inputs can always be covered with n crisp rules by making the hyperboxes (or parallelepipeds) small enough.

C. Crisp decision tree design

Methods for decision tree design can be put into four main categories: (i) bottom-up approaches, some of which are very similar to unsupervised hierarchical clustering as discussed in Chapter 3; (ii) top-down methods; (iii) growing and pruning approaches; and (iv) hybrid methods. Top down approaches with subsequent pruning comprise the large majority of currently popular induction methods. All of the papers we discuss that develop fuzzy decision trees for classification fall into this group. Top down approaches involve node splitting rules, stopping criteria, and leaf labeling. Splitting rules are based on node splitting functions and termination criteria with constraints.

We want a decision tree that minimizes the generalization error $E_{\mathbf{D}_{DT}}(X_{te}|X_{tr})$. Most machine learning algorithms find a tree that is consistent, $E_{\mathbf{D}_{DT}}(X_{tr}|X_{tr})=0$, and then prune it. Typical tree design starts with a crisp partition of the training data, and uses the labels, in conjunction with some node splitting criterion function ι, to determine a tree structure that is optimal with respect to ι. In the training process X_{tr} is repartitioned into subsets of cases. In machine learning this is called partitioning the training examples (the prefix "re" is dropped). Once the tree is built, we abandon ι, and use the structure it provides to define decision functions $\{\phi_i\}$ at the internal nodes $\{v_i\}$. In the trees shown so far, we have indicated the decision functions at the nodes after the tree is built - not the node splitting functions used to build the tree.

Deciding how to split the cases at an internal node v_k is guided by a node splitting or *impurity function* ι_k at v_k. Impurity functions are often functions of relative frequencies of crisply labeled cases "arriving at, or in" the node to be split. Using relative frequencies amounts to deriving a numerical feature from the labels of the training data to cluster the cases, and it can be done for both numerical and categorical data. The basic objective is for the cases that are sent to each child of v_k to be "purer" (more well separated) than the cases that were sent to v_k. A function $\iota: N_{fc} \mapsto [0,\infty)$ is called an *impurity function* when

$$\iota(\mathbf{e}_j)=0,\ j=1,\ldots,c \qquad \text{, and} \qquad (4.66a)$$

$$\iota(\mathbf{1}/\mathbf{c})=\text{maximum} \qquad . \qquad (4.66b)$$

Recall that $N_{hc}=\{\mathbf{e}_1,\ldots,\mathbf{e}_c\}$ are the vertices of N_{fc}, and that $\mathbf{1/c}$ is its centroid (refer to Figure 1.2). Equation (4.66a) requires impurity functions to vanish at nodes where all the cases are in one class. Equation (4.66b) requires impurity functions to maximize at the centroid of N_{fc}, i.e., at nodes where the cases are equally distributed among the c classes. In short, impurity functions vanish at pure nodes, and maximize at the most impure nodes.

Let $\mathbf{p}=(p_1,\ldots,p_c)^T \in N_{fc}$, where $p_i = n_i / n$, i= 1,..., c for n crisply labeled data $X=\bigcup_{i=1}^{c} X_i$, $|X_i| = n_i \neq 0$. The two most common impurity functions (Breiman et al., 1984) are (Shannon's) *entropy* and the *Gini diversity index* of the vector $\mathbf{p}=(p_1,\ldots,p_c)^T$:

$$\iota_{ent}(\mathbf{p}) = -\sum_{i=1}^{c} p_i \log_2 p_i \qquad \text{, and} \qquad (4.67a)$$

$$\iota_{Gini}(\mathbf{p}) = 1 - \sum_{i=1}^{c} p_i^2 = \sum_{i=1}^{c} p_i - \sum_{i=1}^{c} p_i^2 = \sum_{i=1}^{c} p_i(1-p_i) \qquad . \qquad (4.67b)$$

The last form of (4.67b) is the way the Gini index appears when it is called Vadja's quadratic entropy (Vadja, 1970). The Gini index can also be viewed as an approximation to Shannon's entropy in (4.67a) because ($-\log_2$ p) can be approximated by (1-p) for small p. Safavian and Landgrebe (1991) list many other optimality criteria for tree structure design, including minimum expected path length, minimum number of nodes, minimax path length, etc. For example, Sethi and Sarvarayudu (1982) base their impurity function on average mutual information gain.

Once an impurity function is chosen, it is used to measure the impurity of internal nodes before and after splitting them into children. Candidate splits are postulated, and the decrease in impurity due to the split is calculated. The attribute selected for the next split is the one that maximizes the decrease in impurity at that node. Maximizing the change in entropy essentially minimizes the expected number of tests needed to classify an object. The overall impurity I(T) of any tree T with M leaves is defined as

$$I(T) = \sum_{k=1}^{M} \iota(\mathbf{p}_{Lk}) \qquad , \qquad (4.68)$$

where $\mathbf{p}_{Lk}$ is the vector of relative case frequencies in leaf v_{Lk}. When the leaves are all pure, each leaf has a crisp label vector attached to it, $\mathbf{p}_{Lk} = \mathbf{e}_j$ for some j, the impurity of the tree is 0, and so the training error of the tree is also 0.

The two most widely used algorithms for building crisp decision trees are Quinlan's (1983, 1986) ID3 (*interactive dichotomizer*) method and its extension to C4.5; and CART, the *classification and regression tree* approach described in Breiman et al. (1984). ID3 was originally designed to deal only with pretty small sets of categorical data. The machine learning community has essentially abandoned ID3 for Quinlan's (1993) much improved C4.5, which takes care of this deficiency, and which is much more widely used than CART. In statistical circles, however, CART is favored because of the "regression trees" it can build.

CART and ID3 are fairly similar: both models try to represent a crisp partition of the training data in a decision tree structure; both are top-down, node splitting approaches, and both attempt to minimize tree size while simultaneously optimizing some performance measure. The main differences between C4.5 and CART are that C4.5 uses entropy while CART uses the Gini index for node splitting, and CART is pruned by exhaustive search of all subtrees (Breiman et al., 1984), while C4.5 uses a more efficient pessimistic pruning strategy, especially for small data sets (Quinlan, 1993).

Since all of the fuzzy models we discuss in the sequel refer to ID3, we summarize it in Table 4.31, even though it has been supplanted by C4.5 in the machine learning community. The input data to ID3 are a set of n categorical data vectors, $\{\ell_k\} \subset \mathcal{L}^p$. For example, the attribute list or linguistic variables for the data given might be (color, texture and size). Color might be divided into (red, green, blue), *texture into* (smooth, rough), *and size into* (small, medium, large). Each such datum is described by a 3-tuple such as ℓ_k = (red, smooth, small), so $p = 3$. Our specification of ID3 treats the root node as a leaf in the first pass through the WHILE-DO loop.

Table 4.31 The ID3 algorithm (Quinlan, 1983)

```
In    Crisply labeled category data $X_{tr} = \{\ell_1, \ell_2, \ldots, \ell_n\} \subset \mathcal{L}^p$
      $X_{tr} = \bigcup_{i=1}^{c} X_{tr,i}$, $n_i = |X_{tr,i}|$ and $p_{tr,i} = n_i/n \; \forall i$
-----------------------------------------------------------------------------
Do    $v_1 \leftarrow X_{tr}$ ; $V_L = \emptyset$;
      While $I(T) > 0$; % create child nodes ; $\mathbf{p}_j$ = relative
          class frequencies of cases at node j
          Pick a leaf node $v_{Lk}$ at which $\iota_{ent}(\mathbf{p}_{Lk}) > 0$
          For all attributes $\{\mathcal{L}_i\}$ not in path $p(v_1, v_{Lk})$
          For all attribute values $\{\ell_{ij}\}$ of $\mathcal{L}_i$, compute
            $w_j$ = relative # of cases at child node for $\ell_{ij}$
          $\Delta\iota_{ent,ij|Lk} = \iota_{ent}(\mathbf{p}_{Lk}) - \sum_j w_j \iota_{ent}(\mathbf{p}_{ij|Lk})$.      (4.69)
          Choose the split(s) that maximize(s) (4.69)
          Update leaf node set $V_L$
      End While
-----------------------------------------------------------------------------
Out   A fully expanded crisp classification tree T with
      $|V_I|$ internal nodes; $M = |V_L|$ leaves, and overall
      impurity $I(T) = \sum_{k=1}^{M} \iota_{ent}(\mathbf{p}_{Lk}) = 0$.
```

Somewhat analogous to HCM, which favors clusters with many points (see Figure 2.3a), ID3 is biased towards attributes with many values, but this can be partially compensated for by altering the basic formula in (4.69) - see Quinlan (1993) for the details. Since many applications depend on continuously valued numerical data, ID3 has experienced many generalizations since Quinlan's original formulation (Fayyad and Irani, 1992, Cios and Liu, 1992, Seidlemann, 1993, Quinlan, 1993). Most of these updates take the form of discretizing the input range of each numerical variable into a number of subintervals or cutpoints. To appreciate ID3, we (like so many before us), repeat Quinlan's most well known example of the original algorithm.

Example 4.13 This example, adapted from Quinlan (1983), illustrates his original ID3 algorithm for growing a classification tree. Everybody repeats this example, so we have changed the objects from "a" and "o" to "r" and "e" just to be different. The entropy impurity function is used to determine a crisp classification tree that is optimal in two ways: its nodes maximize the information gain at each split of cases in the training data, and it is a consistent tree (the resubstitution error rate is zero). The training data are listed in Table 4.32. There are 8 objects, indexed for brevity by the integers 1 to 8, and these 8 training data are labeled as belonging in one of c = 2 crisp classes named "r" and "e". We let R denote the crisp cluster of 5 "r"s and E denote the crisp cluster of three "e"s in X.

Table 4.32 Training data for Quinlan's ID3 example

label	r	r	r	r	r	e	e	e
object	1	2	3	4	5	6	7	8
height	tall	short	tall	short	tall	tall	tall	short
hair	dark	dark	blond	blond	dark	blond	red	blond
eyes	blue	blue	brown	brown	brown	blue	blue	blue

Each object is represented by three attributes that are particular values of three *categorical variables*: $\mathcal{L}_1$ = height, $\mathcal{L}_2$ = hair color, and $\mathcal{L}_3$ = eye color. *Categorical values* taken by the categorical variables are $\{\ell_{11}$ = tall, ℓ_{12} = short$\}$ for height, $\{\ell_{21}$ = dark, ℓ_{22} = blond, ℓ_{23} = red$\}$ for hair color, and $\{\ell_{31}$ = blue, ℓ_{32} = brown$\}$ for eye color. Visual inspection of the attributes of objects in R and E does not lead to an obvious decision tree that covers the training data.

Since each object is characterized by three attributes, and the number of possible attribute values are 2, 3 and 2, this categorical feature space will support at most $2 \cdot 3 \cdot 2 = 12$ crisp rules, all of which have the general form, for a particular input z submitted to the (as yet to be determined) rule-base

$$\text{If } (\mathcal{L}_1(z) = \ell_{1j_1}) \text{ and } (\mathcal{L}_2(z) = \ell_{2j_2}) \text{ and } (\mathcal{L}_3(z) = \ell_{3j_3}) \text{ then } \mathbf{D}_{DT}(z) = \text{"r" or "e"}.$$

Figure 4. 29 illustrates the initial configuration of the training cases at the root node. Think of the "r"s as rabbits and the "e"s as elephants.

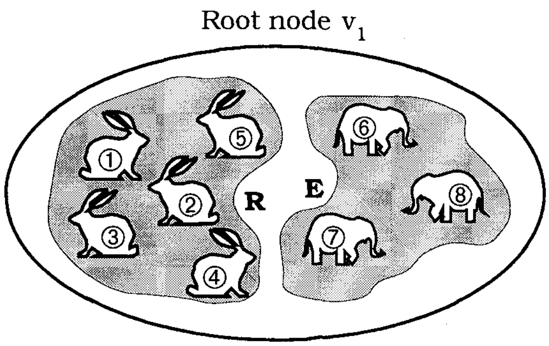


Figure 4.29 Training data set X prior to splitting the root node

The relative frequencies of cases in R and E are 5/8 and 3/8, respectively, yielding $I(T) = -\left(\frac{5}{8}\log_2\frac{5}{8}\right) - \left(\frac{3}{8}\log_2\frac{3}{8}\right) = 0.954$ as the initial impurity of the system (in bits). Since there are three attributes, there are three possibilities for splitting the cases at v_1, and ID3 defines the optimal split as the one which maximizes the gain of information (or gives the largest entropy decrease); the possible splits are shown graphically in Figure 4.30.

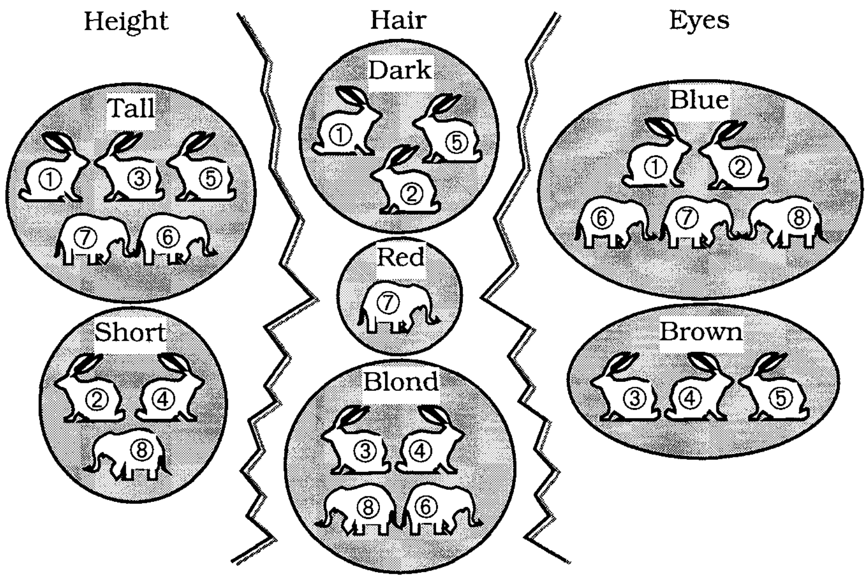


Figure 4.30 Possible case splits at the root node

Table 4.33 shows the proportions of cases that occur for each cluster in Figure 4.30 for each of the possible splits.

Table 4.33 Relative frequencies of clusters for each of the three splits of cases at the root node

$\mathcal{L}_1$ = height	$\mathcal{L}_2$ = hair	$\mathcal{L}_3$ = eyes
tall = ℓ_{11}	dark = ℓ_{21}	blue = ℓ_{31}
$p(\ell_{11}) = 5/8$	$p(\ell_{21}) = 3/8$	$p(\ell_{31}) = 5/8$
$p(r\mid\ell_{11}) = 3/5$	$p(r\mid\ell_{21}) = 1$	$p(r\mid\ell_{31}) = 2/5$
$p(e\mid\ell_{11}) = 2/5$	$p(e\mid\ell_{21}) = 0$	$p(e\mid\ell_{31}) = 3/5$
short= ℓ_{12}	red = ℓ_{22}	brown= ℓ_{32}
$p(\ell_{12}) = 3/8$	$p(\ell_{22}) = 1/8$	$p(\ell_{32}) = 3/8$
$p(r\mid\ell_{12}) = 2/3$	$p(r\mid\ell_{22}) = 0$	$p(r\mid\ell_{32}) = 1$
$p(e\mid\ell_{12}) = 1/3$	$p(e\mid\ell_{22}) = 1$	$p(e\mid\ell_{32}) = 0$
	blond = ℓ_{23}	
	$p(\ell_{23}) = 4/8$	
	$p(r\mid\ell_{23}) = 1/2$	
	$p(e\mid\ell_{23}) = 1/2$	

Next the entropy of each split cluster is computed. For example, the entropies of the tall and short clusters for the height split are

$$\iota_{ent}(\text{tall}) = \iota_{ent}(\ell_{11}) = -\left(\frac{3}{5}\log_2\frac{3}{5}\right) - \left(\frac{2}{5}\log_2\frac{2}{5}\right) = 0.971;$$

$$\iota_{ent}(\text{short}) = \iota_{ent}(\ell_{12}) = -\left(\frac{2}{3}\log_2\frac{2}{3}\right) - \left(\frac{1}{3}\log_2\frac{1}{3}\right) = 0.918.$$

Now we use the prior probabilities of the tall and short clusters to compute the overall entropy of the height split as

$$\iota_{ent}(\mathcal{L}_1) = p(\ell_{11})\iota_{ent}(\ell_{11}) + p(\ell_{12})\iota_{ent}(\ell_{12}) = \frac{5}{8}(0.971) + \frac{3}{8}(0.918) = 0.951.$$

In a similar manner we find the overall entropies for the other two splits as $\iota_{ent}(\text{hair}) = \iota_{ent}(\mathcal{L}_2) = 0.5$ and $\iota_{ent}(\text{eyes}) = \iota_{ent}(\mathcal{L}_3) = 0.607$. Finally, each of these three entropies is subtracted from the initial system entropy to get the overall entropy decrease for that split:

$$\Delta\iota_{ent}(\text{height}) = I(T) - \iota_{ent}(\mathcal{L}_1) = 0.954 - 0.951 = 0.003$$
$$\Delta\iota_{ent}(\text{hair}) = I(T) - \iota_{ent}(\mathcal{L}_2) = 0.954 - 0.500 = 0.454 \quad .$$
$$\Delta\iota_{ent}(\text{eyes}) = I(T) - \iota_{ent}(\mathcal{L}_3) = 0.954 - 0.607 = 0.347$$

Since the split of the root node by the attribute "hair" results in the largest decrease in system entropy, this is the first split made by ID3 for this data set. This split gives the root node 3 children. The children of v_1 are the three nodes shown in the middle of Figure 4.31: two of them are "pure" - they contain samples from only one class - and will thus be leaves in the final tree. The only node left to split is the 'blond" cluster, which contains 2 cases each from the labeled data. This node offers two possible splits, one on hair and one on eyes. Repeating the procedure just completed for this split, you will find that the preferred split is on eyes, and for this simple example, the final tree has been reached. Figure 4.31 shows the final tree.

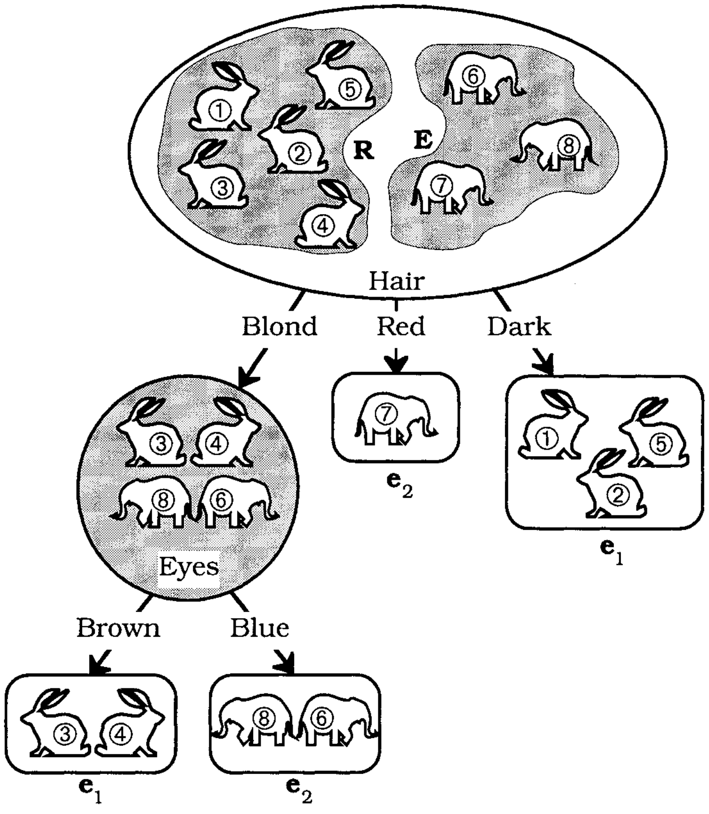


Figure 4.31 Crisp ID3 classification tree for data in Table 4.32

Since the leaves of the tree in Figure 4.31 are all pure, this is a tree with zero resubstitution errors, and is thus optimal with respect to the training data, as it correctly classifies all of them. However, this tree may or may not respond well to inputs that don't have the four combinations of attributes that are missing in the training data. In fact, without some sort of extension, the rules $\mathcal{R}$ from this tree won't even process the four missing cases.

There are four pure leaves in this tree, so the rule-base uses M=4 crisp rules to cover the c=2 classes labeled $\mathbf{e}_1$ and $\mathbf{e}_2$ in Figure 4.31. Notice that values of the height attribute $\mathcal{A}_1$ are not used at all. The four rules, written out with words, in order, from left to right by the ordering of the leaves in Figure 4.31 are:

R_1 : If (hair= blond) and (eyes = brown)	then z = rabbit ;	(4.70a)
R_2 : If (hair = blond) and (eyes = blue)	then z = elephant;	(4.70b)
R_3 : If (hair = red)	then z = elephant;	(4.70c)
R_4 : If (hair = dark)	then z =rabbit.	(4.70d)

Thus, it takes two elementary rules to cover each class. Another point to notice about Figure 4.31 is that the three levels in this tree correspond quite nicely to the levels in dendograms that represent top down hierarchical clustering procedures. Compare Figures 3.4 and 4.31 to see this, but flip Figure 3.4 "upside down", since it was built with a bottom up procedure. At the first level of T in Figure 4.31 all 8 data are in c = 1 crisp cluster; at level 2, there are c = 3 crisp clusters, two of which are pure; and at level 3, there are c = 4 crisp pure clusters. So, it's no surprise that hierarchical clustering has played a role in several tree growing methods - indeed, ID3 *is* top down hierarchical clustering for categorical data; but unlike the algorithms in Chapter 3, ID3 is supervised - it gets to use the crisp labels to construct pure clusters for classifier design.

There are many methods for termination of node splitting before reaching a fully expanded tree, and just as many methods for pruning fully expanded trees (Safavian and Landgrebe, 1991; Weiss and Kulikowski, 1991). These two aspects of the erection of $\mathbf{D}_{DT}$ have not received much attention from fuzzy classifiers. We are content here to note that termination of node splitting affects the performance of $\mathbf{D}_{DT}$ just as surely as termination of, say, any prototype generation algorithm, affects the quality of a 1-np classifier that uses the prototypes. Expansion can be terminated before completion, or fully expanded trees can be pruned back to subtrees. In either instance, the tree that remains will in all likelihood be impure. This is done in hopes that the (guaranteed) increase in training error due to abandoning a pure tree will be rewarded by a concomitant decrease in testing error (i.e., better generalization).

When leaf v_{Li} in the final tree contains cases of more than one type, the relative percentages of each label, $\mathbf{p}_{L_i} \in N_{fc}$, can be regarded as the consequent output for inputs that travel the path $p(v_1, v_{Li})$, i.e., $\mathbf{D}_{DT}(\mathbf{z}) = \mathbf{p}_{L_i} \in N_{fc}$. For example, if the node labeled "eyes" in Figure 4.31 is, after pruning that tree, a leaf in a subtree of T, since it contains 2 cases each of classes 1 and 2, $\mathbf{p}_{eyes} = (0.5, 0.5)^T \in N_{fc}$ is the probabilistic label vector attached to this node. We still get exact, unique matches to training data on the left sides of the crisp rules (the firing strength is still 1 along the unique path $p(v_1, v_{Li})$), but the classifier output is now soft at impure leaves. A strategy such as hardening by equation (1.15) can be used to convert soft output labels to crisp ones. By our convention $\mathbf{D}_{DT}$ is now a soft (decision tree) classifier, but hardly anyone would call the tree that produces such decisions a soft decision tree. This terminology is reserved for the more general situation discussed in Subsection F.

If each of the objects in Figure 4.31 was represented by a numerical feature vector, then each of the four leaves would have a (sample mean) point prototype $\bar{\mathbf{v}}_{Li}$ associated with the data in leaf v_{Li} (don't confuse the vector $\bar{\mathbf{v}}_{Li} \in \Re^p$ with the vertex $v_{Li} \in V_L$), and the classifier tree in Figure 4.31 would be similar to a 1-nmp classifier as discussed in Section 4.2.

While it is nice to exhibit the rules with their semantic meanings (after all, this is one of the attractive features of rule-based classification - easy to understand reasons for the labels assigned - you've never seen a blond elephant with blue eyes? Too bad!), we need to become comfortable with the symbolic notation for rule-base $\mathcal{R}$. Here is system (4.70) in terms of linguistic variables, linguistic values, and the classifier function it defines:

$$R_1 : \text{If } (\mathcal{L}_2 = \ell_{22}) \text{ and } (\mathcal{L}_3 = \ell_{32}) \Rightarrow \mathbf{D}_{DT}(z) = \mathbf{e}_1 \quad ; \tag{4.71a}$$

$$R_2 : \text{If } (\mathcal{L}_2 = \ell_{22}) \text{ and } (\mathcal{L}_3 = \ell_{31}) \Rightarrow \mathbf{D}_{DT}(z) = \mathbf{e}_2 \quad ; \tag{4.71b}$$

$$R_3 : \text{If } (\mathcal{L}_2 = \ell_{23}) \Rightarrow \mathbf{D}_{DT}(z) = \mathbf{e}_2 \quad ; \tag{4.71c}$$

$$R_4 : \text{If } (\mathcal{L}_2 = \ell_{21}) \Rightarrow \mathbf{D}_{DT}(z) = \mathbf{e}_1 \quad . \tag{4.71d}$$

This form for $\mathcal{R}$ is a step towards the fairly compact general formulation of fuzzy rule-based systems given in the next subsection. We need to add a few things here and there (most importantly, membership functions for the linguistic values $\{\ell_{ij}\}$), but (4.71) contains most of the elements we need.

Once $\mathbf{D}_{DT}$ is trained (and in practice, almost always pruned), it is ready to classify test data. One or more components (numerical feature values or categorical attribute values) of an unlabeled input datum $\mathbf{z}$ are assessed by a crisp decision function at each internal node as $\mathbf{z}$ traverses through the internal nodes in T, until it arrives at a leaf. For crisp decision trees with M pure leaves, each leaf is associated with *exactly one of the c labels* $\mathbf{e}_i \in N_{hc}$, and, as in (4.71), a crisp decision can be made without further consideration. In this case the path, call it $p(v_1, v_{Li})$ from the root v_1 to leaf v_{Li} corresponds to crisp rule R_i in $\mathcal{R}$, and when $\mathbf{z}$ traverses $p(v_1, v_{Li})$, we say that rule R_i "fires" with *firing strength* =1, meaning that this is the unique rule whose precedent arguments exactly matched the components of the input datum. The fact that the consequent of R_i in this case is a single label is due to the purity of the leaf v_{Li}. Even when the leaves are not pure (and in C4.5, this is the usual case after pruning), classical decision trees identify each leaf with the crisp class having the majority of cases at the leaf.

D. Fuzzy system models and function approximation

This subsection contains a short description of the two main types of fuzzy rule-based systems: the *Mamdani-Assilian* (1975) model and the *Takagi-Sugeno* (1985) model. We abbreviate these as MA and TS hereafter, without reference to the original papers, and when we say fuzzy system, we mean fuzzy rule-based system.

Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \Re^p$ and $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \Re^q$. We suppose an unknown function $\mathbf{S}: \Re^p \mapsto \Re^q$ for which $\mathbf{y}_k = \mathbf{S}(\mathbf{x}_k)$, k=1,...,n, so $Y = \mathbf{S}[X]$. We call X and Y *input-output* (IO) data, and let $XY = \left\{(\mathbf{x}_k, \mathbf{y}_k)^T = (x_{1k}, \ldots, x_{pk}, y_{1k}, \ldots, y_{qk})^T : \ k = 1, \ldots, n\right\} \subset \Re^{p+q}$ be the concatenation of each input and output vector in X and Y. Finding a good estimate $\hat{\mathbf{S}}$ of $\mathbf{S}$ using XY is variously called interpolation, collocation, function approximation, or most commonly, *supervised learning.* In pattern recognition we are interested in approximating classifier functions $\mathbf{D}: \Re^p \mapsto N_{pc} \subset \Re^c$. We will use $\mathbf{S}$ to emphasize the role of MA or TS systems as approximators to vector fields in a more general setting. When p=1, $\mathbf{S}$ is called single input, and when p >1, it is multiple input. When q=1, $\mathbf{S}$ is called single output, and when q >1, it is multiple output. We abbreviate these four cases in the usual way: *multiple-input multiple-output* is MIMO, and similarly for MISO, SIMO and SISO.

There is some confusion in the literature about the difference in meaning between the terms interpolation and extrapolation. In numerical analysis interpolation and collocation are synonyms that mean "through the training data", while extrapolation means values taken at " any points not in the training data". However, some writers use interpolation to mean values taken "at points not in the training data that lie 'in-between' points in the training data"; for these authors, extrapolation means values taken on "points not in the training data, and 'beyond it'". This is fine for real valued functions, where interpolation would mean in the interval bounded by the minimum and maximum points in the training data, and extrapolation means outside this interval. When the data are p-dimensional however, defining the notions of "inside" and "outside" or "within" and "beyond" (the convex hull of the training data, for example?) become problematical. In this book approximating functions always extrapolate, and may or may not interpolate. Since other writers use these terms in different ways, just be careful to check the writer's definition of how the term is used in a particular book or paper.

There are two basic approaches to approximation. The classical approach assumes a functional form for $\mathbf{S}$ that has a vector $\boldsymbol{\theta}$ of unknown parameters, indicated as $\mathbf{S}(\mathbf{x};\boldsymbol{\theta})$. Then we use XY with a principle of inference (and possibly, an algorithm to optimize the model) to estimate some optimal parameters $\hat{\boldsymbol{\theta}}$ of $\mathbf{S}(\mathbf{x};\boldsymbol{\theta})$. This gives us $\mathbf{S}(\mathbf{x};\hat{\boldsymbol{\theta}})$, an approximation to $\mathbf{S}$ that is optimal in the sense of the model used to obtain it. Examples in this category include regression analysis, collocating polynomials, and least squares estimation with, for example, radial basis functions.

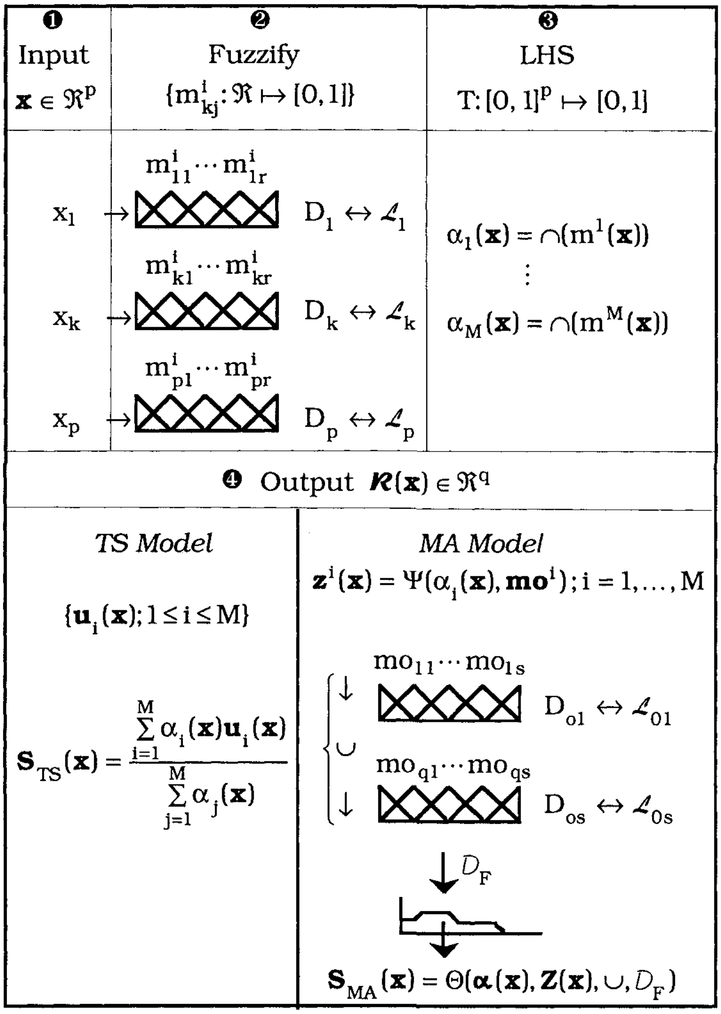
The second approach to approximation by supervised learning is to find a computational transformation (a computer program) that represents $\mathbf{S}$. The computer program also depends on parameters $\boldsymbol{\theta}$ that must be acquired using XY, and there is no harm in again writing $\mathbf{S}(\mathbf{x};\boldsymbol{\theta})$, now meaning a computer representation of $\mathbf{S}$, so that $\mathbf{S}(\mathbf{x};\hat{\boldsymbol{\theta}})$ is again an approximation to $\mathbf{S}$ that is optimal in the sense of the model used to obtain it. This group of techniques is sometimes subdivided into "parametric estimation" and "linguistically descriptive" methods. Neural-like networks, decision trees, and rule-based systems are examples of computational transformations that are used to represent $\mathbf{S}$. (Indeed, in many instances these three model styles can be transformed into each other.) If the learning involves more than just a few numerical parameters - e.g., if the basic structure of the network, number of rules, and so on - are also learned, this field is sometimes regarded as (part of) *model* or system *identification*. System identification covers a lot of ground; we will discuss some aspects of it only in the context of decision trees, rule-based systems and neural networks.

We divide approximation into three major steps: (i) *structure definition*, (ii) *parameter estimation* and (iii) *system validation*. Structure definition specifies the general architecture of $\mathbf{S}$. For example, if we choose a regression model, structure definition includes decisions about whether to use linear or non-linear regression, and the exact form of the objective function to be used. If the model is a decision tree, structure refers to the number of levels, nodes per level, number of leaves, edge weights, and so on. For neural models we choose the type of network architecture, number of layers, number of nodes, integrator and transfer functions for the nodes, etc. For fuzzy models, structure definition involves specification of items such as the number of linguistic values for each linguistic variable, forms for the antecedents and consequents of rules, operators for the reasoning system, etc. Parameter estimation in these three cases means, for example, finding the regression coefficients or decision function parameters or network weights or parameters of the membership functions of different rules or nodes in the tree. Optimization and validation test the system against performance requirements. This last step can include fine tuning of either the initial structure or estimated parameters.

Once the structure of $\mathbf{S}(\mathbf{x};\boldsymbol{\theta})$ is defined, we use XY to estimate $\hat{\boldsymbol{\theta}}$. Finding a good $\hat{\boldsymbol{\theta}}$ is the "learning" done by the model; using Y (as target outputs for $\mathbf{S}(\mathbf{x};\hat{\boldsymbol{\theta}})$) provides the "supervision". Finally, system validation tests $\mathbf{S}(\mathbf{x};\hat{\boldsymbol{\theta}})$ against performance requirements.

Roughly speaking, approximations are *good* in the traditional sense when they can be evaluated on (or extrapolate, or generalize to) inputs other than points in X with some degree of confidence. In pattern recognition, good is almost always defined as low apparent error rates on test data; in other functional approximation contexts (e.g., control), good usually means an acceptable mean squared error on test data, $E_{MSE}(X_{te}|X_{tr}) = \sum_{k=1}^{n_{te}} \left\|\mathbf{y}_k - \mathbf{S}(\mathbf{x}_k;\hat{\boldsymbol{\theta}})\right\|^2 / n_{te}$.

Conceptually, fuzzy models approximate $\mathbf{S}$ with the set of rules $\mathcal{R} = \{R_1,\ldots,R_M\}$. These rules are *if-then* rules whose outputs are combined by some form of approximate reasoning to produce an output for each input to the rule-base. Each rule R_i has a *premise* (*antecedent* or *left hand side*, LHS) with premise parameters, and a *consequent* (*right hand side*, RHS) with consequent parameters. These parameters, which may include M, the number of rules in $\mathcal{R}$, are the items we seek to estimate or need to define. Since the overall action of $\mathcal{R}$ as a function is to approximate $\mathbf{S}$, we may write the input-output relationship represented by $\mathcal{R}$ as $\boldsymbol{\mathcal{R}}(\mathbf{x};\hat{\boldsymbol{\theta}}) = \mathbf{S}(\mathbf{x};\hat{\boldsymbol{\theta}})$ to indicate this explicitly; and when $\mathbf{S}$ is a classifier function, we may write $\boldsymbol{\mathcal{R}}(\mathbf{x};\hat{\boldsymbol{\theta}}) = \mathbf{D}(\mathbf{x};\hat{\boldsymbol{\theta}})$. The basic MA and TS models are summarized in Figure 4.32.



❶ Input $\mathbf{x} \in \Re^p$
❷ Fuzzify $\{m^i_{kj}: \Re \mapsto [0,1]\}$
❸ LHS $T: [0,1]^p \mapsto [0,1]$
x_1 $m^i_{11} \cdots m^i_{1r}$ $D_1 \leftrightarrow \mathcal{L}_1$
x_k $m^i_{k1} \cdots m^i_{kr}$ $D_k \leftrightarrow \mathcal{L}_k$
x_p $m^i_{p1} \cdots m^i_{pr}$ $D_p \leftrightarrow \mathcal{L}_p$
$\alpha_1(\mathbf{x}) = \cap(m^1(\mathbf{x}))$
$\alpha_M(\mathbf{x}) = \cap(m^M(\mathbf{x}))$
❹ Output $\mathcal{R}(\mathbf{x}) \in \Re^q$
TS Model
$\{\mathbf{u}_i(\mathbf{x}); 1 \le i \le M\}$
$\mathbf{S}_{TS}(\mathbf{x}) = \frac{\sum_{i=1}^{M} \alpha_i(\mathbf{x})\mathbf{u}_i(\mathbf{x})}{\sum_{j=1}^{M} \alpha_j(\mathbf{x})}$
MA Model
$\mathbf{z}^i(\mathbf{x}) = \Psi(\alpha_i(\mathbf{x}), \mathbf{mo}^i); i = 1, \ldots, M$
$mo_{11} \cdots mo_{1s}$ $D_{o1} \leftrightarrow \mathcal{L}_{01}$
$\cup$
$mo_{q1} \cdots mo_{qs}$ $D_{os} \leftrightarrow \mathcal{L}_{0s}$
D_F
$\mathbf{S}_{MA}(\mathbf{x}) = \Theta(\boldsymbol{\alpha}(\mathbf{x}), \mathbf{Z}(\mathbf{x}), \cup, D_F)$

Figure 4.32 Architecture of the MA and TS Models

In step ❶, either model takes $\mathbf{x} \in \Re^p$ as an input vector. Step ❷ begins with the identification of the numerical range of each input variable. For k =1 to p, a numerical domain D_k is associated with a *linguistic variable* $\mathcal{L}_k$ that provides a semantic description of (r_k) subdomains of D_k. The number r_k is the *granularity* of $\mathcal{L}_k$. The maximum number of distinct LHSs that can be formed as rule antecedents from the r_k's is $M_{max} = r_1 \cdot r_2 \cdot \cdots \cdot r_p$. When $M = M_{max}$, we call $\mathcal{R}$ a *maximal* rule-base. Generally r can be a function of k, but we will usually use the simpler case $r_k = r$ for k = 1, ..., p.

The j-th subdomain of $\mathcal{L}_k$ represents an attribute value or *linguistic value*, say ℓ_{kj}, which is represented by a *premise membership function* (PMF) $m^i_{kj}:D_k \mapsto [0,1]$. The membership function m^i_{kj} in Figure 4.32 is indexed on i to associate it with rule R_i. We will drop the superscript unless it is explicitly needed.

In Figure 4.32 the membership functions all have symmetric triangular graphs, but this need not be - and very often is not - the case. Assume that each input variable has the same granularity r. The PMF set for the i-th rule, $\{m^i_{kj}:1 \le j \le r\}$, that represents the linguistic termset $\{\ell_{kj}:1 \le j \le r\}$ associated with variable k, $1 \le k \le p$, has many names in the literature: some writers call these functions *cognitive landmarks*; others call them a *membership termset*, but we prefer the more explicit name premise membership functions, which seems to be an accurate description of what they are. We assume that the union of positive supports of the $\{m^i_{kj}:1 \le j \le r\}$ covers D_k. When each of the p input domains is covered by a set of r unimodal, identically shaped, equally spaced PMFs that have the additional property that at each input value the sum of memberships is one, the system is called a *regular fuzzy system*, and the r^p rules in $\mathcal{R}$ are called a *complete* rule-base. Step ❷ is often referred to as *fuzzification* of the input domains.

Many writers call the positive supports of the $\{m^i_{kj}:1 \le k \le p; 1 \le j \le r; 1 \le i \le M\}$ a *fuzzy partition* of the "input space" $D_1 \times \cdots \times D_p \subset \Re^p$. This can be very confusing, as this terminology clashes directly with our earlier and quite different use of the same term in Section 2.1 concerning clustering, which produces a fuzzy partition $U \in M_{fcn}$ of a finite data set. We will use fuzzy partition as it is defined in equation (2.2).

Step ❸ comprises the action of the LHS of the rule-base, which is composed of the antecedent or premise parts of M rules $\mathcal{R} = \{R_i\}$. The premise parts of the rules operate on $\mathbf{x}$ and take the general form:

$$R_i^{LHS}: \ \alpha_i(\mathbf{x}) = T(\mathbf{m}^i(\mathbf{x})) = T(\underbrace{m^i_{1k_1}(x_1),\ldots,m^i_{pk_p}(x_p)}_{\mathbf{m}^i(\mathbf{x}) \in \Re^p}) \ , \ 1 \le i \le M. \quad (4.72a)$$

In (4.72a) $\alpha_i(\mathbf{x})$ is the *firing strength* (confidence level, degree of satisfaction) of rule i and T is any *T-norm* (intersection = $\cap$ in Figure 4.32 or AND) operator on $T:[0,1]\times[0,1] \mapsto [0,1]$. T norms can be extended by associativity to p arguments, so the calculation in

(4.72a) is well defined, and because T is valued in [0,1], $0 \le \alpha_i(\mathbf{x}) \le 1$. Our notation is a little sloppy because $\mathbf{m}^i(\mathbf{x})$ is not the value of a fixed vector field $\mathbf{m}^i$ on $\mathbf{x}$. Instead, the membership functions that *yield* the p values of $\mathbf{m}^i(\mathbf{x})$ for a particular $\mathbf{x}$ depend on different membership functions among the $\{m_{kj}\}$ as $\mathbf{x}$ runs through its domain. We use a similarly careless notation for *consequent membership functions* (CMFs) on the output or RHS of MA models, viz., $\mathbf{mo}^i(\mathbf{x}) \in \Re^q$, "o" meaning output.

The action of T on $\mathbf{m}^i(\mathbf{x})$ is to <AND> its p arguments; this is one aspect of approximate reasoning in the fuzzy system. The most common choices for T are the *minimum* or T_3 norm, and the *product* or T_2 norm that we met in Chapter 3, and will meet in Chang and Pavlidis (1977) in their seminal paper on fuzzy decision trees. For these choices (4.72a) is, more explicitly,

$$R_i^{LHS}: \ \alpha_i(\mathbf{x}) = T_3(\mathbf{m}^i(\mathbf{x})) = m^i_{1k_1}(x_1) \wedge \ldots \wedge m^i_{pk_p}(x_p) \quad , \ 1 \le i \le M; \quad (4.72b)$$

$$R_i^{LHS}: \ \alpha_i(\mathbf{x}) = T_2(\mathbf{m}^i(\mathbf{x})) = m^i_{1k_1}(x_1) \cdot \ldots \cdot m^i_{pk_p}(x_p) \quad , \ 1 \le i \le M. \quad (4.72c)$$

If, say, the j-th component in rule R_i is zero, $m^i_{jk_j}(x_j) = 0$, then $\alpha_i(\mathbf{x}) = T(\mathbf{m}^i(\mathbf{x})) = 0$ in (4.72b) or (4.72c). More generally, the same thing is true in (4.72a) using any T-norm, because T(α, 0) = 0 for any α in [0, 1]. We say that R_i *fires* (or is active, or is satisfied to the extent of the value) whenever $\alpha_i(\mathbf{x}) > 0$. A given input vector $\mathbf{x}$ in $\Re^p$ will probably never fire all M rules - instead, most of the $\alpha_i(\mathbf{x})$'s will be zero. If care is taken during fuzzification, it will never happen that *all* of the firing strengths are zero for any input $\mathbf{x}$. This is called *completeness* of the rule-base $\mathcal{R}$, a property that depends on the rules as well as the membership functions being used. Crisp decision tree rule sets are never complete because in a crisp decision tree which only interpolates its training data, when a non-training input is processed, there is no path for it to follow from the root to any node - that is, the firing strengths of all M crisp rules are zero.

Many (probably most) discussions about LHSs as in (4.72a) use a somewhat different terminology than ours. When A_{jk_j} is a fuzzy set such as "high", "tall", "long", etc. the premise clause it refers to is often stated as "if x_k is A_{jk_j}". Our preference is to use m_{jk_j} whenever we can, because the function - and only the function - *is* the fuzzy set. We will often state the LHS of rule i succinctly as " If $\alpha_i(\mathbf{x})$", understanding this to mean that the full structure of (4.72a) is used. This emphasizes the mathematical action of (4.72a), whereas its semantic interpretation allows users to provide a linguistic prescription for each rule.

Figure 4.33 illustrates the idea of T-norm aggregation for rule R_i. Shown there are two identical sets of 3 premise membership functions that represent two linguistic variables, $\mathcal{L}_1$ = temperature, and $\mathcal{L}_2$ = Speed. The linguistic values for temperature are ℓ_{11} = Low, ℓ_{12} = Med(ium), and ℓ_{13}= High; the linguistic values for Speed are ℓ_{21}= Slow, ℓ_{22}= Med(ium), and ℓ_{23}= Fast; m_{ij} is the membership function for ℓ_{ij}, i = 1,2 and j = 1,2, 3.

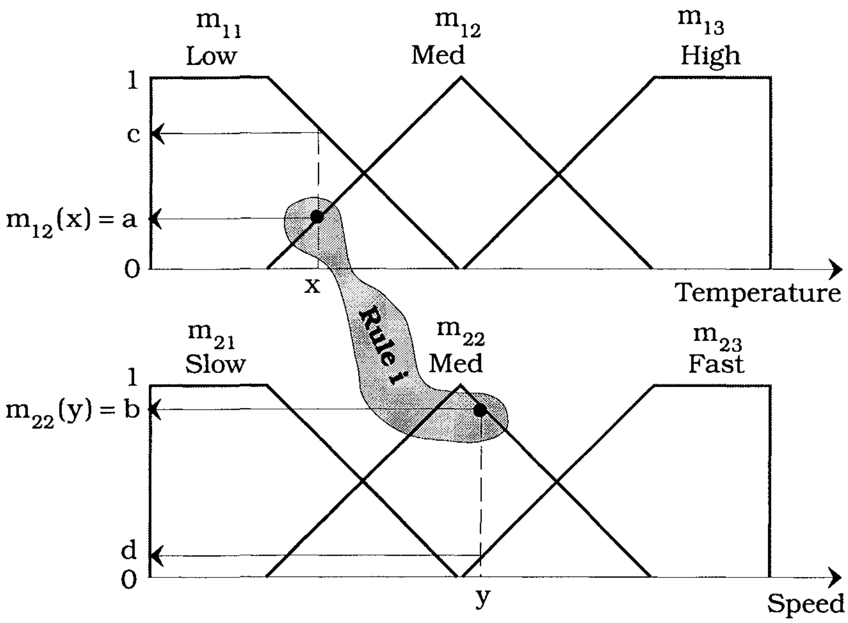


Figure 4.33 How inputs to the LHS of R_i are coupled by a T-norm

Let $\mathbf{x} = (x,y)^T$ denote an input vector for the p = 2 dimensional numerical domain associated with $(\mathcal{L}_1, \mathcal{L}_2)$, and suppose that the antecedents of rule R_i in $\mathcal{R}$ match this input pair as highlighted in Figure 4.33. There are three other possible matches in Figure 4.33 for the same **x**, because there are two active PMFs for each variable. This means that three other rules besides R_i will fire if these rules are in $\mathcal{R}$. The linguistic terms and membership function values that would be produced by these three rules are (Low, Medium) = (c, b), (Med, Fast) = (a,d) and (Low, Fast) = (c,d).

Here is how the LHS of R_i reads in words for the highlighted situation in Figure 4.33: "If (ℓ_{12} = Medium) and (ℓ_{22} = Medium)"; here is how your computer reads the same thing: "If $(m_{12}(x))$ and $(m_{22}(y))$". But you need to tell the computer what "and" means. So, choose a T-norm to represent intersection. This joins the two atomic clauses in the premise. If we use T_3 = minimum, the linguistic statement "If (ℓ_{1j1} = Medium) and (ℓ_{2k2} = Medium)" is translated, for the highlighted case shown in Figure 4.33, into the firing strength $\alpha_i(\mathbf{x}) = T_3(m_{12}(x), m_{22}(y)) = a \wedge b = a$. If you choose the product for "and", $\alpha_i(\mathbf{x}) = T_2(m_{12}(x), m_{22}(y)) = a \cdot b = ab$.

Step ❹ in Figure 4.32 produces the output vector $\mathbf{S}(\mathbf{x})$. For the TS model, the functions $\{\mathbf{u}_i : \Re^p \mapsto \Re^q : 1 \le i \le M\}$ comprise the RHS of the rule-base. Each $\mathbf{u}_i$ is a vector field whose components are scalar fields of some specified form (e.g., constant, linear, affine, quadratic, polynomial, Gaussian, exponential, etc.). It is common - but not necessary - to specify that all the $\mathbf{u}_i$'s have the same functional form.

When the $\mathbf{u}_i$'s are all polynomials of the same order (i.e., all the components of the output functions are, respectively, constant, affine, quadratic, etc.), we refer to the TS model as a *0-th, 1-st, 2-nd, ... etc., order TS model.* Within this class - as is the case in many other branches of applied mathematics - the first order (affine) models are by far the most popular and heavily used. For example, rule extraction by clustering in XY makes sense for exactly this case when the clustering model can produce flat (affine subspace) prototypes (lines, planes, etc.), because these prototypes match the shape of the graphs of the affine output functions being estimated as the RHS's of a 1-st order TS model.

The output of the TS model is a convex combination of its M output functions and firing strengths,

$$\mathbf{S}_{TS}(\mathbf{x}) = \frac{\sum_{i=1}^{M} \alpha_i(\mathbf{x}) \cdot \mathbf{u}_i(\mathbf{x})}{\sum_{j=1}^{M} \alpha_j(\mathbf{x})} \quad . \tag{4.73a}$$

There is an important 0-th order variation of the MISO TS model that replaces $\mathbf{u}_i(\mathbf{x})$ in (4.73) with a fixed number. When the number $u_i(\mathbf{x}) = h_i$ is the center of gravity (of the independent variable of consequent membership function mo) of a single Mamdani style CMF, this is called the method of *height defuzzification* in the MA model, and (4.73a) takes the simpler form

$$\mathbf{S}_{TS}(\mathbf{x}) = \frac{\sum_{i=1}^{M} \alpha_i(\mathbf{x}) \cdot h_i}{\sum_{j=1}^{M} \alpha_j(\mathbf{x})} \quad . \tag{4.73b}$$

When the output function of R_i is a crisp singleton as in (4.73b), it easier to find the parameters of the ensuing model, but the price of simplicity is that it weakens the approximation capabilities of the model. See Sugeno and Yasukawa (1993) for a nice method of training the 0-th order TS model. An even simpler case arises when the LHS membership functions are *regular* (for each of the p input variables, all PMFs are symmetric, triangular membership functions which cross each other at 0.5), for in this case the sum of firing strengths in the denominator will always be 1.

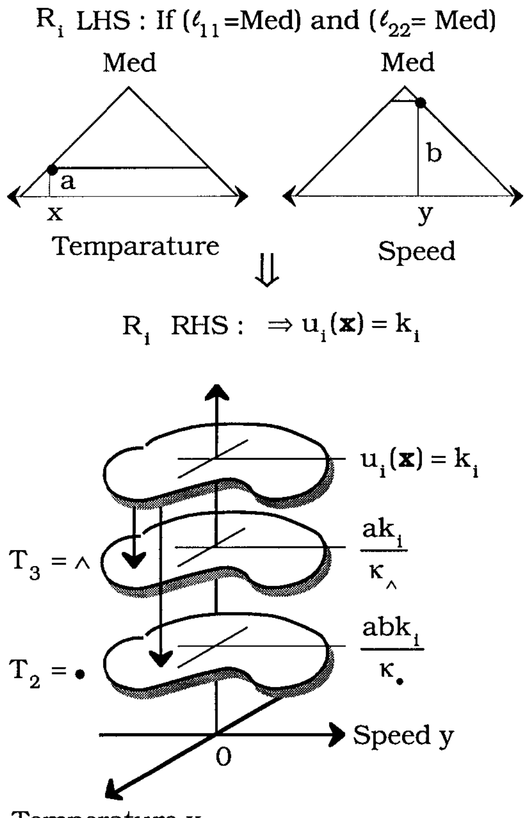

R_i LHS : If (ℓ_{11}=Med) and (ℓ_{22}= Med)
Med
Med
a
b
x
y
Temparature
Speed
⇓
R_i RHS : $\Rightarrow u_i(\mathbf{x}) = k_i$
$u_i(\mathbf{x}) = k_i$
$\frac{ak_i}{\kappa_\wedge}$
$T_3 = \wedge$
$\frac{abk_i}{\kappa_\bullet}$
$T_2 = \bullet$
Speed y
0
Temperature x

Figure 4.34 How the left and right sides of R_i are coupled in the 0-th order TS model for the highlighted input case shown in Figure 4.33 using the minimum and product for the T-norm

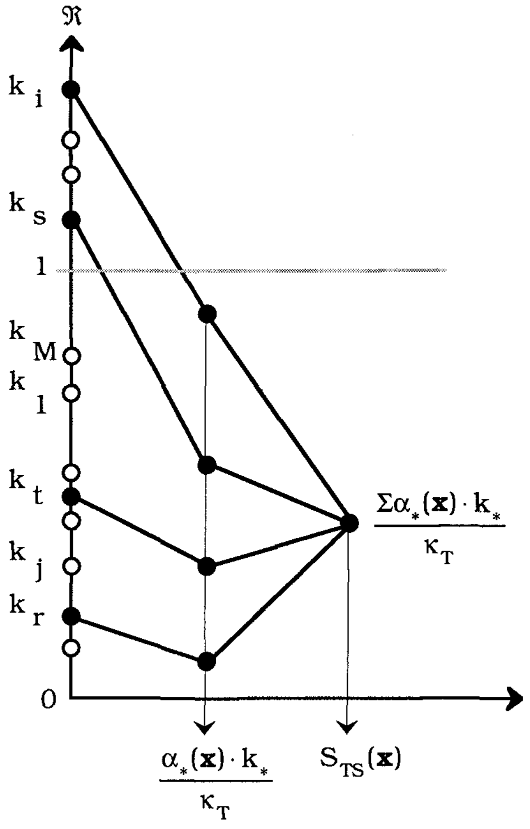
Figure 4.34 illustrates how the clauses in the left side of rule R_i in Figure 4.33 are coupled through Takagi-Sugeno implication for the 0-th order model, where the i-th output function is a constant surface, $u_i(\mathbf{x}) = k_i$. Let $\kappa_T(\mathbf{x})$ denote the denominator of (4.73), $\kappa_T(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i(\mathbf{x})$. The value of $\kappa_T(\mathbf{x})$ depends on your choice for the T-norm; here we consider $T = T_3 = \wedge$ or $T = T_2 = \bullet$. When R_i fires, the effect of using the T_3 norm to compute the firing strength is to lower the corresponding surface $u_i(\mathbf{x}) = k_i$ (remember that a and b in Figure 4.33 are ≤ 1, $a = a \wedge b$, and note that $a/\kappa_\wedge(\mathbf{x}) \le 1$ because a is one term of the denominator $\kappa_\wedge(\mathbf{x})$) to the new, smaller constant $ak_i/\kappa_\wedge(\mathbf{x})$; and for the T_2 = product T-norm, the surface may move even further down (or up, depending on the relationship between $a/\kappa_\wedge(\mathbf{x})$ and $ab/\kappa_\bullet(\mathbf{x})$), to $abk_i/\kappa_\bullet(\mathbf{x})$.

If $\mathcal{R}$ is maximal (i.e., contains 9 rules here) and each input value is evaluated by two membership functions, there are 3 other pictures like Figure 4.34 for the other three rules that would fire for this $\mathbf{x}$, that is, for the three pairs of membership values from the currently active PMFs shown in Figure 4.33. Suppose that the other three rules that fire are R_r, R_s and R_t. Applying equation (4.73) with the minimum and product T-norms results in the outputs

$$S_{TS}(\mathbf{x}) = \frac{\sum_{i=1}^{M} \alpha_i(\mathbf{x}) \cdot u_i(\mathbf{x})}{\sum_{j=1}^{M} \alpha_j(\mathbf{x})} = \left. \frac{ak_i + dk_r + ck_s + dk_t}{a + d + c + d} \right|_{T=\wedge} \text{; and}$$

$$S_{TS}(\mathbf{x}) = \frac{\sum_{i=1}^{M} \alpha_i(\mathbf{x}) \cdot u_i(\mathbf{x})}{\sum_{j=1}^{M} \alpha_j(\mathbf{x})} = \left. \frac{abk_i + adk_r + cbk_s + cdk_t}{ab + ad + cb + cd} \right|_{T=\bullet} .$$

As the input $\mathbf{x}$ to Figures 4.33 and 4.34 changes, there is no change in the values of the constants $\{k_i\}$, but a, b, c and d may change for different inputs that fire the same rules; the four k_i's and a, b, c and d may all change when different rules in $\mathcal{R}$ fire. You would be correct to *imagine that the upper surface in Figure 4.34 is fixed* (once k_i is chosen), and that for M rules, there will be (at most) M such constant surfaces at the heights $\{k_i\}$. As the input changes, the rules fired select which subset (of four or less) of these surfaces to use, the firing strengths decrease the heights of the surfaces chosen above the horizontal (input) plane, and (4.73) combines the current set of heights to get the resultant output. This is illustrated in Figure 4.35, where the four rules R_i, R_r, R_s and R_t are the ones selected as matches on the LHS (that is, the ones that are fired). We emphasize that this figure illustrates the action of the TS rule for just one input - it does *not* illustrate how the output of the TS system "looks" over all of X.



$\Re$
k_i
k_s
1
k_M
k_1
k_t
k_j
k_r
0
$\frac{\Sigma\alpha_*(\mathbf{x}) \cdot k_*}{\kappa_T}$
$\frac{\alpha_*(\mathbf{x}) \cdot k_*}{\kappa_T}$
$S_{TS}(\mathbf{x})$

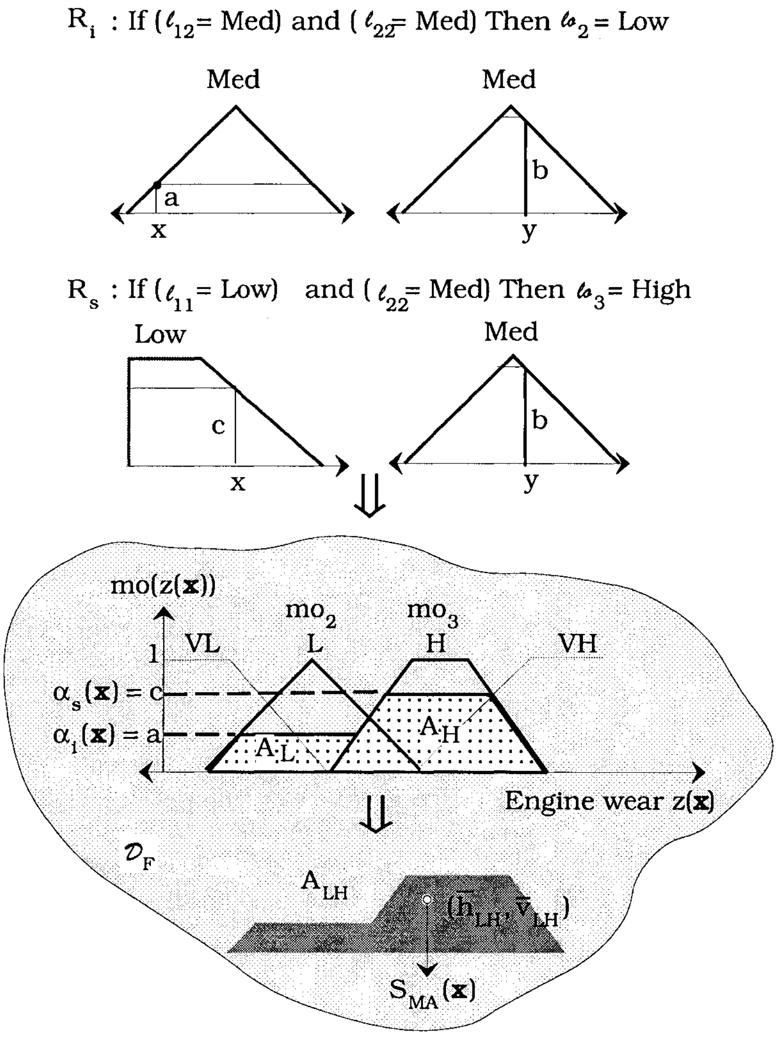
Figure 4.35 An output for the 0-th order TS model

It gets pretty hard to draw figures like 4.35 for more complicated output functions, but the principle is identical. If the output functions in a TS system were all quadratics in two variables, for example, the i-th of the M surfaces would be the graph of the function $u_i(\mathbf{x}) = \mathbf{x}^T A_i \mathbf{x} + \langle \mathbf{b}_i, \mathbf{x} \rangle + k_i$. For a given input, the fired rules would again select subsets of these surfaces, and as in Figure 4.35, the selected ones would be scaled down by their corresponding firing strengths, and then their values at this point in $\Re^2$ added in accordance with equation (4.73) to get the TS output. In the most complicated TS model, each of the M surfaces could be a different type; one might be constant, one a quadratic, another a Gaussian, and so on. As you can see, the approximation we are building with fuzzy systems can have pretty complex components - and the 0-th order TS model is the easiest case to understand!

Step back from Figures 4.33 and 4.34 and ask - what things do we need to learn from training data to make this easiest of all TS systems work? On the LHS, we need 6 PMFs. For this example there are 4 trapezoidal fuzzy numbers, but they are special trapezoids - each one needs 2 parameters for its "shoulder", so there are 8 parameters needed for the 4 trapezoids. There are 2 triangular fuzzy numbers (each needs 3 parameters). So we must estimate or (at least adjust for optimal performance) 14 parameters for the premise membership functions. Each consequent k_i is also needed. Since there can be at most $r^2 = 9$ rules for this system, we need 9 parameters for the CMFs, so there are 14+9 = 23 parameters associated with the membership functions. And we have already decided that both LHS granularities are r = 3, that we will use the types of membership and output functions shown, and that we have chosen some T-norm. All of these choices face the system designer for the 0-th order TS model. And it is the simplest form of fuzzy system we discuss - now you can see why it is so popular! When we design fuzzy decision tree classifiers (which are often equivalent to such a system), many of these decisions are eliminated from the user's view. We will return to the geometry underlying this model when we get to subsection F.

Step ➍ in the MA model is considerably *more* complicated than for the TS model. The LHS works just as we have illustrated in Figures 4.32 and 4.33 - it is identical to the LHS of the TS rule-base. But the RHS of the MA model is very different. Roughly speaking, the sequence of operations on the RHS is (i) fuzzification; (ii) inferencing; (iii) aggregation; and (iv) defuzzification. We briefly discuss each of these steps.

As shown in Figure 4.32, each *output* variable z_k, k =1,..., q, is fuzzified by assigning it a linguistic variable $\mathcal{L}o_k$, a linguistic termset $\{\ell o_{kj}\}$ and corresponding set of consequent membership functions $\{mo^i_{jk}\}$ of, say, granularity s, i = 1, ..., M, k = 1 to q, j = 1 to s. These CMFs reside at the RHS of every rule. When an input is submitted to this system, the LHS of the MA model produces a positive firing strength $\alpha_i(\mathbf{x})$ for each rule fired. Now the role of the firing strength is somewhat different, for we use it to *enter* the CMF set on the RHS of the rule-base. This is illustrated in Figure 4.36, which depicts defuzzification by the center of gravity (COG) method.



R_i : If (ℓ_{12}= Med) and (ℓ_{22}= Med) Then ℓ_{o2} = Low
Med
Med
a
b
x
y
R_s : If (ℓ_{11}= Low) and (ℓ_{22}= Med) Then ℓ_{o3}= High
Low
Med
c
b
x
y
mo(z(**x**))
mo_2
mo_3
1
VL
L
H
VH
$\alpha_s(\mathbf{x}) = c$
$\alpha_i(\mathbf{x}) = a$
A_L
A_H
Engine wear z(**x**)
$\mathcal{D}_F$
A_{LH}
$(\bar{h}_{LH}, \bar{v}_{LH})$
$S_{MA}(\mathbf{x})$

Figure 4.36 One of the million ways to defuzzify MA rules : area COG defuzzification

Rules R_i and R_s, fired for the input **x** shown in Figure 4.33, are shown in the top portion of Figure 4.36. The case illustrated uses the minimum for the T-norm, so rule i has firing strength a and rule s has firing strength c. These firing strengths are carried to the RHS of the MA rules, where the single output variable is the linguistic variable $\mathcal{L}_o$ ="engine wear". The domain of $\mathcal{L}_o$ has been partitioned (in the fuzzy systems sense) into 4 linguistic values: ℓ_{o1} ="Very Low (VL)", ℓ_{o2} = "Low (L)", ℓ_{o3} = "High (H)", and ℓ_{o4} = "Very High (VH)", with corresponding consequent membership functions mo_1, mo_2, mo_3, mo_4 spread across the expected numerical output range, whose variable name is z in Figure 4.36. For the two rules fired the memberships we will look up correspond to the linguistic values that appear on the RHS of the rules. Suppose the rules are:

R_i : If (ℓ_{12} = Med) and (ℓ_{22} = Med) Then $\mathcal{L}o$ (=Engine wear) = Low
R_s : If (ℓ_{11} = Low) and (ℓ_{22} = Med) Then $\mathcal{L}o$ (=Engine wear) = High

Then we pick out the CMFs corresponding to these two output linguistic values, and *operate on them* at their respective levels of firing strength, i.e., at the values $z(\alpha_i(\mathbf{x})) = a$ and $z(\alpha_s(\mathbf{x})) = c$. We say operate on them because what happens next depends on the inferencing operator you choose. In Figure 4.32, the output of rule i is denoted as $\mathbf{z}^i(\mathbf{x}) = \Psi(\alpha_i(\mathbf{x}), \mathbf{mo}^i)$, where the symbol Ψ stands for the operator used to produce the output, which is "some function of" the arguments shown, which are the firing strength $\alpha_i(\mathbf{x})$ from the LHS of rule i, and the q membership functions that fuzzify the RHS of rule i.

There are two methods for combining MA rules. *Rule-based inferencing* uses all M rules without segregation by linguistic values (unfired rules will make no contribution to the output, however). Each rule is represented by a relation, and the union of all the rules gives a composite relation for the entire rule-base. Then inferencing produces a single output fuzzy set which is defuzzified by one of many methods such as the COG. This scheme is somewhat analogous to TS inferencing in that both use the entire rule-base. *Composition-based inferencing* is more complicated. In this scheme each fired rule produces a clipped (modulated) version of the associated CMF. The modulated CMFs belonging to each (fired) linguistic output are then aggregated with a union operator, again resulting in one output membership function which is defuzzified by any of the various defuzzification schemes.

More specifically, the M firing strengths $\boldsymbol{\alpha}(\mathbf{x}) = (\alpha_1(\mathbf{x}), \ldots, \alpha_M(\mathbf{x}))^T$, $\alpha_i(\mathbf{x}) \in [0,1]$ for all i, and the M consequent membership function values $\mathbf{Z}(\mathbf{x}) = (\mathbf{z}^1(\mathbf{x}), \ldots, \mathbf{z}^M(\mathbf{x}))^T$, $\mathbf{z}^i(\mathbf{x}) \in \Re^q$ for all i, are defuzzified with D_F, typically a *center of gravity* (COG) type calculation such as shown in Figure 4.36. In Figure 4.36 the selected membership functions are clipped, and their areas A_L and A_H are found. These areas can be treated separately, summed, unioned, intersected, etc.; and, they need not be trapezoidal - some writers make them triangular, etc. In Figure 4.36 we illustrate the union method, where the area centroid $(\bar{h}_{LH}, \bar{v}_{LH})$ of the union of A_L and A_H is found, and the horizontal coordinate $\bar{h}_{LH}$ is taken as the MA output $S_{MA}(\mathbf{x})$ for input $\mathbf{x}$. An important special case of Figure 4.36 occurs when the MA system has singleton CMFs. If centroid defuzzification is used in this case, the MA system is equivalent to a TS system with constant RHS output functions - that is, a 0-th order TS system.

This brief description of the MA model will almost surely leave you gasping - how can it, how does it work? Our brief treatment of fuzzy systems hardly does this topic justice, but other volumes in this Kluwer handbook series have extensive discussions of both models (Nguyen and Kreinovich, 1998, Tanaka and Sugeno, 1998, Yager and Filev, 1998). Additional references on this topic that we have found helpful include Driankov et al. (1993) and Klir and Yuan (1995). For us it suffices, at least initially, to write the output of the MA model as $\boldsymbol{\mathcal{R}}_{MA}(\mathbf{x}) = \Theta(\boldsymbol{\alpha}(\mathbf{x}), \mathbf{Z}(\mathbf{x}), \cup, D_F)$, where operator Θ depends on choices made by the system designer. We do have examples of the MA scheme to present, and when we discuss them we will try to explain each one explicitly, case by case (or, at least, reference by reference !).

When either type of fuzzy system is used to approximate a classifier function, its outputs will be label vectors. Hardening non-crisp label vectors as in (1.15) may be done after defuzzification when the output of $\boldsymbol{\mathcal{R}}$ is a soft label vector, i.e., when $\boldsymbol{\mathcal{R}}$ represents a soft classifier function, and a crisp classification is required. Summarizing, we now have a formal model of both the LHS and RHS of each rule in $\mathcal{R}$, which takes the general form

$$R_i: \ \alpha_i(\mathbf{x}) \Rightarrow \begin{cases} \text{TS: } \mathbf{u}_i(\mathbf{x}) \\ \text{MA: } \Theta(\boldsymbol{\alpha}(\mathbf{x}), \mathbf{Z}(\mathbf{x}), \cup, D_F) \end{cases}, \ 1 \le i \le M \quad . \tag{4.74}$$

We cannot write the rule-based system in (4.70) or (4.71) using the formalism of (4.74) because the linguistic terms used there are not related to measurable numerical variables (and so, fuzzification of the input domains is not possible), but decision trees that handle numerical variables will fit nicely into this framework. For example, the simple set of rules in (4.63) can be made much more mysterious looking by defining a set of 3 crisp premise membership functions over, say, the extended input domain $[0, \infty)$ as follows:

$$m_{\text{chicken}}(n_L) = m_{11}(n_L) = \begin{cases} 1; & n_L = 2 \\ 0; & \text{otherwise} \end{cases} \quad ;$$

$$m_{\text{crab}}(n_L) = m_{12}(n_L) = \begin{cases} 1; & n_L = 8 \\ 0; & \text{otherwise} \end{cases} \quad ;$$

$$m_{\text{fish}}(n_L) = m_{13}(n_L) = \begin{cases} 1; & n_L = 0 \\ 0; & \text{otherwise} \end{cases} \quad .$$

With these PMFs, the crisp rule-based classifier at (4.63) becomes

$$\text{If } (m_{11}(n_L)) \text{ Then } \mathbf{D}_{DT}(n_L) = \mathbf{e}_1 \quad ; \qquad (4.63a')$$

$$\text{If } (m_{12}(n_L)) \text{ Then } \mathbf{D}_{DT}(n_L) = \mathbf{e}_2 \quad ; \qquad (4.63b')$$

$$\text{If } (m_{13}(n_L)) \text{ Then } \mathbf{D}_{DT}(n_L) = \mathbf{e}_3 \quad . \qquad (4.63c')$$

This isn't a very exciting system, but it's simple, and displays the relationship between the rules and the notation we will use for more complicated models. We will return to several aspects of the use of the rule-based system in (4.74) for approximation of functions in subsection F.

E. The Chang - Pavlidis fuzzy decision tree

Fuzzification of decision trees follows two paths; softening the training process (how to build the tree), and softening the decision functions at internal nodes (how to use the tree). Chang (1976) was apparently the first person to write about fuzzy decision trees. Chang and Pavlidis (1977) is the seminal archival paper on fuzzy decision trees, and it was one unknowing precursor of the now widely known fuzzy systems approach discussed in subsection 4.6.D. The origin of probabilistic decision trees is much older; a specific reference for this depends on what you regard as a probabilistic decision tree. Suffice it to say that Duda and Hart (1973) mention this topic in connection with sequential decision theory in statistics, which dates to the early part of the 20th century.

We begin the exposition of *fuzzy decision trees* (FDT) by returning to the case of the fully expanded crisp decision tree. When $|V_L| = M$, each of the M $\geq$ c paths from the root to a pure leaf corresponds to a crisp rule in $\mathcal{R}$. Two aspects of this need discussion: representation of the choice of path by node decision functions; and aggregation of the edge weights along the path to compute the firing strength of an activated rule.

Figure 4.37 depicts an input **z** traversing the path $p(v_1, v_3, v_6, \ldots, v_s, v_{L_i})$ from root v_1 to leaf v_{L_i} which bears crisp label $\mathbf{e}_j$, with the result that crisp rule R_i fires with firing strength=1, labeling input **z** as class j, $\mathbf{D}_{DT}(\mathbf{z}) = \mathbf{e}_j$. We have appended 1's to each edge along this path, which can be interpreted as weights assigned to the chosen edges by functions residing in the internal nodes that select the correct edge for this **z**. The firing strength value of 1 for rule R_i can be calculated as either the minimum or product of the 1's along the path. The untraveled paths have 0's on their edges for this **z**, so the firing strengths along these M-1 paths will be zero when intersection is done with any T-norm.

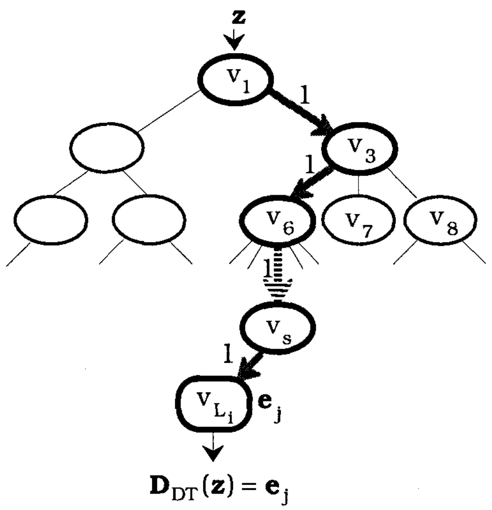


Figure 4.37 Firing strength in a crisp decision tree

In Figure 4.37 internal node v_3 has 3 children, internal node v_6 has 5 children, etc. Let T have n nodes, $M \le n-1$ leaves, and thus, n-M internal nodes. Without loss of generality, suppose that internal node v_k has p children. Chang and Pavlidis (1977) let X represent the domain of node inputs (and are not explicit as to the data type, which seems implicitly to be numerical feature vectors), and call any function $\boldsymbol{\phi}_k = (\Phi_{k1}, \ldots, \Phi_{kp}): X \mapsto [0,1]^p$ a *fuzzy decision function* for v_k. The p values $\{\phi_{ki}(\mathbf{x}): i = 1, \ldots, p\}$ produced by this function can be thought of as edge weights or path indicator values associated with v_k for this $\mathbf{x}$.

Chang and Pavlidis use this idea as a basis for defining *fuzzy decision trees* as decision trees with fuzzy decision functions at each internal node. They did not exclude the zero vector from the range of internal node functions, but we think they meant to, for otherwise the possibility of $\mathbf{x}$ being trapped at an internal node exists if no exit edge has a positive weight. In any case, we call $\boldsymbol{\phi}_k : X \mapsto [0,1]^p - \{\mathbf{0}\} = N_{pc}$ (see equation (1.1)) a soft *node decision function* at internal node $v_k \in V_I$. We use the notation of Chapter 1 here because it is correct and convenient, but $\boldsymbol{\phi}_k$ is not a classifier function in the sense used in this book. The job $\boldsymbol{\phi}_k$ has, in crisp decision trees, is to identify the outgoing edge from v_k that an input should take as it makes its way towards a leaf - in other words, $\boldsymbol{\phi}_k$ is a crisp membership function that represents the output of the internal computation or comparison made by the crisp decision function at or in this node.

If the range of $\boldsymbol{\phi}_k$ is N_{hc} for k = 1,..., n-M, the fuzzy decision tree reduces to a crisp decision tree, so crisp decision trees do have decision functions at their internal nodes, but they are usually represented differently, as for example, in the stipulation of a hyperplane condition. To illustrate, Figure 4.38 shows an expanded view of the situation at node v_6 of Figure 4.37. The node function $\boldsymbol{\phi}_6 : X \mapsto N_{h5}$, so it produces, for any input in its domain, a crisp label vector $\boldsymbol{\phi}_6(\mathbf{x}) \in N_{h5}$. In Figure 4.38, $\boldsymbol{\phi}_6(\mathbf{x}) = (0,0,1,0,0)^T$.

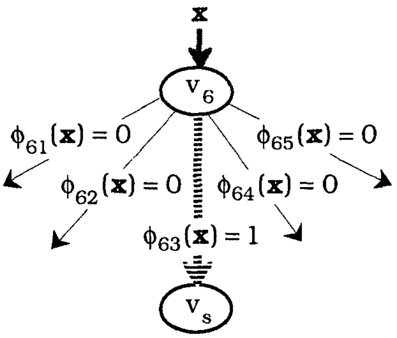


Figure 4.38 A node decision function in a crisp decision tree

Don't confuse internal node decision functions such as $\boldsymbol{\phi}_k$ with internal node splitting functions such as $\iota_{ent,k}$ in ID3: decision functions make decisions (assign edge weights) at internal nodes during classifier operation, while splitting functions make decisions *about how to split internal nodes during tree construction.*

Returning to Figure 4.37, we can now write the crisp rule for the path shown there as

$$\begin{array}{ll} \text{IF} & \phi_{13}(\mathbf{z}) = 1 \text{ and } \phi_{36}(\mathbf{z}_3) = 1 \text{ and} \cdots \phi_{6s}(\mathbf{z}_6) = 1 \text{ and } \phi_{s,L_i}(\mathbf{z}_s) = 1 \\ \text{THEN} & \mathbf{D}_{DT}(\mathbf{z}) = \mathbf{e}_j \end{array} \quad , \qquad (4.75)$$

where arguments of the different node functions are subscripted to indicate that they may not all be the same. Recognizing that "and" can replaced by any intersection operator (T- norm), equation (4.75) can be written more compactly as

$$T(\phi_{13}(\mathbf{z}), \phi_{36}(\mathbf{z}_3), \cdots, \phi_{6s}(\mathbf{z}_6), \phi_{s,L_i}(\mathbf{z}_s)) \Rightarrow \mathbf{D}_{DT}(\mathbf{z}) = \mathbf{e}_j . \qquad (4.75')$$

Equation (4.75) represents the action taken when the input vector matches the premise of rule R_i - in other words, it *is* rule R_i, which is seen by noting the subscript L_i on the last argument of T in (4.75). Now define

$$\alpha_i(\mathbf{z}) = T(\phi_{13}(\mathbf{z}), \phi_{36}(\mathbf{z}_3), \cdots, \phi_{6s}(\mathbf{z}_6), \phi_{s,L_i}(\mathbf{z}_s)),\ 1 \le i \le M. \tag{4.76}$$

Because of the boundary property T(a, 1) = 1 ⇔ a=1 of any T-norm, $T(\phi_{13}(\mathbf{z}), \phi_{36}(\mathbf{z}_3), \cdots, \phi_{6s}(\mathbf{z}_6), \phi_{s,L_i}(\mathbf{z}_s)) = 1 \Leftrightarrow \phi_{**}(\mathbf{z}_*) = 1\, \forall *$. In view of this, we see that $\alpha_i(\mathbf{z}) = 1$ when, and only when, all of the arguments of T in (4.76) are 1. That is, rule R_i fires if and only $\alpha_i(\mathbf{z}) = 1$. Conversely, if any of the arguments of the T-norm are 0, then $\alpha_i(\mathbf{z}) = 0$ and rule i will not fire - that is, the path $p(v_1,\ldots v_{L_i})$ from the root to leaf i will not be used in a crisp decision tree unless the firing strength of its rule is 1. As an historical aside, the term firing strength was not well established, and Chang and Pavlidis called $\alpha_i(\mathbf{z})$ the *fuzzy decision value* of the path $p(v_1, v_{L_i})$ from the root to leaf i in the tree, We call $\alpha_i(\mathbf{z})$ the *firing strength* of R_i for input **z**.

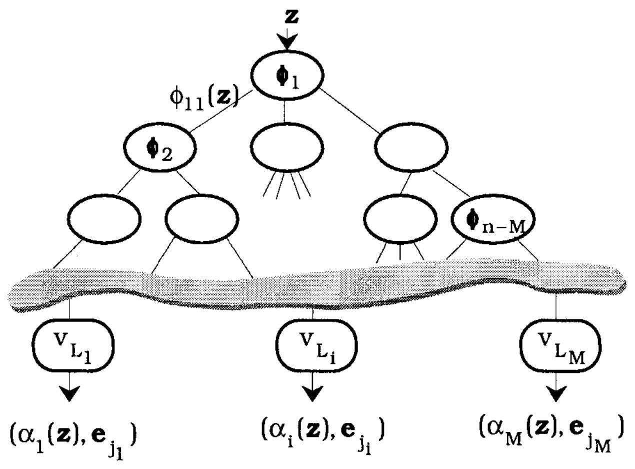

z
$\phi_{11}(\mathbf{z})$
ϕ_1
ϕ_2
ϕ_{n-M}
v_{L_1}
v_{L_i}
v_{L_M}
$(\alpha_1(\mathbf{z}), \mathbf{e}_{j_1})$
$(\alpha_i(\mathbf{z}), \mathbf{e}_{j_i})$
$(\alpha_M(\mathbf{z}), \mathbf{e}_{j_M})$

Figure 4.39 The fuzzy decision tree of Chang and Pavlidis (1977)

Now suppose, as Chang and Pavlidis do, that a fully expanded tree has been developed, and that the internal node decision functions are valued in N_{pc}. What path does **z** take in this case? Conceptually **z** can traverse all n-M paths, and can arrive at all M leaves in the tree. When this happens, we may imagine that all M of the rules fire, each producing a firing strength $0 \le \alpha_i(\mathbf{z}) \le 1$. Chang and Pavlidis discuss two cases. They call the decision tree that uses the T_3-norm (the minimum) to produce firing strengths a fuzzy decision tree, and when the edge weights along a path are multiplied together (so the T-norm is the product or T_2-norm), they call the tree a probabilistic model. Figure 4.39 summarizes the structure we henceforth call the *Chang-Pavlidis* (CP) fuzzy decision tree.

Each leaf of the decision tree in the lower part of Figure 4.39 has two pieces of information attached to it: $\alpha_i(\mathbf{z})$, the firing strength or decision value along the path from v_1 to v_{L_i} in the chosen T-norm, and $\mathbf{e}_{j_i}$, one of the c crisp label vectors for the classes in the training data. Chang and Pavlidis did *not* aggregate firing strengths across the M leaves, nor did they collect leaves with like labels and aggregate these, etc. Instead, they defined the output of the tree in Figure 4.39 as the crisp label associated with the largest firing strength,

$$\mathbf{D}_{DT}^{CP}(\mathbf{z}) = \mathbf{e}_{j_k} \Leftrightarrow \alpha_k(\mathbf{z}) = \max_{1 \le i \le M}\{\alpha_i(\mathbf{z})\} \quad . \tag{4.77}$$

This is a crisp classifier even though the tree that defines it is a soft decision tree. $\mathbf{D}_{DT}^{CP}$ simply assigns $\mathbf{z}$ to the class that has the highest firing strength in $\mathcal{R}$. There are some obvious generalizations of this structure. For example, the tree in Figure 4.39 is pure, but it is well known that pruning decision trees improves them. Some of the leaves in subtrees obtained this way will not be pure, and the crisp label vectors for these leaves will be replaced by soft label vectors. Each of these could of course be hardened in the usual way, and then (4.77) would still apply. A more interesting possibility is to aggregate the evidence residing in the firing strengths of all the rules with a T-conorm or some other form of aggregation such as a weighted mean.

Chang and Pavlidis spend the bulk of their paper on theoretical results about algorithms to *search* a given fuzzy decision tree for the path that leads to the solution shown in (4.77) without enumerating all the paths (remember, this was 1977 - computers were still tiny in power - but huge in physical size!). They defined top down search of a fuzzy decision tree as a search from the root to a leaf that always makes the greedy choice - that is, takes the highest value available - at each edge in the path. They give the simple example shown in Figure 4.40 to illustrate the failure of top down search to find the solution of (4.77). Taking the greedy path from the root accumulates the decision values 0.6 and 0.5, leading to leaf 2 with decision value 0.30 in the T_2 norm, which is not the solution of (4.77). Using the T_3 norm, the greedy path leads to one of two equally correct solutions.

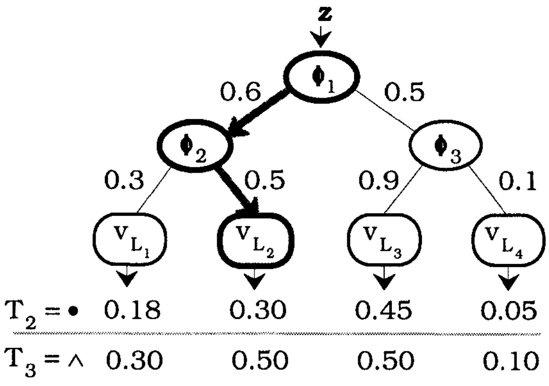


Figure 4.40 Top down search failure (Chang and Pavlidis, 1977)

Top down search of a tree for c classes is almost always O(c), and as in Figure 4.40, can lead to the wrong leaf. There is also a guaranteed O(c) bottom up search. Chang and Pavlidis, wanting a faster method, discovered a *branch and bound backtracking* (BBB) algorithm that finds the path of maximal firing strength in O(c) time, worst case, and in $O(\log_2 c)$ time in the best case. You may think these complexities trivial in 1999 and beyond, and for many problems (Iris, for example, with c=3 classes) perhaps they are. On the other hand, Wang and Suen (1987) process labeled character recognition data with c = 3200 character classes, so evaluation time can become important. Moreover, the number of leaves can be far greater than the number of terminals (Wang and Suen, 1984), as we demonstrate in Example 4.14, so this is a good result.

A fairly clever and interesting secondary result in Chang and Pavlidis is that any linear classifier defined by a set of hyperplanes in $\Re^2$ for a c-class problem can be approximated arbitrarily well by a CP fuzzy decision tree with trivial comparisons alone (i.e., comparisons such as $z_i \le k_i$). They do not give any methods for *finding* or pruning trees, nor are they very specific about internal node decision functions. They do, however, compare their method with both crisp and probabilistic (i.e., using the product of the edge weights instead of the minimum to get firing strengths) classifier trees on the problem of discriminating between handwritten numerals "5" and "9", and their fuzzy decision tree does a little better than the other two.

F. Fuzzy relatives of ID3

Most of the recent papers on fuzzy decision trees are related to either ID3 or some other induction algorithm (how to get trees) ; or they generalize the CP tree (how to define soft decision functions and approximate reasoning along paths in trees). Some of the fuzzy generalizations of ID3 discussed below replace the impurity function ι_{ent} with a measure of fuzziness to assess potential splits at internal nodes, while others continue to use ι_{ent} for node splitting, but apply it to fuzzy quantities instead of probabilities due to relative frequencies.

Wang and Suen (1983, 1987) proposed a set of modifications to the basic CP decision tree, and Suen and Wang (1984) introduced a new crisp hierarchical clustering algorithm called ISOETRP (roughly, ISODATA driven by an entropy objective function) that essentially competes with ID3 as a crisp decision tree building algorithm. These three papers together provide a way to construct decision trees, make them fuzzy, prune them, and infer decisions in a slightly different way than by equation (4.77). The clustering algorithm is interesting and has some nice wrinkles, so we provide a brief discussion of it first.

The basic premise in Suen and Wang (1984) is that node splitting can be viewed as top down crisp hierarchical clustering. They argue that the clustering objectives of the SAHN type algorithms that were discussed in Chapter 3 are not relevant to good node splits from the standpoint of decision tree design. Their method acknowledges the importance of Quinlan's (1983) use of ι_{ent} for node splitting, and their objective function ("GAIN") for node splitting uses ι_{ent} as a building block. The overall node splitting function in ISOETRP is a ratio of *a function of* node entropy to a measure of cluster overlap for potential splits (clusters) of the cases at the node at hand, and this function plays the role of $\Delta\iota_{ent,k}(S; \mathbf{p}_k)$ in ID3 equation (4.69).

The basic idea is to create an initial set of clusters at a node. Then their "GAIN" function uses the labels of these cases to measure the entropy reduction due to this split, normalized by a measure of cluster overlap. Following this, the clusters are adjusted using a number of ISODATA-like operations - INITIALIZE, DIVIDE, LUMP, CREATE, DROP, DISTRIBUTE, RETRIEVE, UPDATE - that alter (sets of) clusters in the node with the aim of improving the split from the decision tree point of view. The adjustment of clusters by application of the ISOETRP operations is done interactively by an operator viewing dynamically updated overlap tables for the splits being adjusted. The end result is a crisp clustering of the cases in the node that determines the number of children nodes as well as the children in them. The issue of cluster validity is solved here by the operator, who simply picks the best looking result by viewing the visually displayed overlap tables.

Suen and Wang compare ISOETRP *as a clustering algorithm* to both HCM and ISODATA on some fairly small 4D data sets derived from noisy handwritten Chinese characters. In a refreshingly candid summary, they concede that HCM and ISODATA are both faster, and both do better at minimizing J_1 than ISOETRP. But, they argue that this is to be expected, since ISOETRP has a different objective - viz., the construction of a good classifier tree. They also reported trying various hierarchical algorithms such as single linkage to crisply partition the cases passing through the nodes, and state that this approach met with little success.

The papers by Wang and Suen (1983, 1984, 1987) begin with the assumption that a crisp classifier tree for continuously valued numerical feature data has been built by whatever means (they use ISOETRP of course, but C4.5 or CART would do). Then they introduce internal node decision functions that attempt to approximate Bayesian decision functions for the clustered regions in each internal node. The diagonal norm is used to create statistically meaningful elliptical regions in the feature space to measure distances between the input datum and within node cluster centers.

Specifically, after training each internal node contains one or more crisp subsets of labeled samples. Suppose c classes are represented at internal node k. Compute the subsample means $\{\bar{\mathbf{v}}_{k,1},\ldots,\bar{\mathbf{v}}_{k,c}\}$, where $\bar{\mathbf{v}}_{k,i}$ is the mean of cases (vectors) labeled class i in node k. For an input $\mathbf{z}$, Wang and Suen compute the c *diagonal norm* distances (see (1.8)), $\left\{\delta_{k,i} = \left\|\mathbf{z} - \bar{\mathbf{v}}_{k,i}\right\|_{D_k^{-1}}\right\}$, and then order them in ascending rank, $\delta_{(k,1)} \le \delta_{(k,2)} \le \ldots \le \delta_{(k,c)}$. Then Wang and Suen define the node decision function as

$$\phi_{ik}(\mathbf{z}) = \begin{cases} \min\left\{1, \dfrac{\delta^2_{(k,2)} - \delta^2_{(k,1)}}{\kappa}\right\} & ; i = 1 \\ \max\left\{0, \left[0.5 - \left(\dfrac{\delta^2_{(k,i)} - \delta^2_{(k,1)}}{\kappa}\right)\right]\right\} & ; i > 1 \end{cases} , \tag{4.78}$$

where κ is a user-defined parameter. These are the fuzzy decision functions Wang and Suen use in (4.76), and like Chang and Pavlidis, they may obtain a firing strength (again called a fuzzy decision value) for every path in the tree. At this point, however, Wang and Suen depart from the strategy shown at (4.77). Instead, they regard the firing strengths as heuristic evaluations that can aid in finding, but possibly not point to, the final label assigned to $\mathbf{z}$.

Choosing a threshold τ, they accumulate all the leaves, say L_τ, with firing strengths greater than τ by conducting a depth first search which abandons paths in subtrees rooted at internal nodes if the fuzzy decision value along that edge is less than τ. Then a global training algorithm prunes the tree by creating a set of "extended" leaves by considering, for each leaf in L_τ, only its immediately "adjacent" terminals (see Wang and Suen, 1987, for specification of the adjacency criterion). At the end of pruning, the extended leaves all have firing strengths above the threshold, and each is equipped with a probabilistic measure of similarity between $\mathbf{z}$ and the mean vector $\bar{\mathbf{v}}_{L_i}$ of the samples in it (assumed pure) that gauges the relevance of each leaf to a given input.

In the recognition mode, top down search (which might miss the solution of (4.77)) finds the maximal firing strength for input $\mathbf{z}$. If the probabilistic similarity of $\mathbf{z}$ to the leaf found is greater than a second threshold γ, the crisp label of that leaf is assigned to $\mathbf{z}$. Otherwise, they commission a heuristic search in the extended leaves to find a terminal that does satisfy $\alpha_i(\mathbf{z}) \geq \gamma$, and if one can be found, they use the crisp label residing there. Such a terminal might satisfy (4.77), or it might not, but Wang and Suen argue that the label of any leaf such that $\alpha_i(\mathbf{z}) \geq \gamma$ is a good decision because it is (i) related to the Bayes classifier through (4.78), and (ii) the tree has been pruned with the fuzzy decision values.

Wang and Suen (1987) give results of applying their fuzzy decision tree classifier to three sets of noisy Chinese characters having c = 64, 450 or 3,200 classes, 15 samples per class. They derive 64 features for each datum, and trained the trees for each case using 2/3 of the data for training, and the remaining 1/3 for testing. In the experiment with 3,200 classes, the average *level* of terminals was 5,415. By their analysis, the tree building phase, using the interactive clustering algorithm ISOETRP, is $O(c\log_2 c)$. They estimate that the pruning phase they call global training takes about 1/10 of this time. Time consuming, but in their view, worth it. Their best result on the 3200 class problem is an error rate of 0.07% - that is, they miss 10 or 11 characters in 16,000 test cases.

Maher and St. Clair (1992) inject fuzzy sets into the ID3 framework, and then generalize the inference procedure of Chang and Pavlidis in equation (4.76). They assume continuously valued real inputs, fuzzify each input datum in both the training and test sets, and use this alteration of the data to create interval valued decision functions. Their algorithm, called UR-ID3, thus builds a new type of fuzzy decision tree, since it creates a support *interval* for each possible classification of any test sample.

UR-ID3 first constructs a fully expanded crisp ID3 tree which contains crisp decision functions at its internal nodes. This construction is based on real-valued data, but quantization of each input datum using cutpoints (like C4.5) is not done. Thus, the ID3 tree will not be able to classify any non-training input. To accommodate generalization, each point in the training data is then spread across each of its feature values by determining a support interval for the similarity of its value to each of the other $n_{tr}-1$ feature values of the same coordinate in the data. Support intervals are computed with possibility theory using triangular membership functions centered at each feature value pair.

The result of softening the numerical features is to replace each edge weight in the ID3 tree, which is either 0 or 1, with an *interval* of the form $[ns_{ik}, ps_{ik}] \subseteq [0,1]$, ns_{ik}, ps_{ik} being, respectively, the necessary and possible supports of a feature at node k for class i. In other words, the node decision function $\phi_k : X \mapsto [0,1]^p - \{\mathbf{0}\} = N_{pc}$ is replaced by an interval-valued function, $\phi_k : X \mapsto \mathcal{P}([0,1]^p)$, where $\mathcal{P}([0,1]^p)$ is the set of all p-tuples of subintervals of $[0,1]$. Thus, edge weights in the Chang-Pavlidis model are replaced by intervals.

When an input datum traverses the tree to its leaves, the result will be a "firing strength interval", which is constructed by taking intersections of path intervals. The interval arithmetic operations used are

$$[a_1, b_1] \wedge [a_2, b_2] = [a_1a_2, b_1b_2] \quad \text{, and} \qquad (4.79a)$$
$$[a_1, b_1] \vee [a_2, b_2] = [a_1 + a_2 - a_1a_2, b_1 + b_2 - b_1b_2] \quad . \qquad (4.79b)$$

The application of (4.79a) along a path results in an interval, say $[\alpha_i^{ns}(\mathbf{z}), \alpha_i^{ps}(\mathbf{z})]$ at leaf v_{L_i}. Since each leaf is pure, it contains, say, n_i crisply labeled samples from one of the c classes in the training data. The relative frequency of samples in leaf v_{L_i} is used to normalize the firing strength interval by multiplying each endpoint of the interval with the fraction n_i/n, so leaf v_{L_i} is now associated with the interval

$$I_i^{\alpha} = \left[\left(\frac{n_i\alpha_i^{ns}(\mathbf{z})}{n}\right), \left(\frac{n_i\alpha_i^{ps}(\mathbf{z})}{n}\right)\right] \quad . \qquad (4.80)$$

Maher and St. Clair then collect all the leaves in the tree that have crisp label $\mathbf{e}_j$, j = 1,..., c, and aggregate the support intervals for label j into one overall support interval $I_j^{\vee\wedge}$ for the terminal block associated with class j. This is done by applying (4.79b) to all the intervals of form (4.80) that support each class. Figure 4.41 pictorially illustrates the soft decision tree of Maher and St. Clair.

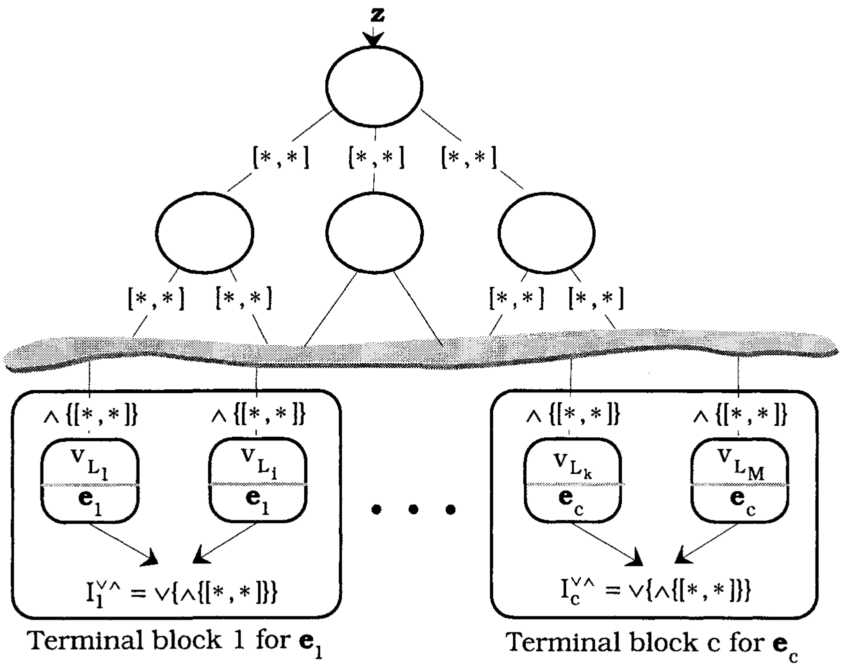

z
[*,*]
[*,*]
[*,*]
[*,*]
[*,*]
[*,*]
[*,*]
∧{[*,*]}
V_{L_1}
e_1
V_{L_i}
e_1
$I_1^{\vee\wedge} = \vee\{\wedge\{[*,*]\}\}$
Terminal block 1 for e_1
V_{L_k}
e_c
V_{L_M}
e_c
$I_c^{\vee\wedge} = \vee\{\wedge\{[*,*]\}\}$
Terminal block c for e_c

Figure 4.41 The soft decision tree of Maher and St. Clair (1992)

At the end of the training step, each of the c classes is represented by one support interval of the form $I_j^{\vee\wedge} = \vee\{\wedge\{[*,*]\}\}$, j = 1,...,c, as shown in Figure 4.41. Now the tree is ready for operation. Input datum **z** passes through the tree, and arrives at its bottom supported by (possibly) c different firing strength intervals $\{I_j^{\vee\wedge}\}$. Of the many possible ways to extract a final label, Maher and St. Clair opt for the most conservative choice, by assigning **z** the label of the terminal block associated with the support interval having the *largest* necessity value for its left endpoint. Three sets of data are used by Maher and St. Clair (1992) to illustrate UR-ID3. Here is an adaptation of their presentation of classifier design with the Iris (?) data.

Example 4.14 Maher and St. Clair (1992) compare four classifier designs using 75% of the Iris data for training and the other 25% for testing. They repeated this for three different sets of randomly drawn test and training data. UR-ID3 was compared to the standard ID3 tree, a 1-nn variation of ID3 due to St. Clair et al. (1992), and a standard *feed-forward back-propagation* (FFBP, Section 4.7) neural network. The 1-nn variant of ID3 differed from ID3 only during testing; in this phase of operation, if a path in the ID3 tree did not exist, then the "nearest neighbor" path in the tree was taken. Table 4.34 repeats the test results of their experiments as percent correct on the test sets.

Table 4.34 Percent correct classification on 3 Iris test sets of 25 points each with four classifiers (Maher and St. Clair, 1992)

	ID3	ID3-nn	UR-ID3	FFBP
Iris 1	75.7	89.2	94.6	91.9
Iris 2	71.1	92.1	94.7	94.7
Iris 3	78.9	94.7	94.7	92.1
Ave.	*75.2*	*92.0*	*94.7*	*92.9*

The average number of internal nodes for ID3 was 5, and the average number of leaves (or crisp rules developed on 112 labeled data) was 45. Since UR-ID3 and ID3-nn use the same trees, these statistics are valid for all three decision tree designs. This agrees with the general belief that if nothing else, decision trees get big - fast.

The last row of Table 4.34 indicates that, for these trials, the fuzzy interval-based decision tree classifier was much better than ID3, and it was slightly better than the crisp ID3-nn approximation. According to these statistics UR-ID3 was also slightly better than the FFBP classifier network they used in this comparison.

We add three remarks about these results. First, the values displayed in Maher and St. Clair for illustration of interval building with an input datum from Iris lead us to believe that they actually processed an integer valued data set that might be Iris with every value multiplied by 10 (see "will the real Iris data please stand up" in the preface). Second, it is pretty easy to train a feedforward network to be consistently achieve 100% success with various data selection schemes when applied to (the) Iris (we use). We illustrate this in Example 4.21.

Finally, crisp decision trees built with C4.5 on Iris are slightly better than any of the decision trees illustrated in Table 4.34. For example, Hall et al. (1998) report that release 8 of C4.5 run with the default parameters builds crisp decision trees on Iris that achieve an average error rate of 4.7% - that is, 95.3% correct classification - when trained and tested by 10-fold cross validation. This scheme uses 90% percent of the Iris (?) data for training (135 samples) and the remaining 10% (15 samples) for testing in each of 10 cycles, rotating through the entire data set so that the union of the 10 test sets is Iris, and their pairwise intersections are empty. This is a somewhat more pessimistic error rate estimate than the 75/25 split used by Maher and St. Clair because individual tests are closer to the leave one out method, and averaging the error rate over 10 trials produces a better estimate. The *average* tree size over 10 runs in Hall et al. (1998) was 5.3 nodes (leaves and internal nodes). Thus, the C4.5

crisp tree size is an order of magnitude smaller than the trees built by Maher and St. Clair's fuzzy decision tree methods.

Umano et al. (1994) present a fuzzy extension of ID3 that can deal with both real and categorically-valued attributes. Their scheme, like that of Maher and St. Clair, uses the basic ID3 algorithm to build a tree, and then they extend its crisp decision functions at internal nodes so that each training datum is captured by a larger domain. Rather than cover each point with a possibly different interval, they impose a set of discrete, user-defined premise membership functions on each input variable.

Umano et al. assume that the input data have c classes, but that each class is fuzzy. This is represented by attaching what is in essence a user-defined possibilistic c-partition $U(X) \in M_{pcn}$ of $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ to the input. Umano et al. use the fuzzy cardinalities of X computed on the entries of U(X) to replace the relative frequencies used in ID3, and the ID3 node splitting function is converted into one that attempts to maximize information gain based on probabilities of membership values. These authors present an example that is very much like Example 4.13. To impart the flavor of their method without filling several pages with fairly routine details, we abstract it here as our Example 4.15.

Example 4.15 Umano et al. (1994) illustrate their fuzzy ID3 method on the following set of data (we have reordered it for clarity),

$$X = \left\{ \begin{pmatrix} 160 \\ 60 \\ \text{blond} \end{pmatrix}, \begin{pmatrix} 175 \\ 60 \\ \text{red} \end{pmatrix}, \begin{pmatrix} 180 \\ 70 \\ \text{blond} \end{pmatrix}, \begin{pmatrix} 180 \\ 80 \\ \text{black} \end{pmatrix}, \begin{pmatrix} 170 \\ 75 \\ \text{black} \end{pmatrix}, \begin{pmatrix} 160 \\ 75 \\ \text{black} \end{pmatrix}, \begin{pmatrix} 175 \\ 60 \\ \text{red} \end{pmatrix}, \begin{pmatrix} 165 \\ 60 \\ \text{blond} \end{pmatrix} \right\}.$$

Point	①	②	③	❹	❺	❻	❼	❽
Class	1	1	1	2	2	2	2	2
Memb.	1.0	0.7	0.5	0.8	0.2	1.0	0.3	1.0

The first two components of each data vector are the p = 2 numerical features height and weight of 8 objects (presumably humans), while the third component is the variable "hair color", with q = 3 values: blond, black and red. Directly beneath the data are the crisp class labels attached to the 8 points by the authors, and directly below the crisp labels is another value associated with these 8 data, which is a subjectively defined set of fuzzy memberships. The authors are not clear about the source or meaning of these memberships, so we interpret them as a measure of confidence in the crisp label assigned, and represent them as a possibilistic 2-partition of X,

$$U(X) = \begin{bmatrix} 1.0 & 0.7 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.8 & 0.2 & 1.0 & 0.3 & 1.0 \end{bmatrix}.$$

By our interpretation, the first datum definitely belongs to class 1 and not at all to class 2, the second belongs to class 1 to the extent 0.7 and not at all to class 2, and so on. Umano et al. don't describe the fuzzification of the input data quite this way. They simply identify the first 3 points as "being in" class 1, and the last 5 points as having a class 2 label. In their paper the non-zero values we show in the matrix U are simply called membership grades given to the 8 examples. This is an example where each datum comes with a crisp label, and other information is used to augment the original label structure of the problem. In effect, each point in the training data has both a crisp and possibilistic label.

Compare the first and last vectors in X to see that the two classes are pretty mixed, since datum 1 is, by its memberships in U, definitely in class 1, while datum 8 is definitely in class 2, but the only difference between these two objects is in the first feature, 5 (cms ?) in height. This is even more pronounced in points 2 and 7, which have identical representations but, according to U, object 2 prefers class 1, while object 7 has a small amount of membership in only class 2.

The authors then *define* three sets of *discrete* premise membership functions over the three input variables. As a first example of the notation we use for fuzzy systems, we list each of these PMFs as a set of ordered pairs in the general form $(x_i, m_{ij}(x_i))$:

PMFs $\{m_{1j}(x_1)\}$ for height:

ℓ_{11} = low	m_{11} = {(160, 1), (165, 0.8), (170, 0.5), (175, 0.2)}
ℓ_{12} = middle	m_{12} = {(165, 0.5), (170, 1.0), (175, 0.5)}
ℓ_{13} = high	m_{13} = {(165, 0.2), (170, 0.5), (175, 0.8), (180, 1.0)}

PMFs $\{m_{2j}(x_2)\}$ for weight:

ℓ_{21} = light	m_{21} = {(60, 1), (65, 0.8), (70, 0.5), (75, 0.2)}
ℓ_{22} = middle	m_{22} = {(65, 0.5), (70, 1.0), (75, 0.5)}
ℓ_{23} = heavy;	m_{23} = {(65, 0.2), (70, 0.5), (75, 0.8), (80, 1.0)}

PMFs $\{m_{3j}(x_3)\}$ for hair color

ℓ_{31} = light	m_{31} = {(blond, 1.0), (red, 0.3)}
ℓ_{32} = dark	m_{32} = {(red, 0.6), (black, 1.0)}

Notice that our parameter r, the granularity of the sets of PMFs in Figure 4.32, is variable here: $r_1 = r_2 = 3$, $r_3 = 2$. Also notice that while the PMFs shown in Figure 4.32 are continuous, these authors use discrete PMFs (but they do not limit their version of fuzzy ID3 to this). Subsequent calculations using Umano et al.'s node splitting functions and several additional heuristics lead to the fuzzy decision tree shown in Figure 4.42, which is our adaptation of Umano et al.'s Figure 1.

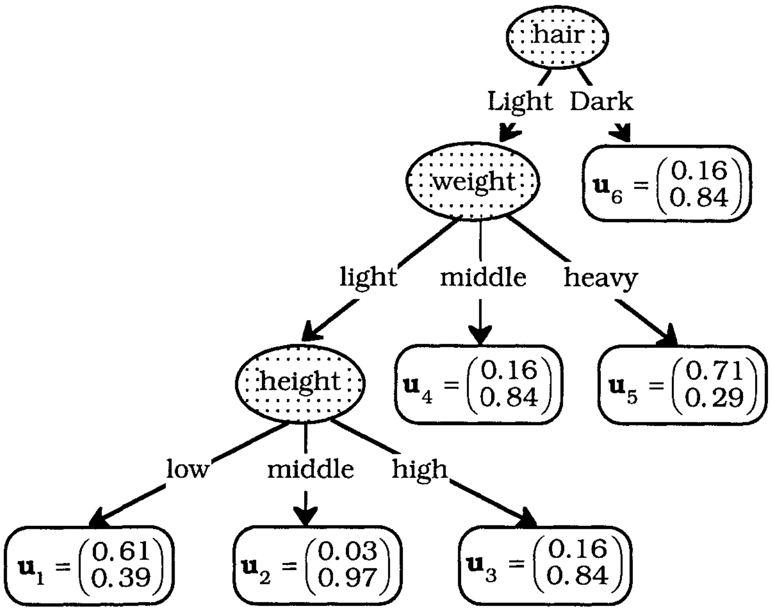

hair
Light
Dark
weight
$\mathbf{u}_6 = \begin{pmatrix} 0.16 \\ 0.84 \end{pmatrix}$
light
middle
heavy
height
$\mathbf{u}_4 = \begin{pmatrix} 0.16 \\ 0.84 \end{pmatrix}$
$\mathbf{u}_5 = \begin{pmatrix} 0.71 \\ 0.29 \end{pmatrix}$
low
middle
high
$\mathbf{u}_1 = \begin{pmatrix} 0.61 \\ 0.39 \end{pmatrix}$
$\mathbf{u}_2 = \begin{pmatrix} 0.03 \\ 0.97 \end{pmatrix}$
$\mathbf{u}_3 = \begin{pmatrix} 0.16 \\ 0.84 \end{pmatrix}$

Figure 4.42 Umano et al.'s fuzzy decision tree for the data set X

This tree has 3 internal nodes and the training data are used to produce fuzzy label vectors at the 6 leaves; $\mathbf{u}_k \in N_{fc}$ is attached to leaf v_{L_k} for k = 1 to 6. Compare this to the CP tree in Figure 4.37, where each leaf contains a path firing strength and crisp label. Umano's tree is equivalent to the rule-base $\mathcal{R} = \{R_1, \ldots R_6\}$ whose i-th rule, $1 \le i \le 6$, has the form

$$\alpha_i(\mathbf{x}) \Rightarrow \mathbf{D}^{U}_{DT}(\mathbf{x}) = \mathbf{u}_i \quad . \tag{4.81}$$

In (4.81) the LHS has 3 premise clauses, but some of the rules have less than three. When an input datum is submitted to this tree, its values may partially match all 6 of the fuzzy rules (that is, may arrive at all 6 leaves in the tree in Figure 4.42). The firing strength along each path is computed with the left side of (4.81) using the product for the T-norm, $T = T_2$. Each edge in the Umano et al. tree has a fuzzy label vector attached to it (this is not shown in Figure 4.42), which stands in sharp contrast to the edge weights in the CP tree (numbers in [0, 1]), and edge intervals in UR-ID3. Umano et. al also apply the product to each component of the fuzzy label vectors along the edges. And finally, aggregation of the evidence developed at each leaf for the input datum is done with addition, which can lead to certainty values greater than 1. Umano et al. say that when this happens, just normalize them. They call the overall inferencing method $(\times\times+)$. The output of Umano et al.' s fuzzy decision tree is a fuzzy label vector for each input, so this design is a fuzzy classifier in the sense used by us - that is, $\mathcal{R}(\mathbf{z}) = \mathbf{D}^{\mathrm{U}}_{\mathrm{DT}}(\mathbf{z}) = \mathbf{u} \in N_{fc}$. If desired, this output can be hardened in the usual way.

Finally, Umano et al. give a numerical example using n = 220 samples of transformer data which have two labeled classes of causes of failure, which are themselves subdivided into 4 and 17 subclasses. Half of the data were used to train the fuzzy decision tree, and the other half were used to test it. They give some error rate statistics for their tests, but since this method is not compared to any other method, it's hard to guess what the statistics tell us about the method. But we like this as an example of generalization of both the fuzzy CP tree, as well as crisp ID3.

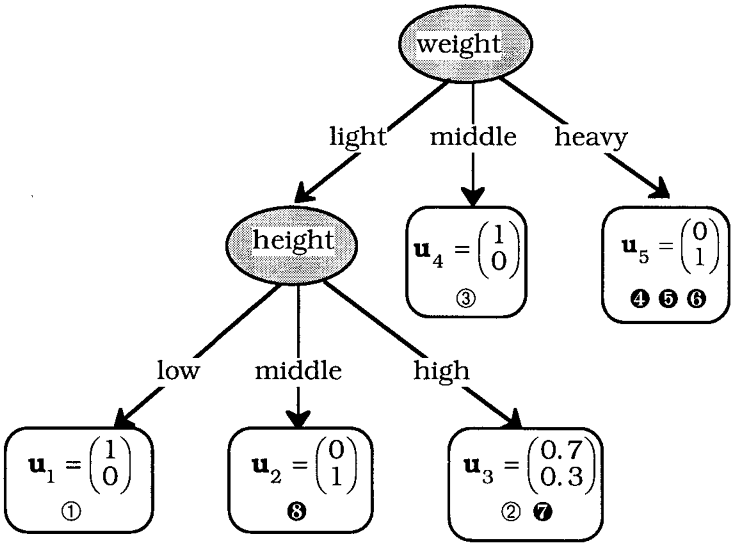

weight
light
middle
heavy
height
$\mathbf{u}_4 = \begin{pmatrix}1\\0\end{pmatrix}$
③
$\mathbf{u}_5 = \begin{pmatrix}0\\1\end{pmatrix}$
❹ ❺ ❻
low
middle
high
$\mathbf{u}_1 = \begin{pmatrix}1\\0\end{pmatrix}$
①
$\mathbf{u}_2 = \begin{pmatrix}0\\1\end{pmatrix}$
❽
$\mathbf{u}_3 = \begin{pmatrix}0.7\\0.3\end{pmatrix}$
② ❼

Figure 4.43 Zeidler et al.'s fuzzy decision tree for Umano's et al.'s data set X in Example 4.15

Zeidler et al. (1996) discuss an interesting modification of the fuzzy ID3 approach of Umano et al. (1994) that seems to extend its utility in that some of the subjectivity in Umano et al.'s design is removed.

These authors give an algorithm for automatic generation of continuous premise membership functions that span each numerical input variable (recall that the user simply defined discrete premise membership functions in Umano et al.). The PMFs are all trapezoidal, and are adjusted dynamically during the construction of the tree. Zeidler et al. process the data shown as X in Example 4.15 with their algorithm, and obtain the decision tree shown in Figure 4.43, which is our adaptation of their Figure 3.

Compare Figures 4.42 and 4.43 - there are some striking differences. Umano et al.'s tree is rooted on the linguistic variable "hair color" and has 6 leaves, all associated with rather fuzzy labels. Zeidler et al.'s tree doesn't even use hair color, is rooted on the numerical variable "weight", has only 5 terminals, and 4 of the 5 terminals are associated with crisp labels - that is, they are pure leaves. The two objects labeled 2 and 7 in the original data end up in the only leaf that doesn't have a crisp label. Recall that these two objects had identical features, but different class labels. We think that Zeidler et al.'s approach produces a clearer picture of the structure of the data than Umano et al.'s. Unfortunately, Zeidler et al. did not try this method on any real data set, so it is even more difficult to make any assessment of its relative utility than the classifier tree of Umano et al. These authors do give a very clear example of processing an unlabeled input vector **z** with their tree:

$$\boldsymbol{\mathcal{R}}(\mathbf{z}) = \boldsymbol{\mathcal{R}}\begin{pmatrix} 62 \\ 162 \\ \text{red} \end{pmatrix} = \mathbf{D}^{z}_{\text{DT}}\begin{pmatrix} 62 \\ 162 \\ \text{red} \end{pmatrix} = \begin{pmatrix} 0.68 \\ 0.32 \end{pmatrix} \in N_{fc}\,.$$

The last method we discuss in this subsection is due to Janikow (1996a, 1998). Janikow fuzzifies both the construction and inferencing procedures for decision trees. His model has many of the same elements as the fuzzy systems shown in Figure 4.32, although he prefers to regard the fuzzy rules aspect of his decision trees as an artifact, rather than the reason for the trees. Janikow gives a nice, clear discussion of most of the previous work on fuzzy decision trees, and their relationship to fuzzy systems. He uses the methodology of ID3 as a template for his fuzzy tree building algorithm, which, in his words, "is the same as that of ID3. The only difference is based on the fact that a training example can be found in a node to any degree."

Janikow's (1998) fuzzy ID3 is not a complicated algorithm, and while he illustrates it only with numerical data, it is equally applicable to nominal data. The node splitting function is formally an entropy function, but the arguments of ι_{ent} depend explicitly on the PMFs of the linguistic variables chosen to fuzzify the input domains. The central idea is that memberships $\{m_{ij}(x_k)\}$ of the attribute values that occur along paths from the root to the current node play an active role in the determination of which cases arrive at a node, and how much each should be weighed in the split. Values of $\{T(\{m_{ij}(x_k)\})\}$ accumulate as incremental firing strengths along each path, using a T-norm of choice, and these contribute to the overall case count at the current node. At termination the leaves of the tree may not all be pure, and further, the same case may occur with partial membership in more than one leaf. These terminal memberships are possibilities (they don't have to sum to 1).

Janikow (1998) points out that once the tree is built, there are any number of possible choices for inferencing with it, some of which are interpolative (if the data are numerical); and some of which are not (necessary if the data are nominal). When operating as a classifier, all the leaves with paths of positive firing strength can be found, and these consequents can be aggregated using a T-conorm and then defuzzified, or simply combined using a weighted mean. Janikow discusses four methods of inferencing based on the weighted fuzzy mean or simplified max-gravity method (Mizumoto, 1988). Two of them use information about the most common label in terminal blocks, and the other two try to account for within-leaf label inconsistencies. Janikow also discusses four reasoning procedures based on finding a dominant leaf with the center of, sum of and maximum gravities defuzzification strategies. Then he gives the numerical example repeated here as our Example 4.16.

Example 4.16 Janikow (1998) illustrates his fuzzy ID3 method on the following set of data, which is strikingly similar to the one used in Example 4.15 (and not just because, like Quinlan and Umano et al., n = 8).

$$X=\left\{\begin{pmatrix}0.20\\0.15\end{pmatrix},\begin{pmatrix}0.30\\0.25\end{pmatrix},\begin{pmatrix}0.90\\0.20\end{pmatrix},\begin{pmatrix}0.60\\0.50\end{pmatrix},\begin{pmatrix}0.90\\0.50\end{pmatrix},\begin{pmatrix}0.10\\0.85\end{pmatrix},\begin{pmatrix}0.40\\0.90\end{pmatrix},\begin{pmatrix}0.85\\0.85\end{pmatrix}\right\}.$$

Point	①	②	③	④	❺	❻	❼	❽
Class	1	1	1	1	2	2	2	2
Weight	1	1	1	1	1	1	1	1

Figure 4.44 is a scatterplot of $X \subset \Re^2$. The point **z** shown in Figure 4.44 is not one of the training data - it's a test input that we will classify with the fuzzy decision tree after it has been built. Janikow imagines that the classes represented in the data are related to decisions a lender must make about borrowers : class 1 = not creditworthy, and class 2 = creditworthy. In our standard notation these two classes would be represented by the crisp label vectors $\mathbf{e}_1=(1,0)^T$ and $\mathbf{e}_2=(0,1)^T$.

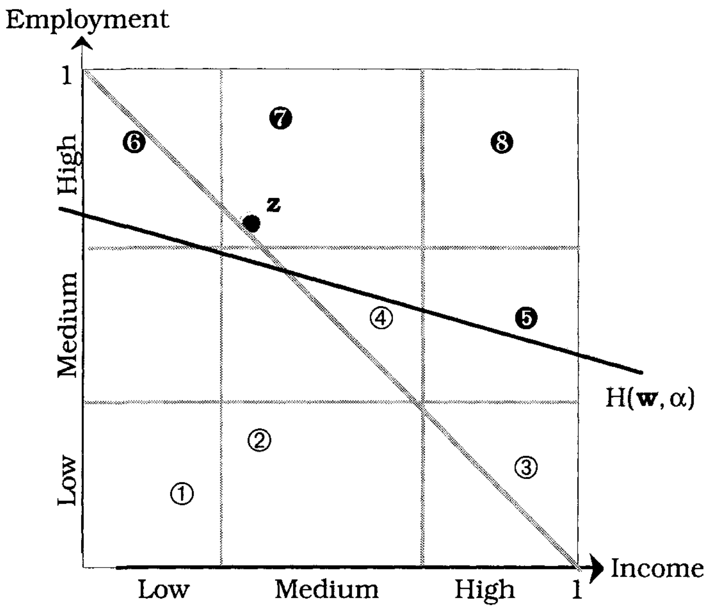


Figure 4.44 Janikow's data and premise membership functions

Since this is a c = 2 class problem, Janikow arranges his decision system outputs so that they are numbers in [0, 1] instead of label vectors, so we write this classifier function as $D^J_{DT}:\Re^2 \mapsto [0,1]$. Janikow uses the labels 0 = not creditworthy and 1 = creditworthy for the two classes, and regards fuzzy outputs of his system as numbers between 0 and 1 (instead of fuzzy label vectors in N_{f2}). Since there are only two classes, hardening a fuzzy output corresponds to using 0.5 as a threshold on the output of the system. For example, 0.47 is hardened to yield 0 = class 1 (non-creditworthy), and 0.64 is converted to the class label 1 = class 2 (creditworthy).

The simplest way to classify anyone on this basis would be to plot their coordinates and see which side of the hyperplane through the corners (0,1) and (1,0) the datum fell on: above would presumably correspond to an acceptable risk, and below, to a person not to be trusted to repay a loan. The data shown are not linearly separable by this hyperplane, which would commit three training errors. There are separating hyperplanes, however, such as H(**w**, α) shown in Figure 4.44, which will produce no errors on resubstitution. Consequently, from the point of view of classifier design, one of the things we will want to know is whether a decision tree approach offers more than this simple solution, which can be found by eye.

Examination of Figure 4.44 tells us - without computation - that horizontal splits (along the employment axis) will be more effective at the root of any tree covering these 8 cases than vertical splits along the income axis. Cases 1-3 and 6-8 can be isolated from 4 and 5 with just two cutpoints along the employment axis, and the eight training data can be easily covered with 4 crisp rule patches that yield no training errors. But we know that such a classifier will not generalize well.

Janikow defines termsets of three linguistic values, {low, medium, high}, for each of the linguistic variables income and employment. Figure 4.45 shows the general form of these functions for the first linguistic variable (income), which are limited in Janikow (1998) to trapezoidal fuzzy numbers. The same functions are used for the variable employment.

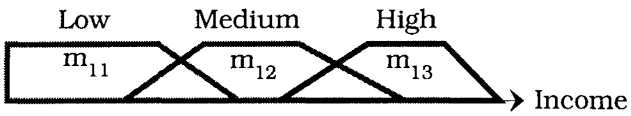


Figure 4.45 Janikow's premise membership functions

Janikow leads the reader through sample calculations for all the functions used during node splitting in his fuzzy ID3 tree building algorithm, and arrives at the final tree shown in Figure 4.46.

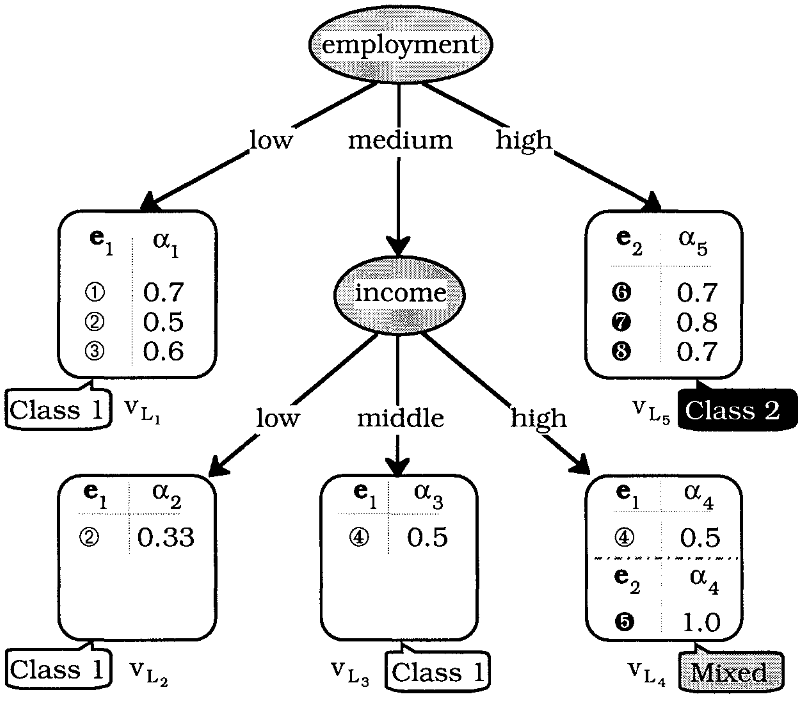


Figure 4.46 Complete decision tree for Janikow's data

As expected from the geometry of the features for the training cases seen in Figure 4.44, the terminal tree is rooted in employment, with cases 1-3 and 6-8 immediately splitting from the root to terminal nodes v_{L_1} and v_{L_5}. The second linguistic variable is used to split the remaining cases, and although there are (presumably) only 2 cases left, notice that Janikow's method also pushes case 2 into a second terminal leaf, v_{L_2}. Case 4 acquires its own terminal leaf v_{L_3} and also moves into v_{L_4}, which it shares with case 5. Also shown in Figure 4.46 are the firing strengths along the paths from the root to the leaves. This tree corresponds to a 5 rule fuzzy system, but note that rule 4 has two possible consequents, since the cases are mixed. And conversely, object 2, which has a crisp case 1 label, arrives at v_{L_1} with a firing strength of 0.5, and at v_{L_2} with a firing strength of 0.33. In other words, rules 1 and 2 in the fuzzy system represented by this tree both support a match to training data point 2, but with different levels of confidence, whereas rule 4 supports a match to several outcomes, the strength depending on the matched label. Similarly, object 4 is also labeled class 1, with equal firing strengths of 0.5 in 2 different leaves.

Janikow's avowed purpose is to focus on decision trees, not fuzzy rules, so he spends little time distinguishing MA and TS type rules that might be equivalent to this tree. Janikow does talk about using the firing strengths that arrive at terminal nodes in conjunction with defuzzification to make subsequent classifications. So, we assume that each of the possible consequents (class 1 = too risky, class 2 = creditworthy) has a fuzzy set associated with it.

Janikow shows how the classifier represented by the tree in Figure 4.46 operates using the center of gravity method of inferencing on six new test data. Since the input space is $[0,1] \times [0,1]$, the rule-base will always have an output in $[0,1]$ with the defuzzification being used, we expect that $D^J_{DT}(\mathbf{0}) = 0, D^J_{DT}(\mathbf{1}) = 1$. And indeed, Janikow shows how the input vector $\mathbf{y} = (0,0)^T$ causes the response $\boldsymbol{\alpha}(\mathbf{y}) = (1,0,0,0,0)^T$, where $\boldsymbol{\alpha}(\mathbf{y})$ is the (ordered) set of firing strengths of the paths leading to the 5 terminal nodes in Figure 4.46. Only rule 1 is fired for this input, and this input will be unequivocally labeled class 1 (too risky). This certainly agrees with the location of this datum in the feature space. In words : "IF employment is low and income is low THEN no credit". What would our simple hyperplane $H(\mathbf{w},\alpha)$ shown in Figure 4.44 do for this input? The same.

The test input $\mathbf{z} = (0.32,0.70)^T$ plotted on Figure 4.44 results in the set of firing strengths $\boldsymbol{\alpha}(\mathbf{z}) = (0,0,0.3,0.67,0.40)^T$. Now three of the five rules have positive support, and it is necessary to combine them with some form of disjunctive aggregation. Janikow, using the center of gravity defuzzification, arrives at an overall value of 0.71 for this input, that is, $D^J_{DT}(\mathbf{z}) = 0.71$. Recall that hardening here corresponds to rounding off, so 0.71 corresponds to the label 1 = creditworthy, that is, $\mathbf{H}(D^J_{DT}(\mathbf{z})) = e_2 = 1$, so we will happily allow $\mathbf{z}$ to go into debt. Our hyperplane H(**w**, α) would too.

Janikow (1998) goes on to process three input data with missing values, the inputs (unk, 0.75), (0.5, unk), and (unk, unk), where unk = "unknown". The test data used do not illustrate the efficacy of this tree as a classifier, however, since none of them has a crisp label. Now the hyperplane fails, but Janikow's tree produces the outputs 0.63, 0.59 and 0.51, respectively for these three points - that is, upon hardening (rounding off to 1 = class 2), all three of these inputs represent people that will be granted credit.

The last input point is particularly interesting; the defuzzified output value is not exactly 0.50, even though the input datum (unk, unk) would suggest a coin flip to make the ruling in this case, since nothing is known about the input and the sample priors are both 0.5. Janikow says the value 0.51 occurs because the case counts in the leaves is different from those in the root. Thus, the root starts with 4 examples of each class, but the leaves contain 3.13 in-leaf cases for class 1, and 3.20 cases for class 2 (these counts are the sums of the firing strengths in the leaves), so the training method imparts a slight bias towards class 2. Tuning the CMFs and PMFs might be used to balance the in-leaf counts so that they matched the root priors to solve this problem, but Janikow does not mention doing this in his 1998 paper. See Janikow (1996b) for a discussion of optimizing the initial tree found by this method. As an aside, we remark that this seems to be the model used by many (at least American) bankers, who cheerfully let anyone who wants to go into debt, with consequences following the truth of their situation - only later.

The last thing we mention is that Janikow (1998) does a creditable job of comparing the utility of his method to another scheme for the function approximation problem we introduced in subsection 4.6.D. Janikow builds a fuzzy ID3 tree using the same data that was used by Suh and Kim (1994) in connection with approximation of the Mexican hat function. Let $\mathbf{x} = (x,y)^T$, and consider the function

$$h_{mex}(\mathbf{x}) = \begin{cases} \dfrac{40\sin(\pi\sqrt{x^2+y^2}/35)}{\sqrt{x^2+y^2}/35} & ;\mathbf{x} \neq \mathbf{0} \\ 40\pi & ;\mathbf{x} = \mathbf{0} \end{cases} \quad . \tag{4.82}$$

Suh and Kim sampled h_{mex} over the domain $[-120, 120] \times [-120, 120]$ 13 times in each direction to obtain the training data X_{mex}. They then used the 169 IO triples $\{(\mathbf{x}_{ij}, h_{mex}(\mathbf{x}_{ij})\}$ to build a fuzzy membership function neural network to approximate h_{mex}. In brief, Suh and Kim manually generated 13 sets of fuzzy rules (one for each set of data along a line of constant y value on the sampling grid), partitioned each of the two input variables with 13 triangular premise membership functions and 7 consequent membership functions, trained the 13 networks, and then combined their outputs to produce approximations $\mathcal{R}(\mathbf{x};\hat{\theta}) \approx h_{mex}(\mathbf{x})$, where $\hat{\theta}$ represents the parameters of the networks acquired during training.

Janikow (1998) trains ID3 based trees on the same data, and shows the output of two trees on the training data and at test points in between them. The approximating rules (as represented by the fuzzy decision trees) differed only in the method of inferencing. Visual comparison of the surfaces recovered by Janikow's fuzzy decision trees and the neural network approximations appear to favor the neural network approach. Janikow (1998) seems to concede this by referring us to his (1996b) paper on optimizing the membership functions as a means of improving the approximation. In favor of his method - and we tend to agree with him - are the facts that his trees were not tailored to this particular problem, and the fuzzy ID3 rules were not generated manually.

We have one more fuzzy decision tree methodology to discuss (Chi and Yan, 1996, Chi et al., 1996), but we defer discussion of these papers to the section on classifier fusion, because these authors combine their version of fuzzy classifier trees with other techniques such as nearest prototype and Markov chain classifiers to (hopefully) improve the overall performance of either individual classifier.

G. Rule-based approximation based on clustering

Since a fuzzy decision tree is equivalent to a set of fuzzy rules, building a fuzzy decision tree amounts to extracting a set of fuzzy rules from numerical or linguistic data. Tree induction (and consequently, the rules a tree represents) from numerical data using algorithms such as ID3, C4.5 or CART does not depend primarily on structure in the data; rather, it depends most heavily on the relative frequency information that resides in the crisp labels of the data.

In this section we develop an alternate approach to rule extraction from numerical data that does just the opposite; it tries to focus on geometric properties of the data as captured by clustering algorithms. In a few cases we find the method of this section used directly for classifier design, but most of the important work in this area is aimed at approximating functions used in prediction and control. In any case rule extraction by clustering is a nice application of the material in Chapter 2 on clustering, now used as a tool in a very different context than its original domain. We begin with a discussion of the feasibility of approximating functions with fuzzy systems.

The Mexican hat example presented by Janikow (1998) that we discussed in subsection 4.6.F was our first example of using fuzzy rules to approximate functions. While Janikow's example shows the feasibility of using a fuzzy decision tree (and therefore, a fuzzy system) for function approximation, there can be problems with this approach. For example, computational complexity can be very high, and further, Janikow's results - the first we have seen for approximation by fuzzy decision trees - are visually inferior to those obtained by Suh and Kim (1994). The first question that comes to mind is - why should we expect a fuzzy rule-based system to do well at all? A theoretical answer to our question comes from the field called *universal approximation* (UA) theory.

We won't spend much time on this topic, because we do not explicitly rely on the results of UA theorems to design and construct a good classifier. But like many before us, we take some psychological reassurance from such theories, and as an old friend of ours once told us, "nothing is so practical as a good theory." Universal approximators are sets of functions $\{\boldsymbol{\mathcal{R}}(\mathbf{x};\boldsymbol{\theta}): X \subset \Re^p \mapsto \Re^q; \boldsymbol{\theta} \in \Omega\}$, where Ω is a parameter space for $\boldsymbol{\theta}$, that provide arbitrarily good approximations to every element in other sets of functions, say $\mathfrak{I} = \{\mathbf{f}: X \subset \Re^p \mapsto \Re^q\}$. The measure of goodness is a norm on $\Re^q$, typically X is compact, and every function in $\mathfrak{I}$ is continuous. The approximation to f is uniform by such families; i.e., once ε is given, for any $\mathbf{f} \in \mathfrak{I}$, you can find a set of parameters $\hat{\boldsymbol{\theta}}$ for which $\left\|\mathbf{f}(\mathbf{x}) - \boldsymbol{\mathcal{R}}(\mathbf{x};\hat{\boldsymbol{\theta}})\right\| < \varepsilon$ for *every* $\mathbf{x} \in X$. For example, Fourier series are a set of universal approximators for square integrable functions on $[0,2\pi]$.

There are any number of theorems guaranteeing that various MA and TS rule-based systems are universal approximators. The conditions on X and **f** vary, and there are usually other special conditions or constraints on the result that depend on the particular system you have in hand. This answers one question we raised in Example 4.17; in principle, a fuzzy system designed with *any* method - clustering included - *may* provide a good approximation to well enough behaved functions. Unfortunately, none of these UA theorems is constructive - that is, none of them tell us how to find the approximating system. That's why their value to the designers of a working pattern recognition system not high.

There are also many UA theorems for neural networks. Figure 4.47 depicts a 21 point IO data set that Narazaki and Ralescu (1993) obtained by uniformly sampling the function

$$S(x) = 0.2 + 0.8(x + 0.7\sin(x)), 0 \le x \le 1 \tag{4.83}$$

over the base points $X_{21} = \{0.00, 0.05, \ldots, 1.00\}$, which comprise a set of input training data, with corresponding output training data $Y_{21} = \{S(0.00),\ S(0.05),\ \ldots,\ S(1.00)\}$.

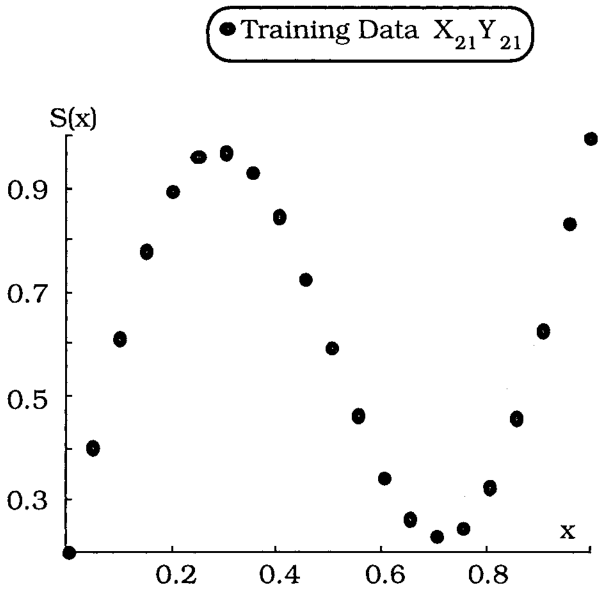


Figure 4.47 Data set $X_{21}Y_{21}$ is 21 samples from (4.83)

Narazaki and Ralescu used $X_{21}Y_{21}$ to illustrate the approximation capabilities of five different feed-forward neural network architectures. Approximation of S by the five schemes they describe yielded an *average* $E_{MSE}(X_{101}|X_{21}) = \left(\frac{100}{101}\right)\sum_{k=1}^{101}\left|y_k - s(x_k;\hat{\theta})\right| = 7.42\%$ on 101 test inputs uniformly distributed over [0,1]. The approximation capabilities of NNs are well known, so this is not surprising. Notice that these test data *include* the 21 training inputs, so this error is a little optimistic; nonetheless, this is a nice result. We will use these data to illustrate several rule extraction methods in Example 4.17.

To appreciate the relationship between smoothness and approximation, recall that the IO data available for identifying **S** are $XY = \left\{(\mathbf{x}_k, \mathbf{y}_k)^T : k = 1, \ldots, n\right\} \subset \Re^{p+q}$. Roughly speaking, XY is the "diagonal" of the Cartesian product X×Y. The discrete set XY is also assumed to be a subset of the *graph* G_S of **S**, which is in turn a subset of $\Re^p \times \Re^q$. Figure 4.48 shows these relationships.

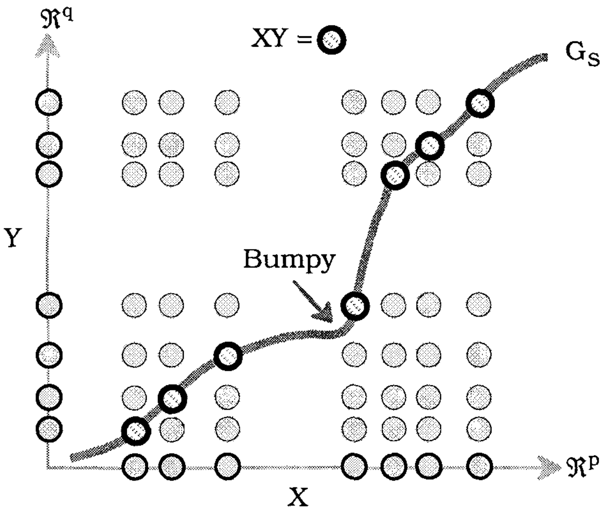


Figure 4.48 The sets X, Y, XY, X×Y and G_S

Rule extraction can be done by clustering in X, Y or XY, resulting in c-partitions $U^X, U^Y, \text{or } U^{XY}$, respectively. The superscript shows which of the three data sets is the basis of clustering. We assume that the clustering method also produces either point prototypes $\mathbf{V}^X = \left\{\mathbf{v}_i^X\right\}, \mathbf{V}^Y = \left\{\mathbf{v}_i^Y\right\}, \text{or } \mathbf{V}^{XY} = \left\{(\mathbf{v}_i^X, \mathbf{v}_i^Y)^T\right\}$; or non-point prototypes $\mathbf{B}^X = \left\{\mathbf{b}_i^X\right\})$, $\mathbf{B}^Y = \left\{\mathbf{b}_i^Y\right\}$ or $\mathbf{B}^{XY} = \left\{(\mathbf{b}_i^X, \mathbf{b}_i^Y)^T\right\}$. Many of the rule extraction methods depend on projections of these prototypes from XY to X and/or Y, and they also rely (almost always implicitly) on the smoothness of **S**.

When **S** is very smooth, as the sine curve in Figure 4.47 is, we will be able to find nice approximations to it with fairly course, low order fuzzy systems - in particular, with first order TS systems. When **S** is "bumpy" but still smooth (the graph of **S** in Figure 4.48 is like this at one spot), we will need a higher order approximation, more rules, finer premise membership function structure, and so on, to get decent approximations to the IO data.

Since we use rule-based systems in classifier design, it's nice to know these UA theorems exist. That's really all we need to say about this aspect of approximation to classifier functions by fuzzy systems, except for this very important point. Crisp classifier functions $\mathbf{D}: \Re^p \mapsto N_{hc}$ cannot be continuous, because their range is discrete, so UA theorems in this special case lose some of their appeal. However, be careful to distinguish what function you are approximating when you worry about this statement. In equation (4.2) we show the crisp classifier function $\mathbf{D}_{\mathbf{V},\mathbf{E},\delta}(\mathbf{z}) = \mathbf{e}_i$, which cannot be a continuous function of $\mathbf{z}$, but the function $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + \alpha$ that defines $H(\mathbf{w}, \alpha)$ is certainly smooth, and can be used to implement $\mathbf{D}_{\mathbf{V},\mathbf{E},\delta}$, so the situation for classifier design is not as bad as it might seem. When classifier functions are soft, UA theorems directly underlie our attempts to approximate them with rule-based systems.

If you want more information on universal approximation, start with Kreinovich et al. (1998). Just to give you a taste of what you will find there, we report some statistics about this paper: (i) UA theorems due to no less than 13 different named authors for the three year period 1990-1992; (ii) 220 references, clustered usefully into categories (numbers in ( ) are number of references) such as: basic results (37), TS model (8), fuzzy rule patches (8), "complicated" implications (3), hierarchical systems (9), distributed systems (4), discrete systems (9), stability (9), neural networks (23), fuzzy neural networks (4), and our favorite, "how to choose the best variant of fuzzy rule-based modeling methodology (21). We are not making fun of these papers - we love them. (Our only complaint is that there aren't any references in the category "how to design a good rule-based classifier".)

Kreinovich et al. (1998) emphasize that there are at least three performance criteria besides the observed *mean squared error* (MSE) on test data that a good UA should possess; stability, computational simplicity, and smoothness. These authors present a really nice discussion that compares fuzzy systems to neural networks using each of these criteria. For us, perhaps the most important aspect of their discussion on this topic is that fuzzy systems are inherently less smooth than neural networks because the T-norms and T-conorms used during reasoning are - with rare exceptions - not smooth themselves. Arguably, this means that classifier functions represented by fuzzy systems will be, on average, less smooth than those built with neural networks. We think that the architecture and membership functions on the LHS of fuzzy systems are also very important when considering the overall smoothness of these two kinds of systems. In any case, this is a good thing to keep in mind when you set out to design that perfect classifier.

When XY appears to have no clusters, can we expect rules extracted *by clustering* to afford good approximations? Look again at $X_{21}Y_{21}$ in Figure 4.47 - how many clusters do you see? Most observers would say either "none" (no *sub*structure), or "one" (all of the data, viewed as a single curvilinear arc), so your initial reaction might be "No way [can you get rules with clustering]". How well would single input, single output rules for a simple MA model that are *extracted by clustering* in $X_{21}Y_{21}$ represent this system? Pretty well. Our next example also shows that the lack of smoothness in fuzzy systems approximations can sometimes be offset by replacing the usual PMFs such as triangular and trapezoidal fuzzy numbers with non-standard PMFs such as polynomials.

Example 4.17 Referring back to Figure 4.47, recall the data set $X_{21}Y_{21}$ of Narazaki and Ralescu. Figure 4.49 shows these data, along with c = 5 point prototypes {☆ $=\mathbf{v}_i$} in $\Re^2$ that lie on 5 prototypical line segments $\{L_i : \mathbf{x} = \mathbf{v}_i + t\mathbf{d}_i\}$ extracted by clustering $X_{21}Y_{21}$ with the fuzzy c-elliptotypes (FCE) algorithm (see equation (2.32)), implemented in the ACE interface (Runkler and Bezdek, 1998c). The lines have infinite extent, but the PMFs extracted from the data only provide each of them with support over a subinterval of [0, 1].

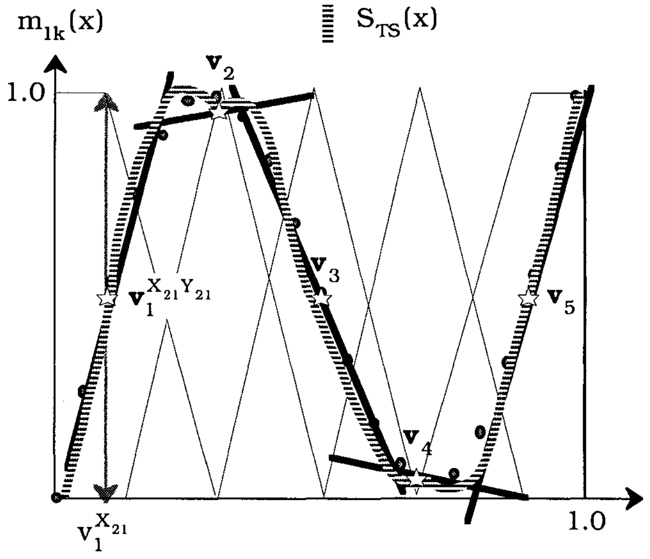


Figure 4.49 TS approximation of S in (4.83) using FCE-AO in the ACE interface with trapezoidal and triangular PMFs

The approximating fuzzy system is a first order TS system whose i-th output function is a line (written here in point slope form), $u_i(x) = s_i(x - x_i) + y_i$. Clustering with FCE provides estimates for the parameters of each $u_i(x)$ as $\mathbf{v}_i = (x_i, y_i)$, $s_i = b^y_{i1}/b^x_{i1}$, where b^x_{i1} and b^y_{i1} are the components of $\mathbf{b}^{XY}_{i1}$ in the x and y directions. To understand exactly how FCE produces these estimates, we repeat equation (2.32)

$$D^2_{10,ik} = \alpha \underbrace{D^2_{L_{1,ik}}}_{\text{lines}} + (1-\alpha)\underbrace{D^2_{L_{0,ik}}}_{\text{points}} \ ; \ 0 \le \alpha \le 1 \quad . \quad \text{(2.32, repeated)}$$

We did not provide a geometric interpretation for this measure of distance in Chapter 2, but think it useful here. Figure 4. 50 shows the geometry of the distance $D_{10,ik}$ used by the FCE objective function.

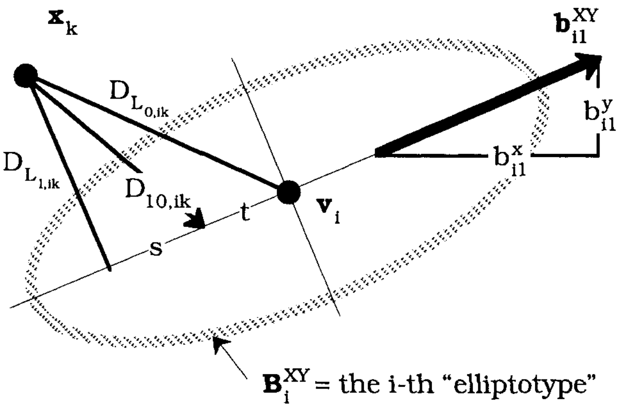


Figure 4.50 Geometric interpretation of FCE distance (2.32)

The distances s and t and the location of the "foot" of the line with length $D_{10,ik}$ in Figure 4.50 are controlled by α. When $\alpha = 1$, s = 0 and FCE becomes FCL with pure line prototypes. When $\alpha = 0$, t = 0 and FCE becomes FCM with pure point prototypes. For $0 < \alpha < 1$, the prototypes are not geometric entities with recognizable names (and in particular, they are *not* ellipses, as we pointed out in Chapter 2). But for any $\alpha > 0$, the lines component of FCE as shown in Figure 4.50 can be used to find linear prototypes. That's how Runkler and Bezdek (1998c) used FCE in the current application, and more importantly, that's how you can get lines from any clustering algorithm that associates a covariance matrix with each cluster.

Parameters for FCE in this example are $m = 2$, $\alpha = 0.001$, and the Euclidean norm was used in the objective function. This choice for α focuses most of the objective function's attention, when computing u_{ik}, on the distance $D_{L_{0,ik}} = \|\mathbf{x}_k - \mathbf{v}_i\|$. This choice makes FCE seek almost "all points", which forces the cluster centers into the data. The direction vector $\mathbf{b}_{i1}$ for the i-th line is the principal eigenvector of C_i, the i-th fuzzy covariance matrix in FCE (see Figure 4.50 and equation (2.27)). Rule i in the system under construction takes the simple form (don't confuse α in (2.32) with firing strength $\alpha_i(x)$ in R_i)

$$\text{IF } \alpha_i(x) \qquad \text{THEN } u_i(x) = s_i(x - x_i) + y_i;\ i = 1, 2, 3, 4, 5.$$

The premise membership functions $\{m_i(x)\}$ in Figure 4.49 are built by projecting the 2D point prototypes $\mathbf{V}^{X_{21}Y_{21}}$ from FCE onto the x axis. This is shown in full notation in Figure 4.49 for only the projection $\mathbf{v}_1^{X_{21}Y_{21}} \to v_1^{X_{21}}$. Then triangular membership functions are centered about v_2^X, v_3^X, v_4^X ; and trapezoidal membership functions are shouldered at v_1^X, v_5^X. The domains of positive support are chosen so that each PMF is zero at the same x at which the next PMF (to the right) is 1; because of this construction, the sum of PMF values at any input is 1.

In this SISO system each $x \in [0, v_1^{X_{21}}] \cup [v_5^{X_{21}}, 1]$ fires just one rule with firing strength 1. Each $x \in [v_1^{X_{21}}, v_5^{X_{21}}]$ will yield two values, say $m_j(x)$ and $m_{j+1}(x)$ from adjacent PMFs, and we know that $m_j(x) + m_{j+1}(x) = 1$. Equation (4.73) produces the output, which for this simple system becomes, for $j = 1, 2, 3, 4$,

$$\begin{aligned} S_{TS}(x) &= \frac{[m_j(x)\cdot(s_j(x - x_j) + y_j)] + [m_{j+1}(x)\cdot(s_{j+1}(x - x_{j+1}) + y_{j+1})]}{m_j(x) + m_{j+1}(x)} \\ &= [m_j(x)\cdot(s_j(x - x_j) + y_j)] + [m_{j+1}(x)\cdot(s_{j+1}(x - x_{j+1}) + y_{j+1})]. \end{aligned}$$

The approximation function $S_{TS}(x)$ produced by these 5 rules is plotted on Figure 4.49. Notice especially that this function is NOT smooth at the local maximum and local minimum of the underlying function S at (4.83) - it has cusps, so this TS model is not so smooth.

Runkler and Bezdek (1998c) give a second method for approximating S based on finding *piecewise polynomials* for the premise membership functions, and the result is a much smoother fit, both to the training data, and to test sets not in the training data. Table

4.35 reports the training and testing errors obtained with both schemes. The test data $X_{101}Y_{101}$ were generated by evaluating $S(x_k)$ at 101 input base points $x_k = x_{k-1} + 0.01;\ k = 1,\ldots,100\colon x_0 = 0$. We omit the Y factors of the data sets in Table 4.35 for brevity.

Table 4.35 Training and testing errors (in percent) for approximations to S(x) extracted from $X_{21}Y_{21}$ with FCE clustering

	Triangular PMFs		Polynomial PMFs	
# rules	$E_1(X_{21}\|X_{21})$	$E_1(X_{101}\|X_{21})$	$E_1(X_{21}\|X_{21})$	$E_1(X_{101}\|X_{21})$
2	36.30	37.60	28.40	28.70
4	15.30	15.50	11.90	11.40
5	6.40	6.03	5.01	4.98
11	4.64	4.03	3.85	3.50

The measures of test and training errors for these results were mean absolute relative errors (converted to % for the Table 4.35 values by multiplication by 100),

$$E_1(X_{te}|X_{tr}) = \frac{\sum_{x_k \in X_{te}} \left|S(x_k) - S_{TS}(x_k)\right|}{101 \cdot S(x_k)} \quad ; \text{ and} \tag{4.84a}$$

$$E_1(X_{tr}|X_{tr}) = \frac{\sum_{x_k \in X_{tr}} \left|S(x_k) - S_{TS}(x_k)\right|}{21 \cdot S(x_k)} \quad . \tag{4.84b}$$

Several observations about these results are in order. First, training and testing errors drop as the number of rules increases (i.e., as c, the number of clusters found in $X_{21}Y_{21}$ increases). Also notice that the improvement afforded by polynomial PMFs is highest when c is lowest. As the number of triangular PMFs increases, the conventional TS system becomes relatively better, but is never as good as the polynomial based system. Both of these trends will generally occur, and are due to the fineness of the fuzzy rule patches used by the approximating system.

Second, the generalization error is about half of that reported in Narazaki and Ralescu (1995) using various neural network approximations. This does NOT tell us that either TS model is better than the neural network models in any sense - it tells us that approximations of the same order of magnitude are easily obtained using both approaches. Finally, the use of polynomial membership functions in the antecedents of the rules smoothes out the approximation considerably.

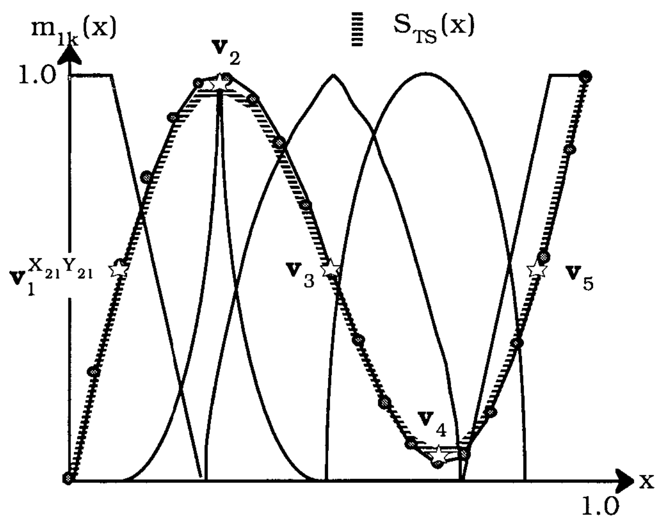


Figure 4.51 TS approximation of S in (4.83) using FCE-AO in the ACE interface with piecewise polynomial PMFs

The solid curve in Figure 4.51 is the graph of the function S that generates the training and test data, and the dashed curve is a pretty smooth approximation to it by the TS model with polynomial premise membership functions. Several of the PMFs, which are now piecewise polynomials, have cusps, but the cusps at the local minimum and maximum of the approximating function S_{TS} in Figure 4.49 have been eliminated.

Runkler and Bezdek (1998c) also present a second approach to the approximation problem in this example that is based on clustering with an algorithm built by selecting hyperconic membership functions and prototypes from the ACE toolbars (Section 2.6) that are not AO matched (that is, are not necessary conditions for minimization of an objective function by alternating optimization). Results from this second method are slightly better than those shown in Table 4.35, but the algorithm used was not discussed in Chapters 2 or 4.

How do we fix the *size of the rule-base* when we cluster to extract rules (cluster validity, hiding again)? In view of Table 4.35 in Example 4.17, tendency assessment and cluster validity seem relatively unimportant for rule extraction by clustering, because good approximations to **S** do not rely primarily on cluster substructure in the pattern recognition sense for their success. For reasonable functions, simply increasing c will almost always improve the approximation accuracy, as the clustering model responds with finer substructure (more rules). This is analogous to choosing smaller and smaller stepsizes for functional approximation as is done in classical numerical analysis. Many authors, however, do use validity functionals $\mathcal{V}$ when clustering for rules, and in this application $\mathcal{V}$ becomes essentially a pruning mechanism for the underlying fuzzy decision tree that maps to the fuzzy system. This trend probably began with Sugeno and Yasukawa (1993), who introduced $\mathcal{V}_{SY}$ expressly for this purpose. Other validity functionals that have been used this way include $\mathcal{V}_{XB}$ and $\mathcal{V}_{GG}$ (Pal et al., 1997b). Babuska and Kaymak (1995) use the compatible cluster merging (CCM) algorithm (Section 5.6.A) to find the number of linear clusters automatically.

Example 4.17 shows that extracting various parameters of a fuzzy system with clustering works. It is easy to find other examples in the literature of data that do not possess visual cluster structure but which, when clustered for rules, produce fuzzy systems that afford excellent approximations to the generating function. For example, Kim et al. (1997) discuss approximation of the MISO function

$$S(x,\dot{x}) = \left(1 + \frac{1}{x^2} + \frac{1}{\dot{x}^{1.5}}\right)^2 \quad ; \quad 1 \le x, \dot{x} \le 5 \qquad , \tag{4.85}$$

with rules extracted by clustering samples from (4.85). Sugeno and Yasukawa (1993) used samples from this function to illustrate function approximation for the 0-th order TS model. Sugeno and Yasukawa report a resubstitution MSE of 0.079 on 50 triples of IO training data using 6 0-th order fuzzy rules.

Kim et al. use the same IO data with fuzzy c-regression models (FCRM) clustering as discussed in Section 2.4 to extract 3 fuzzy rules for a first order TS system. The i-th rule, i = 1, 2, 3 is

$$R_i : \text{IF } \left[m_1^i(x) \wedge m_2^i(\dot{x})\right] \text{ THEN } u_i(x) = a_i + b_i x + c_i \dot{x}, \tag{4.86}$$

where the PMFs are Gaussian, $m_j^i(z) = e^{-\left((z-\mu_j^i)/\sigma_j^i\right)^2}$. Parameters of the LHS PMFs $\{\lambda_j^i = (\mu_j^i, \sigma_j^i)\}$ and RHS output functions $\{\rho_i = (a_i, b_i, c_i)\}$ are estimated by clustering in XY with FCRM. Specifically, FCRM fuzzy partition U yields initial estimates of the Gaussian PMFs as

$$\hat{\mu}_j^i = \frac{\sum_{k=1}^{n} u_{ik} x_{kj}}{\sum_{k=1}^{n} u_{ik}} \quad ; \quad \hat{\sigma}_j^i = \sqrt{\frac{\sum_{k=1}^{n} u_{ik}(x_{kj} - \hat{\mu}_j^i)^2}{\sum_{k=1}^{n} u_{ik}}} \quad . \tag{4.87}$$

Initial parameters for the RHS output functions are obtained directly from FCRM as linear regression functions (that is, local linear models of the IO data). The final step in Kim et al.'s approach is to fine tune both sets of parameters $\{(\hat{\lambda}^i_j, \hat{p}_i)\}$ using gradient descent. They report that the final set of three fuzzy rules produces a resubstitution MSE of 0.0551 - an improvement over the error reported by Sugeno and Yasukawa (1993).

The models used by Runkler and Bezdek (1998c) and Kim et al. (1997) have two important things in common, and one important difference. The big difference between these two approaches lies in the use of the clustering outputs. Both methods use the IO data XY to find estimates for $(U^{XY}, \mathbf{B}^{XY})$, and in both cases the $\{\mathbf{b}_i^{XY}\}$ are linear prototypes. However, Runkler and Bezdek essentially ignore the fuzzy partition $U^{XY} \in M_{fcn}$, and use only the prototypes $\{\mathbf{b}_i^{XY}\}$ during construction of the rules. Kim et al., on the other hand, chose to use everything the clustering algorithm provides them, viz., $(U^{XY}, \mathbf{B}^{XY})$. There is no reason to prefer one scheme to the other, and more generally, Pal et al. (1997b) survey many other schemes besides these two that use the information extracted from XY by $\mathcal{C}$ in other ways. We are not willing to say that there is a "best way" to use the information you can get from clustering XY to extract rules; we think the choice is dictated by a number of factors, one of the most important of which, and the one you have the least control of, is the data itself. However, the similarities between Runkler and Bezdek (1998c) and Kim et al. (1997) do give us one clue.

In both Runkler and Bezdek (1998c) and Kim et al. (1997) the underlying fuzzy system is a first order TS model, and the clustering algorithms used can both generate linear prototypes. Thus, $\mathcal{C}$ produces direct estimates of the TS output functions for this case. Functional approximation with linear models is hardly new. After all, the geometric meaning of the derivative of any real function at a point is that its value gives us the slope of the line tangent to the graph of the function at this point, and the tangent line provides the best local linear approximation to the curve. Our supposition is that clustering algorithms most effectively extract TS rules when their prototypes match TS output functions. If this is correct, then the best choice for $\mathcal{C}$ if you are building a first order TS system would seem to be any clustering algorithm that is capable of generating lines in the product space. This includes, for example, the GK, GMD, FCL, FCE, RFCM, and FCQS algorithms discussed in Chapter 2, and any other $\mathcal{C}$ that involves hyperellipsoidal clusters with covariance matrices (such as the model of Nakamori and Ryoke, 1994), whose principal eigenvectors can be used to supply lines through the corresponding cluster centers.

Extending this idea, if you wanted local quadratic approximations, then a second order TS model would be appropriate, and you would have a somewhat more limited set of natural choices for the clustering algorithm $\mathcal{C}$, which in this case would have to be able to generate quadratic prototypes. Thus, you might try RFCM or FQRS. We will discuss several other ways to build fuzzy systems for function approximation with clustering, but the fact that some clustering non-point prototype algorithms can produce direct estimates of first and second order polynomials, coupled with the fact that first and second order TS models have exactly these functional forms on the RHS of the rule-base, suggest to us that this is probably the best combination of fuzzy systems and clustering for function approximation.

Having some examples of function approximation by rules extracted with clustering under our belts, we ask some general questions about the use of clustering in this domain. *What tasks* in the design of a rule-based fuzzy system can be relegated to clustering? *Where* should we cluster, X, Y, XY, or all of these? *What clustering algorithm* should we use? How do we *use the clustering outputs* in c-partitions $U^X, U^Y, \text{or } U^{XY}$; point prototypes $\mathbf{V}^X, \mathbf{V}^Y$, or $\mathbf{V}^{XY}$; and non-point prototypes $\mathbf{B}^X$, $\mathbf{B}^Y$, or $\mathbf{B}^{XY}$ to create pieces of a fuzzy system? *What might go wrong* when clustering is used for rule extraction? We address these questions, but like many topics in this book, functional approximation by clustering is an area of <u>right-now</u> research, so don't expect definitive off-the-shelf answers. Instead, look for the general ideas, and think of ways to improve them.

What tasks in the design of a rule-based fuzzy system can be relegated to clustering? Table 4.36 shows nine tasks involved in the establishment of $\mathcal{R}$ that seem most amenable to clustering.

Table 4.36 What humans (👥) and clustering (✱) can do for the MA and TS fuzzy systems

Left Side of the Rule Base

👤		1	Select input variables $x_1, \ldots, x_p$
		2	For i = 1 to p: choose or find:
👥		2a	numerical range D_i for x_i
👤		2b	linguistic variable $\mathcal{L}_i$
👤	✱	2c	the # r of linguistic values for $\mathcal{L}_i$
👥	✱	2d	linguistic values $\{\ell_{ij}\}$ for $\mathcal{L}_i$
👤	✱	2e	PMFs $\{m_{ij}\}, 1 \le j \le r$
👥	✱	3	Select the number of rules, c
👥	✱	4	Define the structure of each rule
👤		5	Select T-norm $T = \cap$

Table 4.36 (con't.) What humans (🧑🙂) and clustering (✶) can do for the MA and TS fuzzy systems

Right Side of the Rule Base

🙂		6	Select output variables $z_1, \ldots, z_q$
🧑		7TS	Select forms of $\mathbf{u}_i$, $1 \le i \le c$
🙂	✶	8TS	Determine parameters of the $\{\mathbf{u}_i\}$
▨	▨	7MA	For k = 1 to q: choose or find:
🧑		7MAa	numerical range Do_k for z_k
🙂		7MAb	linguistic variable $\mathcal{L}o_k$
🧑	✶	7MAc	# s of linguistic values for $\mathcal{L}o_k$
🧑		7MAd	linguistic values $\{\ell o_{kj}\}$ for $\mathcal{L}o_k$
🙂	✶	7MAe	CMFs $\{mo_{kj}\}, 1 \le j \le s$
🧑		8MA	Select T-conorm $\cup$
🙂		9MA	Select defuzzification operator D_F
🧑	✶	10	Couple LHS-RHS (choose $\Rightarrow$)

Steps 1, 2a, 2b, 2d, 5, 6, 7TS, 7MAa, 7MAb, 7MAd, 8MA and 9MA in Table 1 are always done by the modeler, perhaps with the help of an expert. For example, although each cluster may correspond to a linguistic value in the LHS or RHS of a rule-base, linguistic values are *words* such as "high", "fast", "light" that *must be* chosen by humans, but the PMFs and CMFs that correspond to each of these words can be chosen by humans, or discovered by clustering. The first column of the table shows you that humans can (and often) do all of the remaining tasks too. The hypothesis for this subsection is that clustering may be able to do some of them more reliably, and perhaps more efficiently. We will discuss some clustering methods that have been used to replace intuition and/or trial and error in one or more of steps 2c, 2e, 3, 4, 7MAc, 7MAe, 8TS and 10 in Table 4.36.

Where should we cluster: X, Y, XY, or all of these? This interesting question has no easy answer, since it's easy to give examples where each domain is needed, and other examples where each domain fails. Figure 4.52 illustrates a situation where c = 4 in XY, c = 3 in Y because Y_1 and Y_4 will be mixed into one cluster, and c = 2 in X because of the mixing of X_1 with X_2 and X_3 with X_4. (We have "lifted" the projections of X_3Y_3 onto $\Re^p$ and X_4Y_4 onto $\Re^q$ so you can see them.) If you had a reliable cluster validity function or other means for discovering the "right" number of clusters, you would not obtain consistent results when comparing the rules suggested by clustering in these three domains.

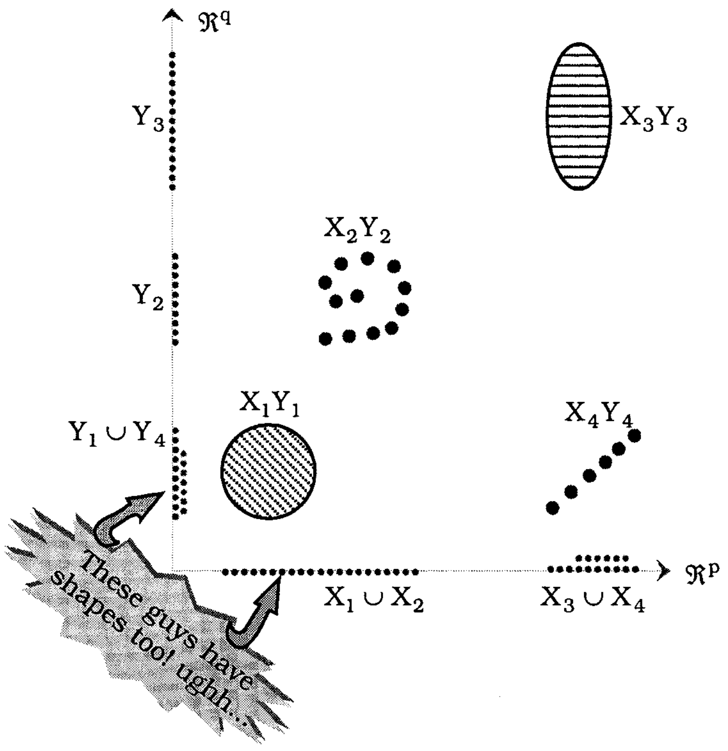


Figure 4.52 Different numbers of clusters in all three domains

Figure 4.52 illustrates an important point that often causes confusion when clustering is used to build fuzzy systems. The tacit assumption in the pattern recognition use of clustering (chapter 2) is that some unlabeled data set has "clusters", and all we want to do is find them. The data in Figure 4.52 do have visually apparent clusters in each of X, Y and XY. The problem here, however, is that the clusters don't seem to properly reflect the additional information we have in this application - viz., that the labels tell us there is a functional relationship between the input and output pairs in the training data. There may be rules that cover the data in Figure 4.52, but our point in this figure is that discovering the rules *by clustering* might be difficult, if not impossible in this situation.

We have already discussed the principle of matching prototype shapes to TS output functions. Another point about Figure 4.52 concerns the shapes of the clusters in the data. We know from Chapter 2 that one of the principal desires for clustering algorithms when used in the context of pattern recognition is that the model underlying them match the geometric shapes of the clusters. In Chapter 2 we discussed models that attempt to match ellipsoidal shapes - volumetric or cloud clusters. Most cloud seeking models are point prototype models that looked for central tendencies in X, and represent structure with point prototypes. Shell clusters, on the other hand, are best matched by non-point prototype models. You see the shapes of the clusters in XY in Figure 4.52. A single clustering model would have a hard time matching the variety of shapes in the clusters you see in Figure 4.52. Moreover, in this illustration it looks like the clusters in X and Y are linear, but this is an artifact of the drawing - X and Y are sets in $\Re^p$ and $\Re^q$, and they can also have a variety of shapes, perhaps all different. And finally, for p, q > 3, you have very little information about cluster shapes in any of the three domains. Nonetheless, to the extent possible, the choice of $\mathcal{C}$ should also be dictated by any knowledge you can glean about cluster shapes.

Figure 4.53 illustrates a case where there appear to be c = 2 clusters in XY, and c = 4 clusters in both X and Y.

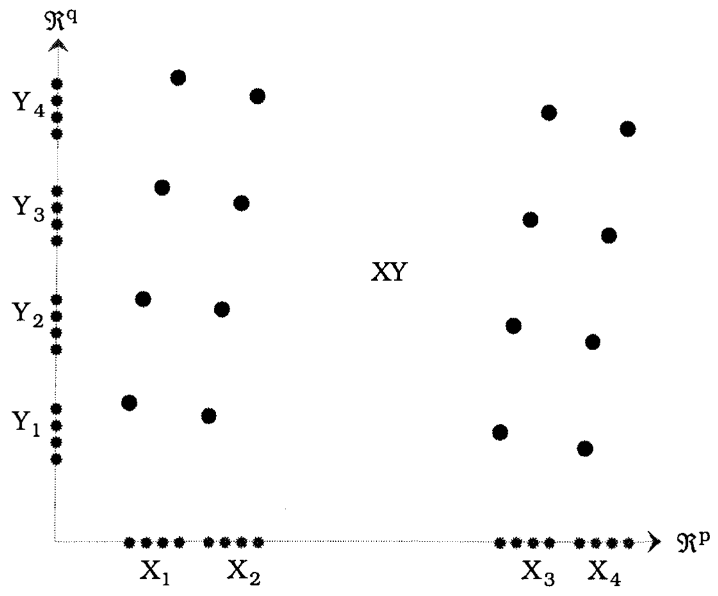


Figure 4.53 Different numbers of clusters in the product and factors

Trying to extract rules for function approximation from the data shown in Figures 4.52 or 4.53 by clustering may lead to very confusing results. When we want to build a *classifier* with rules, the training outputs are usually crisp label vectors, and this presents a somewhat different situation. Figure 4.54 illustrates a case where there are c = 2 crisply labeled classes in the training data. In the upper view in Figure 4.54, the input data X lie along the horizontal axis, and the output data (crisp labels or label vectors) lie along the vertical axis.

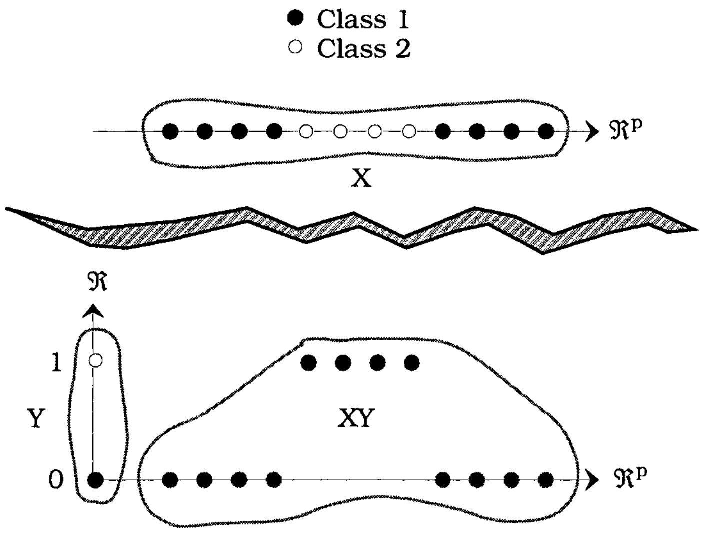


Figure 4.54 An XOR-like data set for classifier design

We can arrange the scales of the data so that there are c = 1, 2, 3, ..., 12 clusters in X. For example, with p = 1 we might have

X_1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}	: c=1
X_2 = {1, 2, 3, 4, 101, 102, 103, 104, 105, 106, 107, 108}	: c=2
X_3 = {1, 2, 3, 4, 101, 102, 103, 104, 201, 201, 203, 204}	: c=3
X_4 = {1, 2, 3, 101, 102, 103, 201, 201, 203, 301, 302, 303}	: c=4
etc... etc...............	etc.

Now suppose we append the label 0 to the class 1 feature vectors and the label 1 to the class 2 feature vectors - that is, the target output set is Y = {0, 1}. In the product space $\Re^{p+1}$, shown in the lower view of Figure 4.54, the IO data XY will have the general form $\mathbf{x} = (x_1,\ldots,x_p,0)^T$ for class 1, and $\mathbf{x} = (x_1,\ldots,x_p,1)^T$ for class 2. There are either c = 1 or c = 2 clusters in Y: your assessment will depend on the relationship of Y to X. For example, if the input data were

X = {11, 11.1, 11.11, ..., 11.1111111111} , : c=1

a scatterplot of XY at equal resolution along each axis would suggest that there was c = 1 cluster in X, 2 in Y, and 2 in XY. On the other hand, for the input data

X = {1, 102, 103, 104, 201, 202, 203, 204, 301, 302, 303, 304}, : c=4 a scatterplot would suggest 4 clusters in X, 1 in Y, and most likely 3 in XY. The point here is that IO data for classifier design is somewhat different than for functional approximation, because the outputs are not continuously valued, nor do they necessarily satisfy any "smoothness" criterion as they might in the functional approximation problem.

Figures 4.52-4.54 illustrate the difficulty of proposing a guideline about where to cluster that reliably covers all possible cases. In a recent survey by Pal et al. (1997b) of 14 papers on rule extraction by clustering, the authors of the papers studied used either MA and TS models (or both), or some hybrid of one of them. Of the 14 authors, 11 advocated clustering in XY, 3 clustered in X, and 2 clustered in Y. One set of authors (Delgado et al., 1997) clustered in all three spaces, and one set (Nakimori and Ryoke, 1994) clustered in part of XY.

Another important consideration when choosing the proper domain for clustering is the relationship of p to q. In every example we know of where functions are approximated by fuzzy systems that are derived by clustering, the input and output domains have roughly the same (order of magnitude of) dimensions. We know of at least one industrial application at Siemens in Germany where p is about 220 and q = 1 (this application is proprietary, so we can't give you a reference). What if, for example, p = 200, q = 1? Do you think this would have any effect on the efficacy of clustering to extract rules? Most clustering algorithms eventually rest their cases on distance calculations. For example, if you cluster in XY with any of the c-means models in this situation, you will need to make calculations that entail distances like $\|(\mathbf{x}_k, y_k) - \mathbf{v}_i\|^2$. Using the Euclidean norm, for instance with p = 200, q = 1, we can write this distance in component form as

$$\|(\mathbf{x}_k, y_k) - \mathbf{v}_i\|^2 = \underbrace{\left(\sum_{j=1}^{200}(x_{jk} - v_{ji})^2\right)}_{\text{input}} + \underbrace{(y_k - v_{201,i})^2}_{\text{output}} \quad . \tag{4.88}$$

If the scales of values in the input and output features are about the same, the input feature values will certainly dominate the distance, essentially masking the contribution of the output values to the location of cluster centers in the product space $\Re^{201}$. The same remark applies to the opposite scenario, when p = 1, q = 200. Clusters discovered in XY's with such imbalance may be a poor choice for representing IO relationships. We are not aware of any studies that investigate this problem in relation to rule extraction by clustering, and what to do about it, but we think it is a problem that deserves careful attention if and when the dimensions p and q differ by more than a handful of integers. To shoot from the hip (risking a total miss, of course), we suggest trying joint statistical normalization, so that each feature in XY has sample mean 0 and sample variance 1 when p and q differ by more than a half dozen or so integers. This will at least partially offset the effect of inequitable domination of the joint distances by one set of variables or the other when making calculations like those in (4.88) during clustering in XY.

Chiu (1997) unequivocally states that when the rules are for approximation of functions with continuous outputs, clustering should always be in XY, and when the function being approximated is a classifier function (as would be the case in Figure 4.54), clustering should be in each of the crisply labeled subsets of X alone. Since each of the c subsets of X can be separately clustered into say, c_i rules, $\mathcal{R}$ will be subdivided into c subsets of rules (one set for each class in XY) using this scheme. We tend to agree with Chiu's advice about where to cluster when the labels are crisp, because this strategy is in line with our general belief that the RHS of the TS model should be chosen to reflect the geometry of the function being approximated. Here, the function is not continuous (many inputs from $\Re^p$ may cause the same response in $N_{pc} \subset \Re^c$). However, if the training data have soft labels they cannot be subdivided and clustered separately, and we are back to the question of where best to cluster, X, Y or XY.

Suppose $X = X_1 \cup \cdots \cup X_c \subset \Re^p$ has c crisply labeled classes with $|X_i| = n_i$ for i = 1 to c. Chiu applies the SCM clustering algorithm to each X_i separately, obtaining, say, c_i clusters for X_i, i = 1,...,c. Each of the clusters thus contributes c_i rules to the rule-base, and the total number of rules is $M = \sum_{i=1}^{c} c_i$. Since there are c subsets of rules corresponding to the c subsets in X, we add an index to the rules that indicates which class they pertain to: let R_{ij}, α_{ij} and u_{ij} denote the j-th rule, its firing strength, and its output function for the i-th class, respectively, j=1,...,c_i; i=1,...,c. Chiu uses a variation of the zero-th order TS model for which the ij-th rule has the general form

$$R_{ij}: \ \alpha_{ij}(\mathbf{x}) \Rightarrow u_{ij}(\mathbf{x}) = i, i = 1, \ldots, c; j = 1, \ldots, c_i \quad . \tag{4.89}$$

Chiu uses the T_2 norm (product) to compute the LHS firing strengths in (4.89), and departs from the standard TS model by abandoning the general TS output formula in equation (4.73). Instead, Chiu computes the output of $\mathcal{R}(\mathbf{x})$ for a given input $\mathbf{x} \in \Re^p$ as

$$\mathcal{R}_{\text{Chiu}}(\mathbf{x}) = i \Leftrightarrow \alpha_{ij}(\mathbf{x}) = \underbrace{\max}_{\substack{1<k<c \\ 1 \le s \le c_k}} \{\alpha_{ks}(\mathbf{x})\} \quad .$$

Although many writers have used other methods to design fuzzy rule-based classifiers, only a few have used clustering towards this end. Here we abstract an example presented by Chiu (1997) on- yipes! - the Iris (but which one ?) data.

Example 4.18 Chiu (1997) advocates the use of clustering to extract rules for classifier design based on his subtractive clustering method (SCM), which is related to the mountain clustering method of Yager and Filev (1994a, b). Since this is an example of classifier design, the crisp labels of Iris play an active role in the development of the fuzzy rules.

First, like many before him, Chiu drops the first two features in Iris, so the data set for which results are discussed is really the 2D data set X = Iris_{34} (Figure 4.12). This simplification not only makes the classifier work better, but more importantly, means that we are looking for a 2 input, single output system. Chiu says he normalizes the input data, but does not give the method of normalization. However, the domains of the extracted rules suggest that he multiplied Iris_{34} by 10. The output training data consist of the integers 1, 2 and 3, corresponding, respectively, to the crisp labels of the three classes in Iris. Consequently, the model being developed is a 0-th order TS model - that is, a TS system with crisp singleton output functions, $u_i(\mathbf{x}) = i$, i=1, 2, 3. Remember that here $u_i(\mathbf{x})$ is simply a label to identify a class; the rule-base $\mathcal{R}$ does *not* attempt to approximate the numerical values 1, 2 or 3.

Chiu subdivides Iris_{34} into its three 50 sample components, and separates each subset into 40 training data and 10 test data (the method of subdivision is not specified). Recall that the MCM and SCM objective function is $J_{\mathrm{MCM},1}(\mathbf{v}_j;X) = \sum_{k=1}^{n} e^{-\alpha\delta(\mathbf{v}_j,\mathbf{x}_k)}$. Chiu defines a constant $r = \sqrt{4/\alpha}$ that he calls the *SCM cluster radius* for all prototypes. In the example being discussed r = 0.5 (so α = 16).

Chiu clusters the 40 points in each subtraining set, and finds that c = 1 cluster (i.e., one SCM prototype per class) is sufficient to produce a training error of 3/120 = 2.5% on the training data, and 0/30 = 0% apparent error on the 30 test data. This is a somewhat curious reversal of the usual case, where the resubstitution error is lower than the test error. But that's pattern recognition - it all depends on the points you happen to use for training and testing! Each of the three rules found by this process started with a set of two PMFs, which were "two-sided" Gaussians, and the PMFs were optimized by the gradient descent method given in Chiu (1995).

Figure 4.55 is our adaptation of Chiu's (1997) Figure 9.5, which shows the three rules extracted by this process in a pleasant graphical style. Chiu does not identify linguistic values for the two sets of three premise membership functions shown vertically in Figure 4.55, so we have assigned them the values "Low", "Medium" and "High" simply to make this example more uniform with previous illustrations. Chiu does not give functional forms for the PMFs either, and although the two sets are not visually identical, they are certainly very similar.

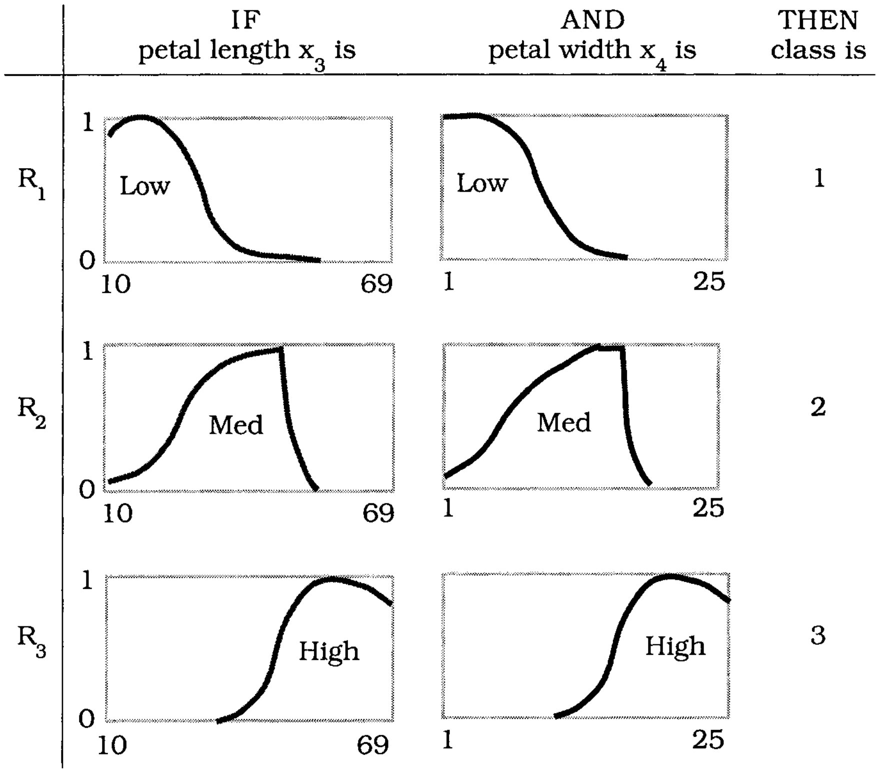


Figure 4.55 SCM rules for classifying Iris$_{34}$ (Chiu, 1997)

According to Chiu, Figure 4.55 shows the linguistic aspect of rule-based classification to good effect. For example, he asserts that the first rule essentially states that flowers with small petals (small petal length and petal width) are Iris Sestosa; that medium size petals are class 2 (Iris Versicolor), and the Iris Virginica (class 3) have relatively large petals. Although it might take you a while - say, 10 minutes - this conclusion can be reached by simply looking at the values of features three and four of Iris (try it - look at the third and fourth columns of Iris in Appendix 2, or at the scatterplot in Figure 4.12). This is not to take away from Chiu's example, for if the data had, say, 100 variables, an exercise like this would be an exercise in sheer folly.

Now we return to the remaining questions on our checklist about using clustering to extract fuzzy rules. The main questions outstanding are: what *clustering algorithm* $\mathcal{C}$ should we use? ; and how do we *use the clustering outputs* in c-partitions U^X, U^Y, or U^{XY}; point prototypes $\mathbf{V}^X, \mathbf{V}^Y$, or $\mathbf{V}^{XY}$; and non-point prototypes $\mathbf{B}^X$, $\mathbf{B}^Y$, or $\mathbf{B}^{XY}$ to create pieces of a fuzzy system? We have already provided one answer for these questions - viz., match the prototypes **B** to TS output functions, and the answers to them are almost inseparable, so we tackle them together here in a little more detail.

The most important distinction (after the types of functions being approximated) between various approaches to rule extraction by clustering seems to be whether the clustering algorithm generates point prototypes or non-point prototypes. When $\mathcal{C}$ produces point prototypes, they are usually used to locate central tendencies in the input and output domains, and the memberships from U are used to (somehow) produce at least initial estimates of the PMFs and CMFs (or output functions in the TS model) which are "centered" about the prototypes. This case is best understood by first studying the situation for crisp partitions of the data.

Let $U^X \leftrightarrow \{X_1, \ldots, X_c\}$ be any crisp c-partition of X. When Y=**S**(X), under the assumption that **S** is a 1-1 function, each cluster $X_i \subset X$ is carried to a crisp subset $Y_i = \mathbf{S}[X_i] \subset Y$, and the labels of points in Y_i are inherited from those in X_i. Moreover, $Y = \bigcup_{i=1}^{c} Y_i$ and $Y_i \cap Y_j = \emptyset$ for $i \neq j$. Consequently $\{Y_i\}$ is a crisp c-partition of Y with the *same partition matrix* as X, $U^X = U^Y$. We say that U^Y is **S**-*induced* on Y by the pair (**S**, U^X), and indicate this by writing $U^X \bar{\mathbf{s}} U^Y$. (**S**, U^X) also induces (the same) crisp c-partition $\{XY_i\} \leftrightarrow U^{XY}$ on XY, viz., $U^X \bar{\mathbf{s}} U^{XY}$. Similarly, if we start with a crisp partition U^Y of Y, the pair (**S**, U^Y) induces the same crisp partition on X and XY (but if the relationship of IO pairs is not 1-1, the same $\mathbf{x} \in X$ or $\mathbf{y} \in Y$ may end up with more than one label vector). And if the beginning partition is of XY, it induces, using the forward and inverse algebras of sets, the same partitions on X and Y. Thus, our assumption that **S** is 1-1 and that $\mathbf{y} = \mathbf{S}(\mathbf{x})$ for every $(\mathbf{x}, \mathbf{y}) \in XY$ insures a unique correspondence between crisp partitions and sample means of the sets X, Y and XY, namely, $(X, U^X, \bar{\mathbf{V}}^X) \leftrightarrow (Y, U^Y, \bar{\mathbf{V}}^Y) \leftrightarrow (XY, U^{XY}, \bar{\mathbf{V}}^{XY})$ with $U^X \bar{\mathbf{s}} U^Y \bar{\mathbf{s}} U^{XY}$.

Many authors correctly call $(U^X, \overline{V}^X)$ and $(U^Y, \overline{V}^Y)$ "projections" by **S** of $(U^{XY}, \overline{V}^{XY})$. These operations and this terminology carry over to fuzzy, probabilistic and possibilistic partitions created by clustering in X, Y or XY. Thus, the projection of a fuzzy $U^{XY} \in M_{fcn}$ to X and Y, for example, simply means $(X, U^X) \leftrightarrow (Y, U^Y) \leftrightarrow (XY, U^{XY})$ with $U^X \bar{\bar{s}} U^Y \bar{\bar{s}} U^{XY}$. On the other hand, we are aware of papers that use the values in one or more of these three partitions in a functional (as opposed to partitional) role, and in some cases the authors again refer - incorrectly - to the use of an induced U as projection. So, be careful about this term.

The situation illustrated in Figure 4.52 - where **S** is not a 1-1 function - makes it clear that U^{XY}, for example, may partition XY in a "nice" way, but this does not imply that the **S**-induced partitions U^Y and U^X are equally "nice" partitions of Y and X. Thus, in Figure 4.52 a natural partition of XY into 4 clusters would induce quite unnatural partitions on X and Y in the input and output domains. This comment applies as well to partitions induced on the other sets starting from Y or X, and it bears importantly on the question of which of the three sets, X, Y or XY, is the appropriate domain for clustering in the context of rule extraction.

From the approximation point of view, **S** should be at least continuous, and if it is a 1-1 function it will be invertible. Figure 4.56 shows the (ideal) relationship between the input and output data that seems to underlie many methods based on point prototype clustering algorithms. The assumption of continuity (which cannot be verified for computational representations of **S** anyway, but which is important to recognize from the analytical point of view) is the key one. Continuous functions bind neighborhoods in the three domains together (but continuity alone is not enough to guarantee that disjoint sets are carried to disjoint sets); and contained in these neighborhoods, we hope, will be the crisp clusters in X, Y and XY found by $\mathcal{C}$, and the sample means of the crisp clusters give us their central tendencies. The likelihood that our hope will be fulfilled depends on many factors, the principal one of which is that the data actually come from a smooth process.

Consider the sample means $\{\overline{\mathbf{v}}_i^X\} \overset{\mathbf{S}}{\leftrightarrow} \{\overline{\mathbf{v}}_i^Y\}$ of the clusters in a pair of crisp partitions of X and Y where $U^X \bar{\bar{s}} U^Y$. If **S** were *linear* we would have $\overline{\mathbf{v}}_i^Y = \mathbf{S}(\overline{\mathbf{v}}_i^X)$ for each i. This is far too strong for our purposes, but if **S** is continuous, every neighborhood of $\overline{\mathbf{v}}_i^X$ will map to a neighborhood of $\overline{\mathbf{v}}_i^Y$ as shown in Figure 4.56.

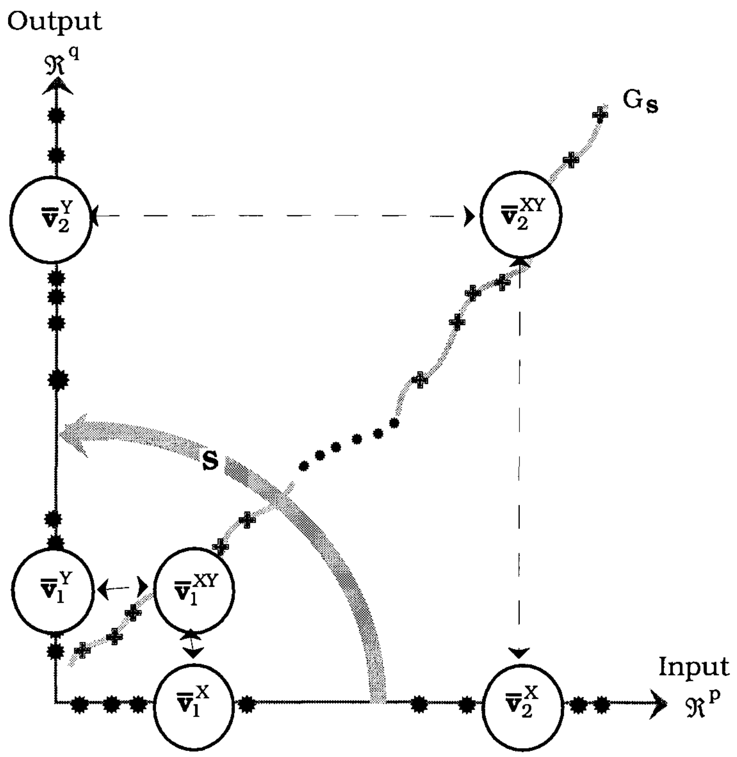


Figure 4.56 Relationships between neighborhoods, crisp clusters and sample means in X, Y and XY data when S is continuous

Continuity means that for any $\varepsilon > 0$ there is a $\delta(\varepsilon) > 0$ so that $\|\mathbf{x}_k - \bar{\mathbf{v}}_i^X\| < \delta(\varepsilon) \Rightarrow \|\mathbf{y}_k - \bar{\mathbf{v}}_i^Y\| < \varepsilon$. Consequently, it is reasonable to assume that when $\|\mathbf{x}_k - \bar{\mathbf{v}}_i^X\|$ is small, $\|\mathbf{y}_k - \bar{\mathbf{v}}_i^Y\|$ will be too (this is an assumption because $\delta(\varepsilon)$ could be very large for a very small ε). This assumption enables us to (conceptually) translate the i-th cluster into an i-th fuzzy rule :

MA models : If $\mathbf{x}$ is *close* to $\bar{\mathbf{v}}_i^X$ then $\mathbf{y}$ is *close* to $\bar{\mathbf{v}}_i^Y$; (4.90a)

TS models : If $\mathbf{x}$ is *close* to $\bar{\mathbf{v}}_i^X$ then $\mathbf{y} = \mathbf{u}_i(\mathbf{x})$. (4.90b)

Point prototypes are almost always used for 0-th order TS models. Usually the antecedent part (LHS) of either form in (4.90) is written as a conjunction of p atomic clauses,

If(x_1 is *close* to $\bar{v}_{i1}^X$ $\cdots$ and $\cdots$ x_p is *close* to $\bar{v}_{ip}^X$) . (4.90c)

As soon as we make the term "close to" precise, equation (4.90) extracts rules from crisp clusters in X, Y or XY using their sample means. For convenience we refer to this method as *crisp rule extraction* (CRE). One role played by the crisp membership functions in CRE is to identify the points from which $\{\bar{\mathbf{v}}_i^X\} \overset{s}{\leftrightarrow} \{\bar{\mathbf{v}}_i^Y\} \overset{s}{\leftrightarrow} \{\bar{\mathbf{v}}_i^{XY}\}$ are built. Once this is done, the PMFs $\{m_{ij}\}$ (and for the MA model, the CMFs $\{mo_{ij}\}$) can be erected in $\Re^p$ and $\Re^q$ in several ways.

Crisp membership functions defined by the rows of any crisp partition are supported by discrete sets of points, and the membership values (there are c of them over each support point, but c-1 of them have the value 0) are 0's and spikes of height 1. This is illustrated in Figure 4.57 for c = 2 and p = q =1, where $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$ and $XY = X_1Y_1 \cup X_2Y_2$ all share the identical crisp 2-partition

$$U^X \equiv U^Y \equiv U^{XY} = \underbrace{\begin{matrix} 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \end{matrix}}_{\substack{\text{cluster 1} \\ \text{(ducks)}}} \; \underbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \end{matrix}}_{\substack{\text{cluster 2} \\ \text{(llamas)}}} \quad . \qquad (4.91)$$

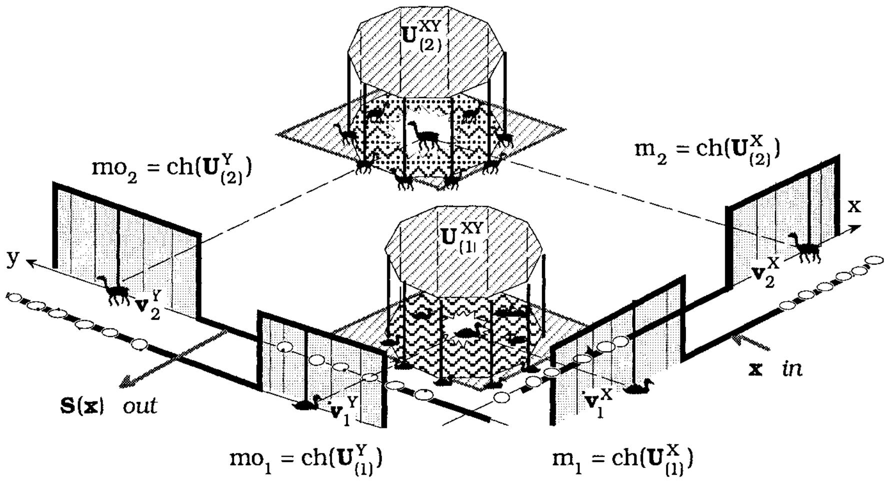


Figure 4.57 CRE for the MA model from 2 crisp clusters and sample means (not shown : zero values of U^{XY_1} and U^{XY_2})

A common way to make crisp membership functions from these rows is to take the *convex hull* of the data corresponding to each row in the projected crisp partitions U^X and U^Y as the numerical domain of input and output membership functions. For example, the convex hull of the first row of U^X yields the domain for the crisp

PMF $m_1 = ch(\mathbf{U}^X_{(1)})$ shown in Figure 4.57, etc. Graphs of the PMFs $\{m_j\}$ and CMFs $\{mo_j\}$ for linguistic termsets of granularity r = s = 2 are the rectangular functions defined over domains $\{D_j\}$ and $\{Do_j\}$, which are the convex hulls of $X_i, Y_i, i = 1, 2$.

We have displaced the CMFs and PMFs in Figure 4.57 so you can see them as the two rows of the matrix in (4.91), and thus can extend your imagination to a similar figure for any value of c. In reality, all c membership functions will be distributed along a single axis, that is, the c rows of U^X and U^Y will result in c crisp membership functions in the input and output domains along each input and output variable axis. Figure 4.57 depicts the ideal case, where the clusters are well separated so their projections don't overlap. Often, however, the cluster substructure is mixed (overlapping), and then the nice clean picture shown in Figure 4.57 can deteriorate into a real mess.

There can only be 2 rules for Figure 4.57 and because $m_1 \cap m_2 = \emptyset$, a given input fires just one of them. Since the membership functions are crisp, any T norm in the antecedents of the rules will return the number 1 as the firing strength for any input. For the MA model shown in Figure 4.57, $y = S_{MA}(x)$ depends on the choice of D_F. If we use the center of gravity approach, and let COG_1, COG_2 denote the y coordinate of the COG of the rectangular functions mo_1 and mo_2, respectively, we get : IF $x \in D_j$ THEN $y = S_{MA}(x) = COG_j, j = 1, 2$. Thus, the implemented MA approximation for **S** would be a function, but not 1-1, and would not generalize well at all. In the TS model, the result of firing rule i, i=1, 2 is simply $y = S_{TS}(x) = u_i(x)$. This is a pretty uninspired use of the clustering outputs: we can do much better.

Understanding Figure 4.57 leads to an appreciation of how the rows of projected non-crisp partitions can be used in various ways to soften crisp rule extraction. Please compare Figure 4.57 to Figure 4.28 to see the relationship between the crisp rules extracted by clustering and crisp rules that you might get from a decision tree approach. Figure 4.28 shows you the same crisp rule patches that you see in the product domain in Figure 4.57. The crisp rule patches extend beyond the training data so that the rule-base can produce outputs for non-training inputs. This immediately shows us why rule extraction based on (4.89) with crisp clustering algorithms is not very robust - the rules suffer from exactly the same problem as crisp decision trees. Thus, we follow path similar the one taken by Chang and Pavlidis (1977). Here, the failure of crisp patches leads us to the use of fuzzy or possibilistic clustering algorithms for rule extraction in function approximation.

There are many *soft rule extraction* (SRE) methods based on clustering (e.g., Sugeno and Yasukawa, 1993, Yoshinari et al., 1993, Sin and deFigueiredo, 1993, Yager and Filev, 1994b, Nakamori and Ryoke, 1994, Chiu, 1994,1995, Babuska and Kaymak, 1995, Cheng et al., 1995, Runkler & Palm, 1996, Delgado et al., 1997, Kim et al., 1997, Runkler and Bezdek, 1998c, 1999). The number of ways authors have used the information produced by clustering for SRE defies an intelligent (computational, artificial *or* biological intelligence!) classification of methodologies. We are content here to illustrate several approaches of SRE for function approximation, and refer you to the literature for detailed discussions.

If you apply any soft clustering algorithm to (say) XY that results in a pair $(U^{XY}, \mathbf{V}^{XY})$, the point prototypes can be projected onto X and Y exactly as in Figure 4.57. Now, instead of 0-1 rows in the matrix in (4.91), you will have (say) a fuzzy partition of the ducks and llamas,

$$U^X \underset{s}{=} U^Y \underset{s}{=} U^{XY} = \underbrace{\begin{matrix} 0.9 & 0.8 & \cdots & 0.7 & 0.2 & 0.1 & \cdots & 0.4 \\ 0.1 & 0.2 & \cdots & 0.3 & 0.8 & 0.9 & \cdots & 0.6 \end{matrix}}_{\text{soft boundary between ducks and llamas}} . \qquad (4.92)$$

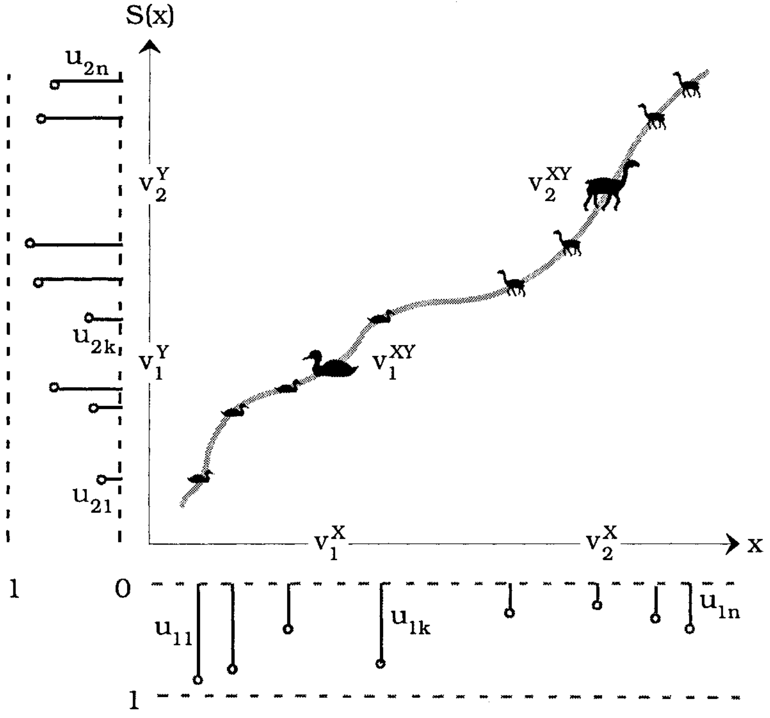


Figure 4.58 The basis of MA point prototype soft rule extraction

At this juncture different authors strike out in many directions. The most straightforward extension of CRE is to project the rows of U^{XY} onto X and Y, leading to the situation illustrated in Figure 4.58 for an SISO system. In Figure 4.58 we show only the projection of the first row of U onto X and the second row of U onto Y. Both values from each column in U can be projected into both spaces. You can visualize the "missing" values in Figure 4.58 by recalling that each column sum in U is 1, so the difference between the value shown and the dashed line marked "1" represents the value not shown for each column. Alternatively, if you imagine rotating, say, the vertical memberships 90 degrees to the right and aligning their "0" axis with the "1" axis of the horizontal memberships, each membership line will have two components that sum to 1.

We will not repeat Figure 4.58 for the non-point prototype case, since the only difference between a figure for this case and Figure 4.58 would be the depiction of non-point prototypes (lines, planes, quadratics, etc.) in the product space containing XY. Figure 4.49 shows non-point prototypes this way, but does not depict the values of U like Figure 4.58 does only because Runkler and Bezdek (1998c, 1999) did not use U in the work discussed in Example 4.17.

The problem you now face is what to do with the projected (point or non-point) prototypes and discrete sets of memberships lying along the range of each variable in the input and output domains. In Example 4.17 Runkler and Bezdek (1998c) simply ignored U, and placed premise membership functions with user-selected shapes that satisfied a regularity constraint by the positions of the projected point prototypes (because they used a TS model, CMFs were not needed). No attempt was made to subsequently optimize the PMFs.

Yager and Filev (1994b) use their MCM algorithm to procure c prototypes $\mathbf{V}^{XY} \in \Re^{pq}$ for the MA model, MISO case. For an MISO system with input $\mathbf{x} \in \Re^p$ and output $y \in \Re$, $\mathbf{V}^{XY}$ is converted into the fuzzy rule: If "$\mathbf{x}$ is CLOSE to $\mathbf{v}_i^{\mathbf{x}}$" then "y is CLOSE to v_i^y". Now defining the fuzzy sets m_i = CLOSE to $\mathbf{v}_i^{\mathbf{x}}$ and mo_i = CLOSE to v_i^y, we get the rules : If $m_i(\mathbf{x})$ then $mo_i(y)$; $i = 1,\ldots,c$. Each antecedent clause is translated into p atomic clauses : x_k is m_{ik}; $k = 1,\ldots,p$. Gaussian-like membership functions are used for the PMFs and CMFs : $m_{ij}(x_k) = \exp^{-\left((x_k - v_{ij}^{\mathbf{x}})^2/2\sigma_{ij}^2\right)}$ and $mo_i(y) = \exp^{-\left((y - v_i^y)^2/2\sigma^2\right)}$. σ_{ij} is the spread of the j-th antecedent of the i-th rule and σ is the spread of all of the consequents.

Yager and Filev (1994b) used the height method of defuzzification for the MA model (equivalently, the TS model with zero-th order functions for the consequents). Initial estimates of the parameters σ_{ij} are taken as $\sqrt{1/2\beta}$, where β is one of the MCM parameters in Table 4.16. All parameters of the system $(v_{ij}^{\mathbf{x}}, v_i^{y}, \sigma_{ij})$ are then further tuned with gradient descent to minimize the total squared error $\sum_{k=1}^{n}\left\|y_k - S(\mathbf{x}_k)\right\|^2$. Although MCM determines the number of prototypes (and hence rules) automatically, control of c is implicit in the parameters of the MCM potential function and the threshold value used to stop the process. Thus, an inappropriate choice of these parameters may over-determine or under-determine the number of rules. U is not used, of course, because MCM does not produce one. The approach in Chiu (1994) is very similar, differing principally in the use of SCM instead of MCM.

Some authors abandon the PMF structure of the LHS of $\mathcal{R}$ shown in Figure 4.32 altogether, opting instead for a much simpler scheme in which the firing strength $\alpha_i(\mathbf{x})$ of R_i in (4.72) is replaced by some presumably reasonable function of $\mathbf{x}$. Specifically, a simplified form of (4.72a) is used:

$$R_i\colon\ \psi_i(\mathbf{x})\ ;\ 1 \le i \le c\ (= M) \qquad . \tag{4.93}$$

In (4.93) the functions $\{\psi_i\}$ are usually interpreted as membership functions for clusters in the input space $\Re^p$, and indeed, are often found or defined this way. The fuzzy system defined using (4.93) instead of (4.72) is not the LHS of a proper MA or TS model, so we call the resultant fuzzy system a *hybrid MA* or *hybrid TS* system, respectively, according as the RHS of $\mathcal{R}$ is configured in an MA or TS fashion.

Abandoning the PMF structure disables linguistic interpretation of the rules, effectively skipping much of the bother and computational complexity (and arguably, some of the utility) of finding linguistic termsets and using approximate reasoning to find $\alpha_i(\mathbf{x})$. To see how authors use this idea, it is convenient to have a slightly different notation for the function in (2.7a) defined by the first order necessary condition required of U for local extrema of the FCM objective function. For any $\mathbf{x}, \mathbf{v}_i \in \Re^p, \mathbf{x} \ne \mathbf{v}_i, i = 1,\ldots,c, \mathbf{V} = \{\mathbf{v}_i\}$, any inner product induced A-norm $\|\mathbf{x}\|_A^2 = \mathbf{x}^T A\mathbf{x}$, and m > 1 we let

$$\psi_i(\mathbf{x}) = \mathrm{FCM}_i(\mathbf{x}, \mathbf{V}) \equiv \left(\sum_{j=1}^{c} \left(\frac{\|\mathbf{x} - \mathbf{v}_i\|_A}{\|\mathbf{x} - \mathbf{v}_j\|_A} \right)^{\frac{2}{m-1}} \right)^{-1}, \quad 1 \le i \le c. \tag{4.94a}$$

Another popular choice for ψ_i in (4.93) is an exponential function in p variables centered at $\mathbf{v}_i$,

$$\psi(\mathbf{x}) = \mathrm{EXP}_i(\mathbf{x}, \mathbf{v}_i) \equiv e^{-\frac{1}{2}\left(\|\mathbf{x}-\mathbf{v}_i\|_A^2\right)} \quad, 1 \le i \le c \quad . \tag{4.94b}$$

Values of (4.94a) and (4.94b) lie in (0, 1), and both are maximum when $\mathbf{x} = \mathbf{v}_i$ (for this to be true for (4.94a) it is necessary to define $\mathrm{FCM}_i(\mathbf{x}, \mathbf{V}) \doteq 1 \Leftrightarrow \mathbf{x} = \mathbf{v}_i$). The shapes of these two functions as continuous variables of $\mathbf{x}$ for fixed $\mathbf{V}$ can be very different because FCM_i depends on the location of all c prototypes, whereas EXP_i is always radially symmetric in $\mathbf{x}$ about $\mathbf{v}_i$. Moreover, (4.94a) is not generally unimodal, whereas (4.94b) has but one maximum. This important consideration has been largely ignored in rule extraction by clustering (see Runkler and Bezdek (1999) for an extended discussion of this topic).

Sin and deFigueiredo (1993) consider only *hybrid* SISO TS models. They use FCM to cluster in XY, and the XB index V_{XB} at (2.102) to select an optimal number of clusters. With A the identity in (4.94a) rule i is : IF $\mathrm{FCM}_i(\mathbf{z}_i(\mathbf{x}), \mathbf{V}^{XY})$ THEN $u_i(\mathbf{x})$, where $\mathbf{z}_i(\mathbf{x})$ depends on both the input $\mathbf{x}$ and the output $u_i(\mathbf{x})$. The CMFs $\{u_i(\mathbf{x})\}$ are then *estimated* by minimizing $E_i = \sum_{k=1}^{n} u_{ik}^{XY}\left(u_i(\mathbf{x}_k) - y_k\right)^2$ for i =1, 2,..., c. Please be careful about our confusing use of u here; u_{ik} is the ik-th entry of U, while $u_i(\mathbf{x})$ is the output function for the i-th rule in $\mathcal{R}$. The role of U^{XY} is limited to using its i-th row as weights of E_i. Sin and deFigueiredo suggest that the output functions might be represented, for example, by training c feed-forward neural networks with XY. The only example they give, however, uses the psuedoinverse method to find the least squared error solution for the coefficients of a first order RHS.

For generalization, given $\mathbf{x} \in \Re^p$, let $\mathbf{z}_i(\mathbf{x}) = \begin{pmatrix} \mathbf{x} \\ u_i(\mathbf{x}) \end{pmatrix}$. The firing strength of rule i is computed as $\alpha_i(\mathbf{x}) = \mathrm{FCM}_i(\mathbf{z}_i(\mathbf{x}), \mathbf{V}^{XY})$; then $\mathbf{S}_{TS}(\mathbf{x})$ is computed with equation (4.73) as usual. Notice that each rule uses a different $\mathbf{z}_i(\mathbf{x})$ to get its firing strength, and that this number depends on both the input to and output of rule R_i.

In one of the most widely ranging papers we know of for SRE, Delgado et al. (1997) offer 6 methods they call EST1, ..., EST6, for constructing both proper and hybrid MISO TS models for function approximation. We briefly review this paper to exemplify just how rich the variety of methods you can choose from really is. Several methods of cluster validity are alluded to, but only the Fukuyama-Sugeno (1989) index is explicitly discussed. EST1-EST3 are 0-th order hybrid models that do not decompose the input space, instead relying on (4.93) for direct estimation of firing strengths.

For EST1 and EST2, FCM is used to cluster XY, and then only $\mathbf{V}^X$ and $\mathbf{V}^Y$ are used. Rule i for EST1 is, using the Euclidean norm in (4.94a) : IF $\mathrm{FCM}_i(\mathbf{x}, \mathbf{V}^X)$ THEN $u_i(\mathbf{x}) = v_i^Y$. For generalization, given $\mathbf{x} \in \Re^p$, compute $\alpha_i(\mathbf{x}) = \mathrm{FCM}_i(\mathbf{x}, \mathbf{V}^X)$ and $u_i(\mathbf{x}) = v_i^Y$, $1 \le i \le c$; then $\mathbf{S}_{TS}(\mathbf{x})$ is computed with equation (4.73). This scheme makes no use of the fuzzy partition U^{XY}. EST2 is EST1 with (4.94a) replaced by (4.94b) and $A = C_i$, the fuzzy covariance matrix for cluster i (from, for example, the GK algorithm).

EST3 applies FCM to XY, uses the terminal fuzzy partition U^{XY} to initialize U^X, and then runs FCM on X alone (at the same value of c). Then the IO data and equation (4.94a) with the Euclidean norm are used with both $\mathbf{V}^{XY}$ and $\mathbf{V}_*^X$ (here $\mathbf{V}_*^X$ is obtained by running FCM on X - it is not the projection of $\mathbf{V}^{XY}$ onto X) to define constants for the output functions of a zero-th order TS model. Specifically, for $1 \le i \le$ c,

$$u_i(\mathbf{x}) = \left(\frac{\sum_{k=1}^{n} \left[\mathrm{FCM}_i(\mathbf{x}_k, \mathbf{V}^X) \cdot \mathrm{FCM}_i((\mathbf{x}_k, y_k), \mathbf{V}^{XY})\right]^m y_k}{\sum_{k=1}^{n} \left[\mathrm{FCM}_i(\mathbf{x}_k, \mathbf{V}^X) \cdot \mathrm{FCM}_i((\mathbf{x}_k, y_k), \mathbf{V}^{XY})\right]^m} \right). \tag{4.95}$$

In (4.95) m is the same weighting exponent that is used for FCM clustering. Again with the Euclidean norm, rule i for EST3 is : IF $\alpha_i(\mathbf{x}) = \mathrm{FCM}_i(\mathbf{x}, \mathbf{V}^X)$ THEN $u_i(\mathbf{x})$ at (4.95). For generalization, given $\mathbf{x} \in \Re^p$, compute $\alpha_i(\mathbf{x}) = \mathrm{FCM}_i(\mathbf{x}, \mathbf{V}^X)$ and $u_i(\mathbf{x}) = v_i^Y$ (by (4.95)), $1 \le i \le c$; $\mathbf{S}_{TS}(\mathbf{x})$ is computed with equation (4.73).

Method EST4 in Delgado et al. (1997) uses an approach to design a 0-th order hybrid TS model that is quite unlike any of the methods reviewed so far. In this scheme X and Y are clustered *separately* into, say c_i and c_o clusters, where c_i and c_o are not necessarily equal, but are chosen by one of a number of different validation strategies which are enumerated in Delgado et al. only through references. Under this plan the rule-base can have $c_i \cdot c_o$ rules, and each rule is assigned a weight w_{ij} that is called the certainty of the rule that relates cluster i in X to cluster j in Y. Also mentioned are one set of input membership functions $\{m_i(\mathbf{x})\}$ defined on $\Re^p$ and a set of constant output functions, $\mathbf{V}^Y$, that are found by clustering in Y. Delgado et al. do not specify what clustering model is used to find these parameters.

If FCM is the clustering model that produces $\mathbf{V}^X$ and $\mathbf{V}^Y$ in X and Y, respectively, and $\psi_i(\mathbf{x})$ is computed with (4.94a) or (4.94b), rule i takes the general form : if $\alpha_i(\mathbf{x}) = \text{FCM}_i(\mathbf{x}, \mathbf{V}^X)$ then $u_j(\mathbf{x}) = v_j^Y$ with certainty w_{ij}. Delgado et al. suggest several ways to compute the $\{w_{ij}\}$ from XY and the $\{\psi_i(\mathbf{x})\}$. Finally, a T norm is selected to integrate the information in the weights and input memberships, resulting in the following generalization of (4.73): given $\mathbf{x} \in \Re^p$, compute $\alpha_i(\mathbf{x}) = \text{FCM}_i(\mathbf{x}, \mathbf{V}^X)$ and $u_i(\mathbf{x}) = v_i^Y$, $1 \le i \le c_i$; then EST4 is computed as

$$\text{EST4}(\mathbf{x}) = \frac{\sum_{i=1}^{c_i}\sum_{j=1}^{c_o} T(\psi_i(\mathbf{x}), w_{ij}) \cdot v_j^Y}{\sum_{i=1}^{c_i}\sum_{j=1}^{c_o} T(\psi_i(\mathbf{x}), w_{ij})} \quad . \tag{4.96}$$

Several examples given in Delgado et al. (1997) suggest that EST1-EST3 are somewhat better than EST4, and not enough details are given about EST4 to understand exactly how things are done. Nonetheless, this exhibits several very different approaches to the use of clusters in X and Y to secure rules to approximate **S**.

EST 6 clusters in XY to establish a hybrid 1-st order MISO TS model. FCM is used to produce U^{XY} and $\mathbf{V}^{XY}$, which are then used to initialize the GK clustering algorithm (recall that this is one of the clustering models capable of producing "linear" prototypes from the principal eigenvectors of fuzzy covariance matrices, Section 2.3.A). Outputs of the GK algorithm are then used to initialize the LHSs of the system. Equation (4.94a) is correct in functional form for the GK model, but instead of a fixed A-norm, AO of the GK functional produces estimates of c matrices $\{A_i\}$. Delgado et al. use these in (4.94a) with $\mathbf{V}^X$ from GK to define rule i for EST6 as follows: If $\alpha_i(\mathbf{x}) = \text{FCM}_{A_i}(\mathbf{x}, \mathbf{V}^X)$

Then $u_i(\mathbf{x}) = a_{i0} + \sum_{k=1}^{p} a_{ik}x_k$. The coefficients $\{a_{ik}: 1 \le i \le c; 0 \le k \le p\}$ of the consequent functions are then estimated using recursive least squares, and Delgado et al. state that the GK partition U^{XY} is used during this procedure, but they do not state how. Finally, these authors also state that they used a genetic algorithm to optimize this system with respect to the MSE error it commits on XY, but no details of the GA or how it was used were given.

We have now seen several ways that the values $\{u_{ik}\}$ in fuzzy partitions U found by clustering in X, Y, or XY are used functionally in conjunction with point prototypes. Another approach taken by some authors returns us to Figure 4.58, where the values $\{u_{ik}\}$ are shown as sets of discrete points projected from U^{XY} onto the input and output domains of the fuzzy system. Again, a variety of approaches for using these memberships have been reported in the literature. Figure 4.59 shows a set of projected memberships (like the ones in Figure 4.58) in either an input or output domain of a fuzzy system, and a few of the many ways we might construct membership functions from them.

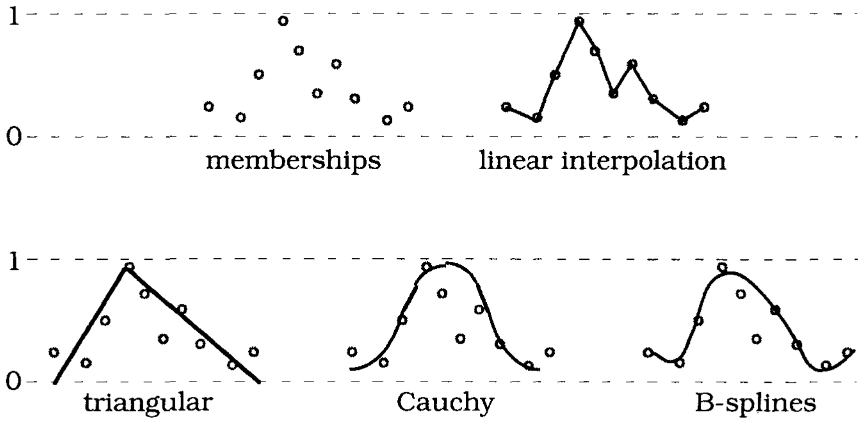


Figure 4.59 Using projected memberships from U to build PMFs

More generally, each row of U_{XY} corresponds to (n values of) a membership function for the i-th cluster. It is possible that "projection" assigns more than one membership value to a feature value. When this happens some type of aggregation operation (usually the maximum) must be used to resolve the conflict and assign a unique membership value to the feature value. When there are p features, the i-th row of U^{XY} will be "projected" onto all p of them for each k = 1 to n, but the shapes of the p membership functions resulting from this operation may be very different because the distributions of values in each feature vary.

Sugeno and Yasukawa (1993) discussed several ways to build membership functions from the values $\{u_{ik}\}$, including piecewise linear interpolation and convex completion as shown in Figure 4.59. If the functions used to fit the projected memberships (such as the ones produced by convex completion) are not smooth, when they are combined with a non smooth T-norm or T-conorm such as the minimum or maximum, the approximation to **S** can be pretty bumpy.

The bottom tier of Figure 4.59 depicts three methods that are smooth approximations to projected memberships in U^{XY}. We can simply erect triangular (or trapezoidal) membership functions (e.g., Sugeno and Yasukawa, 1993, Genther and Glesner, 1994, Klawonn and Kruse, 1997), perhaps centered at the projections of $\mathbf{V}^{XY}$; we can use predefined shapes such as Cauchy or Gaussian functions, again centered at the projections of $\mathbf{V}^{XY}$ (e.g., Chung and Lee, 1997, Runkler and Bezdek, 1999); or we can use a numerical technique such as B-splines or least squares to fit, say a radial basis or cubic function to the memberships (Halgamuge et al., 1995). See Runkler and Bezdek (1998c) for a catalog of other functions that can be used, as well as a unified interface (the ACE membership function toolbar) from which they can be built.

In any case, once we have the membership functions, they can be taken either as the final PMFs (and/or CMFs in an MA model), or as initial estimates that will be subsequently tuned using a technique such as gradient descent (Yager and Filev, 1994b, Chiu, 1994), or genetic algorithms (Delgado et al., 1997). With a little thought, you can invent a new way to use this information too. The point is, (U^{XY}, $\mathbf{B}^{XY}$) carries a lot of information that can be used, but may not be trusted with absolute confidence. Why not? That's the last question on our list, and the easiest one to answer.

Finally, *what might go wrong* when clustering is used for rule extraction? Chapters 2 and 3 contain only a fraction of the clustering algorithms you can use to extract rules from data for fuzzy systems. But even this fraction is fraught with peril for the unexperienced user in both pattern recognition, where you really just want the clusters, and here, where you are using clustering as a tool for building fuzzy systems. The biggest danger all of us face in either application? Clustering algorithms WILL produce clusters (i.e., partitions) - that's their job - whether the data possess any or not. Perhaps approximation of functions by fuzzy systems is an even bigger danger. We'll leave you hanging on this unsettling note, and return to this thought in Section 4.11. Here's a hint to whet your appetite : our coverage of this topic amounts to surveying just a few trees in the ***jungle of function approximation.***

H. Heuristic rule extraction

Structural parameters of rule-based classifiers are not always estimated with training data using decision trees, clustering or whatever else happens to be on your mind at design time. Often the rules are simply defined by the modeler, and the IO data are used for testing and refinement of the subjective design, and possibly for parametric estimation, optimization and validation. While this may sound unscientific, the two examples in this subsection show that the "trial and error" method of rule-base development is alive and well, is sound, and can lead to very effective rules for classification. This section contains two examples of classification performed with fuzzy rule-bases that are developed this way, and which, for lack of a better word, we will call heuristic designs. Both examples use the MA formulation of rule definition and inference structure.

Our first example involves a straightforward classification problem - recognizing two similar chromosomes from features extracted from their images (Keller et al., 1995a, b). The rule-based system discussed in Example 4.20 "locates" a portion of an image of a handwritten address that contains the street number, and is based on the work of Gader et al. (1995a). This is not a traditional use for a classifier, and it shows quite nicely the power and flexibility of fuzzy rule-bases for classification. As you will see, this system uses MA rules.

Example 4.19 Human genetic investigations have provided some of the most dramatic progress in medicine in recent times. One of the standard tools used is karyotyping, a process of visualization and interpretation of chromosomes. This labor-intensive process can yield a large amount of information about a human subject and suspected or potential disease processes. To decrease the labor involved, efforts have been made to automate the process of karyotyping. These efforts have achieved only limited success to date. Successful automation of the karyotyping procedure would have far reaching economic implications. Cost reduction would be significant because of the large number of specimens analyzed each year around the world.

Many pattern recognition approaches have been used to classify isolated chromosomes using features which are either directly or indirectly related to the banding patterns that result when chromosomes from cells in metaphase (the stage before cell division) are stained (Errington and Graham, 1993, Graham and

Piper, 1994, Stanley et al., 1995, 1998). The banding patterns are, in principle, unique to each of the 24 classes of chromosomes in a human cell (homologue pairs of chromosomes numbered 1-22, and either a homologue pair of X chromosomes for a female, or X and Y chromosomes for a male).

Figure 4.60 shows idealized representations (ideograms) and particular examples for two similar chromosome classes (chromosomes within the same "Denver Group", Errington and Graham, 1993). It is difficult to directly match the real chromosomes to the ideograms. The "banding level" is connected to the resolution of the bands in a complete cell image (a metaphase spread). The "400-band level" in Figure 4.60 means that there should be roughly 400 dark bands visible in all 46 chromosomes in the metaphase spread. The "narrow" point of the chromosome is called the centromere, which divides the chromosome into two arms: the P-arm (or short arm) and the Q-arm (or long arm).

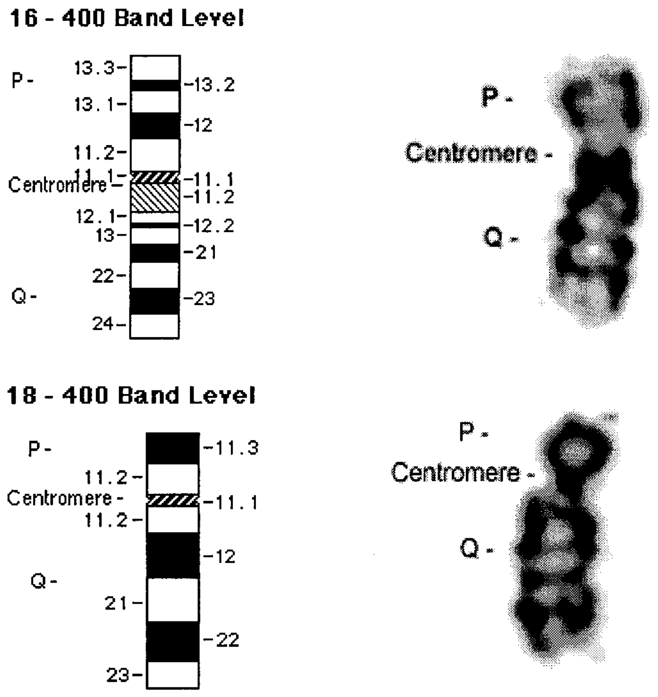


Figure 4.60 Ideograms and examples for chromosomes 16 and 18

Features such as centromeric index (the ratio of the length of the short arm to that of the entire chromosome), relative length, and banding pattern information, including bandwidth, numbers of bands, band spacing, and band intensity can be used with human knowledge to create a rule-based classifier for recognition of these two chromosomes (Keller et al., 1995a, 1995b).

To distinguish chromosome 16 from chromosome 18, rules were developed to generate class confidences directly from *Centromeric Index* (CI) and *Relative Length* (RL). Table 4.37 contains the rules used to determine the class 16 confidence from these measurements (class 18 rules are similar).

Table 4.37 Fuzzy rules for class 16 confidence based on Centromeric Index and Relative Length

CI → ↓ RL	VL	L	M	H	VH
VL	VL	VL	L	M	H
L	VL	VL	L	M	H
M	VL	L	L	H	H
H	VL	L	M	H	VH
VH	M	M	H	VH	VH

Keller et al. used five linguistic values for all their rules: *Very Low* (VL), *Low* (L), *Medium* (M), *High* (H), and *Very High* (VH). Since relative length is less reliable than the centromeric index, its variation has less effect on the consequent than changes in CI. An example of the rules used is:

IF	Centromeric Index is High
AND	Relative Length is Very High
THEN	Chromosome 16 Confidence is Very High

The rules and membership function definitions for the premises and consequents were entered into the CubiCalc RTC fuzzy logic development environment (CubiCalc, 1990). The fuzzy sets described in the rules were heuristically generated by examining the values of the variables on a small training set of 400 band level chromosomes taken from images acquired at Ellis Fischel Cancer Center, University of Missouri-Columbia. For the two chromosome classes 50 rules were generated based on the CI and RL features.

A set of rules involving CI and RL would be sufficient to separate some chromosome classes. However, chromosomes 16 and 18 have similar relative lengths and centromeric indices. So, additional feature information (found in the banding pattern) is needed. The banding pattern is characterized by the number of bands in each arm, relative bandwidths, and relative distances of bands from the centromere. However, it is difficult to correctly segment the bands in real chromosome images, so indirect measurements are often used. Chromosome "blobs" are found in metaphase spreads (not an easy task in itself: Stanley et al., 1995, 1998). The medial axis, or skeleton of each (hopefully) single chromosome is generated by standard image processing techniques such as thinning (Gonzalez and Woods, 1992). The length of the chromosome is then the Euclidean or pixel length of the skeleton. For each point of the skeleton, both the average intensity along each line perpendicular to the skeleton and within the blob (the density profile) and the second moment of those densities along the perpendicular (the shape profile) are calculated (Piper and Granum, 1989).

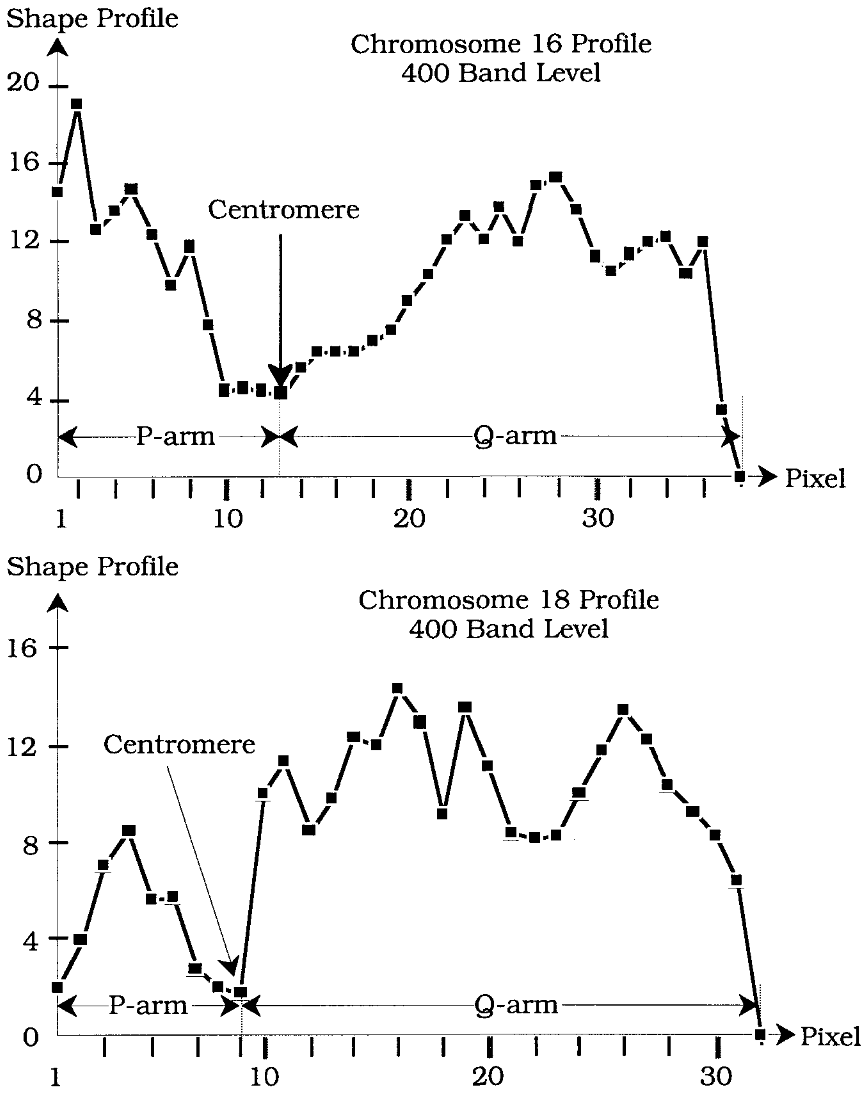


Figure 4.61 Shape profiles for chromosomes 16 and 18

Figure 4.61 shows shape profiles for typical examples of chromosomes 16 and 18 as measured at pixel locations along the axis of generated skeletons. The shape profile contains direct information about the banding patterns.

To extract band-related information, "standard" weighting functions were correlated with the shape profile of each arm of the chromosome under investigation. The weighting functions were designed to match the banding pattern exhibited by the chromosome arm for the specific class. This approach was used because it eliminated the need to segment the bands directly, which could lead to considerable false information by disobeying the principle of least commitment. These features are similar to the "wdd" features employed by Piper and Granum (1989) but they carry more direct evidence about particular class banding patterns (see Keller et al., 1995a for more details on the functions used).

Table 4.38 shows the eight rules generated for one of the three banding pattern correlation values for the p-arm. Keller et al. used similar rules for three band correlation functions, giving a total of 24 rules for class confidence based on shape profile information. The rules and membership functions were heuristically generated. Hence, $\mathcal{R}$ had M = 74 rules for this 2 class problem.

Table 4.38 Class confidence based on the p-arm banding pattern of data file (wd16tbp)

wd16tbp	16 Confidence	18 Confidence
VL	VH	VL
L	H	M
M	M	H
H	L	VH

In a preliminary test, features were extracted from 23 400-band-level chromosome # 16 images and 30 400-band-level chromosome # 18 images in the database. The inference done in Cubicalc was based on the MA model and employed the minimum operator to compute firing strengths, summation as the rule aggregation method, and center of gravity (COG) for defuzzification. By using maximum class confidence as the crisp decision rule, Keller et al. obtained 100% correct classification for # 16 and 87% correct classification for # 18 (resubstitution). By thresholding the difference between chromosome 16 confidence and chromosome 18 confidence, and rejecting chromosomes whose confidence difference was too small, they report 100% (resubstitution) reliability on this small set with a 23% rejection rate. Finally, the confidence values can be used in subsequent processing.

Our second example comes from the field of handwritten address recognition (Gader et al., 1995a). Recognition of handwriting is important for automating document processing functions such as mail sorting and check reading. As we have seen and will see again (Wang and Suen, 1983, 1984, 1987, Chi and Yan, 1995, 1996, Chi et al., 1995, 1996), fuzzy set theory can be an appropriate framework to address several problems in handwriting recognition. Handwritten character and word classes are not crisp sets. Inherent ambiguity exists at several levels, requiring that multiple sources of information be utilized to correctly interpret handwriting. Furthermore, document analysis systems consist of multiple stages of processing: image processing to separate handwriting from background, segmentation to isolate individual regions such as lines, words, and characters, feature extraction to characterize pattern classes, and finally, classification. Each stage of processing contains uncertainty since the algorithms do not always yield the correct result. Therefore, there are two sources of ambiguity in handwriting recognition: the data are inherently ambiguous and the algorithms are imperfect.

Example 4.20 The ambiguity between numerals and alphabetic characters in handwriting is a problem, as shown in Figure 4.62, which contains, for example, an "F" as the first letter of the word "Franklin" that looks like the numeral "7"; and an "I" as the first letter of the word "Ingraham" that can be mistaken for the number "2"; and several number "7" ' s that are very similar to the "F".

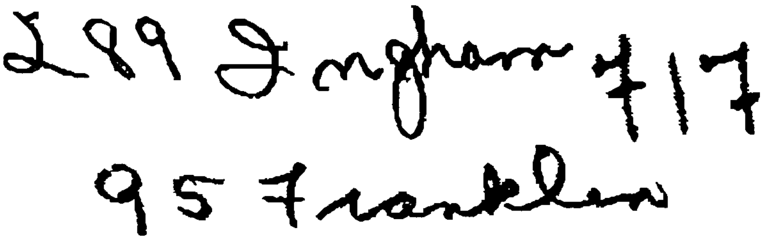


Figure 4.62 Examples of confusing street numbers and letters

Developing an effective interpretation system of handwritten addresses for automation of mail delivery is a challenging task. The numeric fields in an address, i. e., the street numbers and ZIP code, play a crucial role in reducing the complexity of the address interpretation task. If these numeric fields are correctly detected and identified, the number of possible addresses is significantly reduced. Correct location and interpretation of the street number field reduces the number of possible street names. Thus, we must locate the street number without any knowledge of the street name. This is not a "standard" classification problem, since the goal is to find the location where the street number ends (if there *is* a street number in the image). There is the equally important task of recognizing the digits, which in this example was performed by feed-forward neural networks (Section 4.7).

Potential address images were input to the system as described in Gader et al. (1995a). Image processing was used to segment subimages of lines from handwritten addresses into sequences of primitives. Six neural networks were used in the confidence assignment: two for numerals (0-9), and four for alphabetic characters.

Two types of feature vectors were used as inputs to the neural networks, the transition feature vector and the bar feature vector. The bar-features are completely described in (Gillies et al., 1992, Gader et al., 1992, Gader et al., 1995a), while the transition features are described in (Gillies et al., 1992, Gader et al., 1997a, b). The neural networks were trained using backpropagation, and used class-coded outputs. They also contain a class named "garbage" to account for segments which did not represent any character image, such as multiple characters or pieces of characters.

Transition features are the locations and numbers of transitions from white pixels to black pixels along horizontal and vertical lines. Transition calculations are performed from right to left, left to right, top to bottom and bottom to top. This information is encoded as a feature vector with 100 elements. Three neural networks for the confidence measurements used transition feature vectors, one each for upper and lower case alphabetic characters, and one for digit recognition.

The bar features encode directional information from the foreground and the background. First, up to 8 feature images are generated, each corresponding to one of the directions east, northeast, north and northwest, in either the foreground or the background. Each feature image has an integer value at each location that represents the length of the longest bar which can fit at that point in that direction. For each of the 8 feature images, 15 different subimage zones are created. The values in these zones are summed and normalized between 0 and 1. The result is a feature vector of size 120. Three neural networks for the confidence measurements used bar feature vectors, one each for upper and lower case alphabetic characters, and one for digit recognition.

Primitives often contain only parts of characters. To obtain confidence measurements on characters, subimages of pairs and triples of the primitives were also used to obtain character confidence assignments using the neural networks. Hence, there were six character confidence readings and measurements at the end of each primitive, each corresponding to a single primitive, a pair or a triplet of primitives, in either upper or lower case characters. The maximum of these 6 confidence measurements was used as the character confidence feature for the fuzzy rule-based system.

A fuzzy logic system with 48 rules that aggregated results of image processing and character recognition modules to assign confidences concerning the locations of street numbers in address blocks was developed in Gader et al. (1995a). The neural networks described above were used to assign alphanumeric character class confidences to combinations of primitives. Each consecutive combination of primitives starting with the leftmost primitive was assigned a confidence value by the fuzzy rule-base indicating the possibility that the combination represented a complete numeric field, i.e., the potential location marker for the numeric portion of the street address. One example of a rule in this system is:

IF	the next primitive is *too complex* to be recognized as digits,
AND	the numeric field confidence of the current primitive is *large*,
AND	the gap size between the current and the next primitive is *medium*;
THEN	the street number confidence should be *positive large*.

Linguistic values of each linguistic variable were represented by standard trapezoidal membership functions. For example, the membership functions for the linguistic values small, medium, large, and huge are shown in Figure 4.63 for the linguistic variable gap (size). Notice that Large and Huge are both 1 for x close to 1.

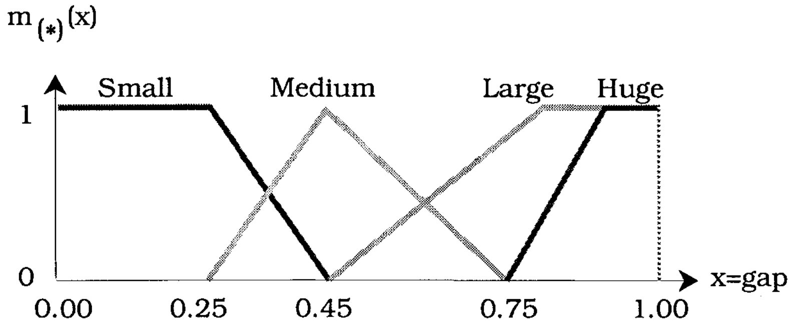


Figure 4.63 Membership functions for the linguistic variable "gap"

Gader et al. followed the usual pattern for the development of heuristic rule-based systems: an iterative cycle of rule definition, testing, and rule refinement. The rules in the fuzzy rule-base were initially written based on pictures of address blocks (SUNY, 1989). The system was then trained with 71 image blocks using the same MA model that was described in Example 4.19, implemented in Cubicalc. The system was trained on the 71 images, each of which was crisply labeled as having or not having a numeric address field, and if present, its location. The training process was iterated until the results were satisfactory. Following each training cycle, the system was adjusted based on an analysis of the results, paying particular attention to the error cases.

After training, 78 new image blocks were used as a test set. A few adjustments were made based on these results. For example, additional rules (such as rules to handle "P.O. Box") were added to the rule-base, and a few rules were changed. The union of the training and test sets was then used as a reference training set, and a blind (validation) test was conducted on 155 additional image blocks. During the blind test, the output confidence value was thresholded. Those locations at which the overall system confidence was above a user specified threshold were labeled as locations of street numbers by the system.

Table 4.39 Success and location error rates for the training and validation sets

Success rate		Location rate	
train	validate	train	validate
91%	86%	91%	87%

Table 4.39 shows the results of the final training run and the blind test of 155 images, 79 of which contain street numbers. The success rate is the percentage of answers that are correct; either an image block contained a street number and it was correctly located or it did not contain a street number and the system indicated no street number. The location rate is the percentage of street numbers that were correctly located.

The performance of this system illustrates the capacity of an MA fuzzy rule-based system for locating street number fields. A wide variety of multi-layer feedforward networks for locating the street numbers were also trained using backpropagation and tested using the same training, test and validation data sets. The numeric input variables used by the fuzzy logic system were also used for the neural networks. The networks performed reasonably well - but not as well as the fuzzy logic system. The best success rate on the test data obtained by any neural network was 79%. The fuzzy rule-base achieved a testing success rate of 86%, which is significantly better than that achieved by any of the neural networks ☺.

Gader et al. (1995a) conjectured that the reason the fuzzy logic system outperformed the optimized neural network was that the granularity of knowledge required to locate street numbers is "coarser" than that required to perform tasks such as character recognition. Tasks that require knowledge about the world that is not statistically represented in the data are difficult or impossible for neural networks to learn, but this type of knowledge can be encoded with rules.

I. Generation of fuzzy labels for training data

Several of the methods discussed so far require soft labels for the training data in order to build the decision or classifier function. This includes the soft k-nn rules, the fuzzy integral, fuzzy decision trees, soft rule - based classifiers, and fuzzy aggregation networks (Section 4.7). This was done as early as 1985, when Keller and Hunt (1985) softened the training of the classical linear perceptron. The assignment of soft labels to training data is an important step in the overall process of classifier design. In this subsection we discuss some methods of assigning fuzzy labels to data in order to generate fuzzy classifier functions.

Most of chapters 2 and 3 deal with generating soft labels for objects represented by feature vectors or relational data. Since clustering is unsupervised in general, it may not be the best choice for labeling training data in this context, since the best clusters are those which minimize some clustering criterion, and the algorithmic clusters found may not reflect the actual "ground truth" available in the physical labels in the training data (but see House et al., 1999 for an example that this is not *always* the case). Clustering algorithms sometimes create membership values for training points that have crisp label i which are larger for some class j≠i. This happens, for example, in the context of 1-np design, if a training point from class i is *closer* to the class j prototype than to that of class i.

To insure that soft training labels maintain the "truth" about the training data (i.e., are consistent with the physical labels supplied with the data), some form of supervision is required. One way to accomplish this in the clustering framework is to cluster the data one class at a time, as, for example, Chiu (1997) recommends when you want to build a fuzzy rule-based classifier using labeled data with clustering. In order to get meaningful "typicality" memberships, you should consider using a possibilistic model such as PCM to acquire the soft labels, since label vectors from any fuzzy clustering algorithm contain values that represent degrees of sharing between classes. PCM produces an inverse distance-type membership for each training point from its class prototype. After finding and correctly labeling class prototypes by any means, inverse distance membership functions can be generated for the training data.

Variations of the k-nn technique have been used to obtain fuzzy labels. Just taking the fraction of the number of class i vectors in the k nearest neighbors to a training point as its membership in class i suffers from the same problem as fuzzy clustering. There is no way to guarantee that the "true" class of a training point will maintain the largest membership. For example, consider the "F" in the word "Franklin" in Figure 4.62. It is possible that for a given training set, in feature space all of the neighbors of the feature vector from that "F" would be vectors from the character "7", because this "F" really looks like a "7". Hence, simple fractions would give this feature vector zero membership in the class "F", even though it actually was an "F" written by some person. This is one dilemma you have when dealing with real data: even though an object may actually be member of class i, its feature vector often mingles with those of other classes. Example: build a classifier that identifies men and women based on the 2D input feature vector height and weight. No classifier we are aware of would, based on this pair of measurements, correctly label Heidi Gillingham, who at one time was a center for the Vanderbilt women's basketball team, height = 6 feet, 11.5 inches. If you were to create a soft label vector for Heidi, its maximum value would almost certainly interchange the correct label with the wrong one.

One clever but simple method to acknowledge this uncertainty in the training data was developed by Gader et al. (1995c), and could be called a *possibilistic k-nn labeling procedure.* For a training vector from class i, their approach was to use the fraction of the k nearest neighbors to be the memberships for all classes j $\neq$ i, and to preserve unit (or at least very high) membership in the true class i. This way, the "F" may have high memberships in multiple classes, reflecting the ambiguity of its feature vector. As an example, the "F" in Figure 4.64 received high membership in its class, but also reasonably high values for "I", "L", "S" and "T". In Gader et al. (1995b), these possibilistic training labels were used as desired outputs for a multilayer perceptron. What was discovered was that in terms of pure character recognition, crisp labels worked better, but when the results of the character recognizer were submitted to a word recognition system (Gader et al., 1995b), word recognition rates increased significantly when using the possibilistic labels. By acknowledging the ambiguity of handwritten characters, the total system could keep multiple hypotheses alive and hence, piece the words together more effectively. This adheres to the principle of least commitment.

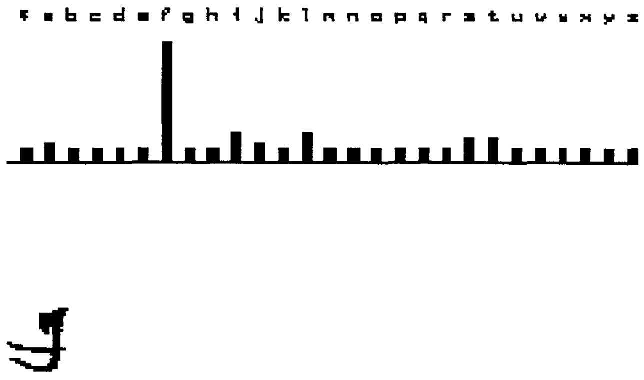


Figure 4.64 A handwritten training "F" and its possibilistic memberships in the character classes

If enough training data are available (as in some image processing applications), normalized histograms of the feature data can be used to estimate class memberships. While this approach is most often used to calculate class confidence with respect to one feature, it can easily be extended by constructing histograms of each of the various features, and the individual feature memberships can be aggregated to get combined memberships both for the training and test data.

Generating soft labels for crisply labeled training data is tricky, and it is very problem dependent. If the goal of a classifier is character recognition, then the evidence in Gader et al. (1992, 1995b) suggests that crisp training labels are better, However, as the goal (and the processing) became more complex, e.g., word recognition, fuzzy, probabilistic or possibilistic labels may provide more realistic information and better final results. You should use the simplest tool to solve your problem. As system complexity grows, tools such as employing soft labels for the training data become more attractive. In fact, uncertainty is always fruitful - as long as you try to understand it too.

4.7 Neural-like architectures for classification

Much has been written in the last twenty years or so about "*neural networks*", a term we abbreviate by "NN" - recall that we use (nn) for *nearest neighbors*. Network architectures such as the MA and TS fuzzy systems and fuzzy decision trees do not draw their original inspiration from a desire to mimic *biological NNs* (BNNs) - although they do have desirable properties such as parallelism which can of course be associated with the BNN. However, several of the models discussed in this section are in some rudimentary sense (neural-like) network structures that do attempt to imitate BNNs. Other volumes in this handbook contain good descriptions of many neural-like network structures, especially for control (Nguyen and Sugeno, 1998). Our presentation is limited to those fuzzy set-related NN models that seem particularly useful for pattern recognition.

Most authors distinguish between BNNs and computational NNs by calling computational structures aimed at imitating BNNs *Artificial NNs* (ANNs). A few authors have made a further distinction between ANNs and *computational NNs* (CNNs). Fine distinctions about the meaning of various terms used in this field (if they have useful meanings at all!) simply distract readers from the main point, which is the interface between NN models and fuzzy logic as used for numerical pattern recognition. Readers interested in this aspect of NNs, including discussions about "computational intelligence" and "soft computing" are encouraged to consult, e.g., Bezdek (1992, 1998), Marks (1993), or Zurada et al. (1994). We will use NN for computational approaches that mimic the BNN, and leave it at that.

There are two distinct areas of integration between fuzzy pattern recognition and NNs. First, we may use the conventional NN for a variety of computational tasks within the larger framework of a pre-existing fuzzy model. In this category, for example, are attempts to build [membership] function representations with NNs; implementation of fuzzy logic operations such as union (max-nets), intersection (min-nets), and even fuzzy logic inference. There is also a great amount of current effort being expended in using NNs to derive optimal rule sets for fuzzy controllers (another approach to rule extraction) - that is, to automate the process of membership function extraction and tuning of term sets that are used in both fuzzy pattern recognition and control.

On the other hand, many writers are investigating ways and means of building "fuzzy NNs", by incorporating the notion of fuzziness *onto* or *into* a NN framework (as opposed to using the NN within a fuzzy framework). For example, the target outputs of the NN during classifier training can be fuzzy label vectors (points in the interior of the triangle N_{fc} shown in Figure 1.2). In this case, the NN itself is implicitly functioning as a fuzzy classifier, and is conceptually identical to any other fuzzy classifier function $\mathbf{D}$ imaged in N_{fc}. Operationally, of course, the mathematical function $\mathbf{D}$ is implicitly represented by an explicit computer program or piece of hardware that implements the NN.

Another way to incorporate fuzziness into a standard NN is by altering the integrator/transfer functions at each node so that they perform fuzzy aggregation (union, weighted mean, or intersection) on the numerical information arriving at each node. Yet another way to introduce fuzziness into the NN framework is through the input data X to the NN, which may be "fuzzified" in one of several ways.

A. Biological and mathematical neuron models

The BNN is one of the systems that enables organisms (in particular, humans) to perform biological pattern recognition. Figure 4.65 depicts the simplest ideas we have about the atomic unit - a neuron - of a BNN. Each neuron has an axon (pulse transmitter), soma (pulse emitter), dendrites (pulse receptors), and is connected to other neurons by synapses (connectors). In Figure 4.65 a packet of data (electro-chemical pulse $\mathbf{x}_k$) has been emitted by the soma, and is traveling along the axon.

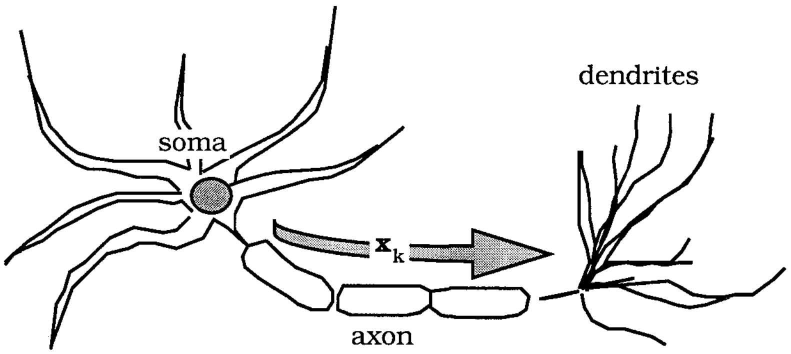


Figure 4.65 A rudimentary biological neuron

Figure 4.66 depicts part of a BNN. The term network derives from the interconnections (which may not be entirely physical) between neurons. At a point of data transfer, a synapse (the connection neighborhood) transmits data packet $\mathbf{x}_j$ from a dendrite of the emitting neuron to a receptor of the receiving neuron. It is believed that each transfer encounters variable resistance (modeled in Figure 4.66 by a synaptic weight vector $\mathbf{w}_j$) to the conduction of energy. Information (electrical, chemical, biological in form) is generated, flows, is assimilated, and somehow used to solve problems in the BNN. Our *assumption* is that each neuron does *something like* (numerical)computing - this gives rise to the hope that computational "neurons" and networks of them can be used to imitate this structure and its performance.

The synaptic weights at a node in the BNN are believed to vary over time, and it is assumed that this is one of the major mechanisms by which the brain "adapts" to changes in its environment (i.e., to changes in its input data and/or output requirements). Another means for achieving adaptation to system tasks is thought to be through the activation and deactivation of (sets of) nodes in the network, again "on the fly". That the brain can and does adapt in real time is inarguable - it is the mechanisms for doing so that are not well understood.

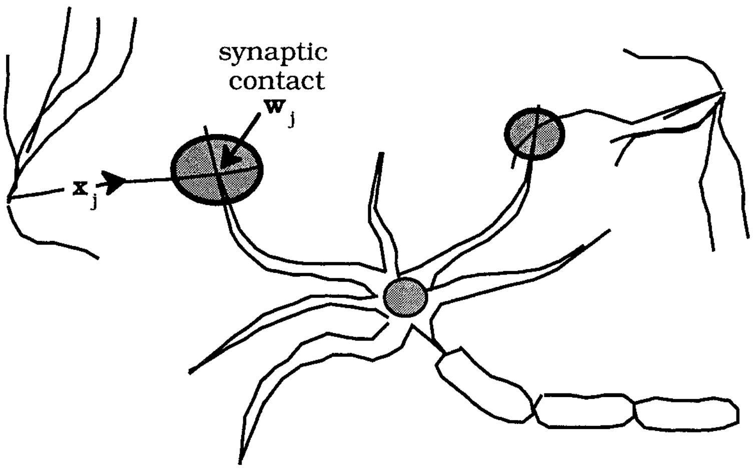


Figure 4.66 Part of a biological neural network

Hardware implementations of computational NNs have become common (e.g., Serrano-Gotarredona et al., 1998). Many companies now market "NNs on a chip", including products advertised as "fuzzy NN chips", etc. If you are interested in this aspect of NNs see issue 4(4) of the 1996 *IEEE Transactions on Fuzzy Systems.* It is not our purpose to pursue this topic, so we are content to show Figure 4.67, which illustrates the components of a typical electro-mechanical (or possibly optical) version of Figure 4.66, that is, a layout of (hardware and/or software) components in an architecture that ostensibly mirrors the biological version of one neuron.

Components of the input and weight *vectors* $\mathbf{x}_j$ and $\mathbf{w}_j$ of Figure 4.66 are shown as real numbers in Figure 4.67; as usual, we assume that $\mathbf{x}_j, \mathbf{w}_j \in \Re^p$. The analog of the soma is called a *node* of the network, $\mathbf{x}_j$ is the *input* to the node, and $\mathbf{w}_j$ is the *weight vector* for (or at, or in) the node. Some writers prefer to regard the scalars $\{w_{ij} : i = 1, \ldots, p\}$ as *weights on the edges* entering the node, while others regard $\mathbf{w}_j$ as a weight vector *attached to* the node. We use one or the other of these interpretations at various times. The node in Figure 4.67 is called N_j - the jth node in the network.

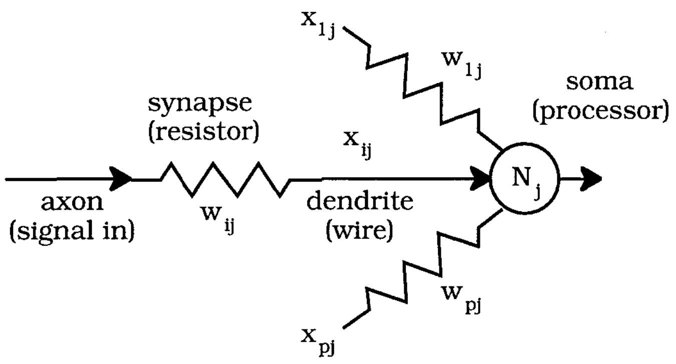


Figure 4.67 An electrical circuit that models the standard neuron

Figure 4.68 illustrates what usually happens at each node in the NN. Two functions are active. First an *integrator function* $f: \Re^{p'} \times \Re^{q'} \mapsto \Re$ combines a *node weight vector* $\boldsymbol{\omega} \in \Re^{q'}$ with the input vector $\mathbf{x} \in \Re^{p'}$; often, but not always, $\boldsymbol{\omega} = \mathbf{w}$, the weight vector shown in Figure 4.67. Notice that the dimensions of $\mathbf{x}$ and $\boldsymbol{\omega}$ are, respectively, p' and q'. Usually $p' = q'$, but these dimensions can and do change from layer to layer, so we leave the notation flexible. We use primes here to indicate that this neuron can be anywhere in the network (the input vector to the network will always be in $\Re^p$, and the output vector from the network will be in $\Re^q$.

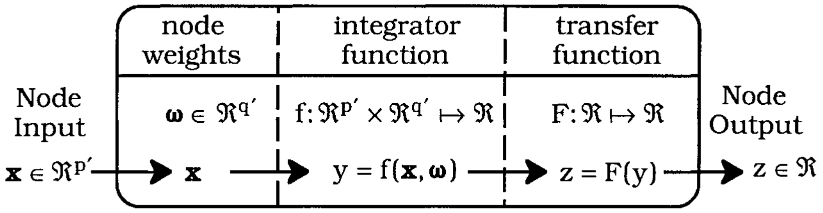

node weights
integrator function
transfer function
Node Input
$\boldsymbol{\omega} \in \Re^{q'}$
$f: \Re^{p'} \times \Re^{q'} \mapsto \Re$
$F: \Re \mapsto \Re$
Node Output
$\mathbf{x} \in \Re^{p'}$
$\mathbf{x}$
$y = f(\mathbf{x}, \boldsymbol{\omega})$
$z = F(y)$
$z \in \Re$

Figure 4.68 A mathematical neuron

The traditional, historically first, and still most popular choice for f is the Euclidean inner product (McCulloch and Pitts, 1943); when $\boldsymbol{\omega} = \mathbf{w} \in \Re^p$, $y = y(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \langle \mathbf{x}, \mathbf{w} \rangle + \alpha$. Recalling equation (4.3) and Figure 4.4, we see that this choice sets up a hyperplane H in $\Re^p$ at each computing node where it is used, and we call f a *linear integrator function.* Justification for this terminology lies in the fact that every affine function on $\Re^p$ can be written as a linear function on $\Re^{p+1}$ by defining the p+1 tuples $\mathbf{x}' = (x_1, \ldots, x_p, 1)^T$ and

$\mathbf{w}' = (w_1, \ldots, w_p, \alpha)^T$, for which $\langle \mathbf{x}', \mathbf{w}' \rangle = \langle \mathbf{x}, \mathbf{w} \rangle + \alpha$. The parameters $\mathbf{w}' = (\mathbf{w}, \alpha)$ of H are (part of) the weights that are sought during training for each node of the network that uses linear integrator functions. In the neural networks literature α is often called the *offset* or *bias* of such a node, and the node itself may be called a *first order neuron.* It will be convenient to have a special notation for this oft-used integrator function; we call it f_H, the subscript referring to the hyperplane that it defines.

"Higher order" neurons arise when the inner product is replaced by a more complicated function. For example, a *second order neuron* is realized by replacing $f(\mathbf{x}, \mathbf{w}) = \langle \mathbf{x}, \mathbf{w} \rangle + \alpha$ with a quadratic form in $\mathbf{x}$, $f(\mathbf{x}, \mathbf{w}, W) = \mathbf{x}^T W \mathbf{x} + \mathbf{x}^T \mathbf{w} + \alpha$, where W is a $p \times p$ matrix of additional weights that are associated with f. In this case the weights of the integrator function f are the triple of parameters $\boldsymbol{\omega}$ = (W, $\mathbf{w}$, α). The form of f is limited only by your imagination. You will encounter many substitutes for these simple functions in the literature of "fuzzy NNs".

The action of f is followed locally in each computing node by applying a *transfer (or activation) function* F to the value of the integrator function on its inputs. F is used to decide if the node should "fire" (produce an output), and if so, how much "charge", and of what sign, should be broadcast as output in response to the input $\mathbf{x}$. The most typical choice for F is the unipolar *logistic* (sigmoidal, squashing) function,

$$F_L(y) = \frac{1}{1 + e^{-\lambda(y-\beta)}} \quad , \tag{4.97}$$

where λ in $\Re^+$ and β in $\Re$ are real constants that adjust the shape of F. Specifically, λ controls the steepness or slope of F_L and β controls the crossover point along the y axis at which inflection occurs, viz., $F_L(y) = 0.5 \Leftrightarrow y = \beta$. Without loss of generality we discuss F_L for β = 0, since this parameter simply shifts F_L to the left or right of the origin. F_L is called a *unipolar* activation function because its range is (0, 1). Figure 4.69 depicts F_L for three choices of the steepness parameter λ with β = 0 for y in [-5, 5].

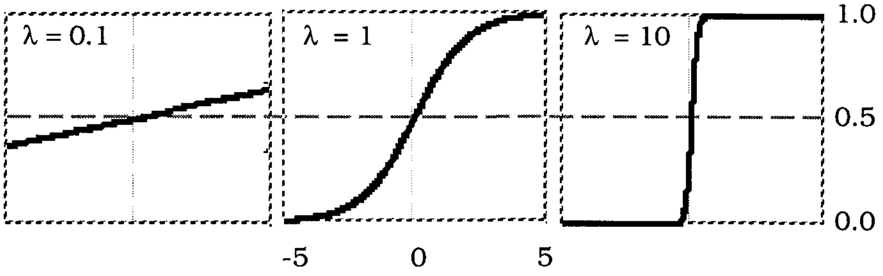


Figure 4.69 Effect of steepness parameter λ on F_L

At $\lambda = 0$ the graph of F_L is the horizontal line y = 0.5. As λ increases, the shape of F_L becomes more and more like the step function which jumps from 0 to 1 (which are the asymptotes of F_L as $y \to \pm\infty$) as λ approaches ∞. The linear transformation

$$F_{L,bi}(y) = 2F_L(y) - 1 = \left(\frac{2}{1+e^{-\lambda(y-\beta)}}\right) - 1 \quad , \qquad (4.98)$$

of (4.97) is called the *bipolar* form of the logistic function because its range is (-1, 1), with limits ±1 as λ approaches ∞, the sign depending on y. That is, the limit of $F_{L,bi}$ with $\beta = 0$ is just the sign function,

$$\underset{\lambda \to \infty}{\underline{\lim}}\left\{F_{L,bi}(y)\right\} = \operatorname{sgn}(y) = \begin{Bmatrix} 1; & y > 0 \\ -1; & y < 0 \end{Bmatrix} \quad . \qquad (4.99)$$

Another function that can be used as a transfer function which has the same basic properties as $F_{L,bi}$ is the hyperbolic tangent, $F(y) = \lambda \tanh(\beta y)$. There are many other transfer functions in the fuzzy NN literature; we will meet some of them later in this chapter.

Now combining the action of the integrator and transfer functions, consider the composition of f followed by F. We call this the *node function* $\Phi = F \circ f$, whose job is to convert vector inputs to a single node into real outputs, $z = \Phi(\mathbf{x}) = F \circ f(\mathbf{x}) = F(y)$. When the integrator function is f_H and the transfer function is F_L (unipolar) we write $\Phi_{LH} = F_L \circ f_H$, and we call it the *standard* or *McPitts* (after McCulloch and Pitts, 1943) *neuron.*

We have already met the idea of node functions, which in section 4.6 were associated with the nodes of a decision tree classifier. It is entirely proper to regard those node functions in the same light as the ones currently under discussion. Both types make decisions about what values "travel" along paths in the network. One of the main differences in the two network structures is that in decision trees there is but one input edge per node, and usually many output edges with different values; whereas most neuron models have nodes with many inputs, and only one distinct output (that may go many places, but has the same value on each outgoing edge). In Section 4.11 we will discuss the equivalence between some special classes of decision trees and certain types of neural networks.

Decomposition of Φ into its two components enables us to analyze the mathematics of one node more carefully, and is very helpful in understanding the relationship of NN methods to other classifier designs. Imagine that you can rotate a hyperplane H so that it stands vertically, parallel to the vertical z-axis, and you are standing at infinity, looking down along H towards the origin of the horizontal axis. Figure 4.70 shows an "end-on" view of what you would see if you could position yourself at the "edge" of the hyperplane H in Figure 4.4 that is created by f_H. Then superimpose the action of transfer function F_L with $\lambda = 1$ and $\beta = 0$ on [-5, 5]) onto your field of view.

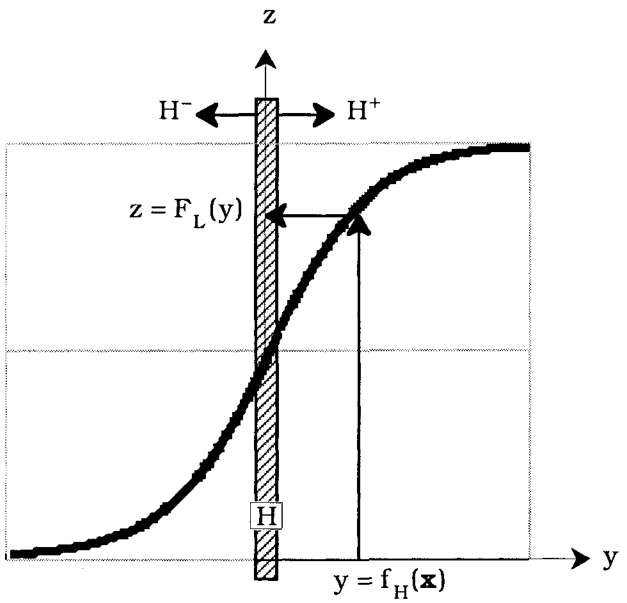


Figure 4.70 Geometric interpretation of node function $\Phi_{LH} = F_L \circ f_H$

In this different view of the geometric meaning of the linear integrator function f_H, you would see the half spaces H^- and H^+ to the left and right of H. The logistic function provides a non-linear response to node inputs that fall on either side of H. Since F_L takes values in the open interval (0, 1), you might be tempted to interpret them as memberships, and in the proper linguistic framework, this could certainly be a membership function for some linguistic value.

But, under these circumstances, would you call this a "fuzzy neuron"? *We think not.* We will encounter instances of fuzzy models that discuss node computations in terms of Φ rather than $F \circ f$, and we will look carefully for the added value provided by fuzzification of the node function, or of its constituents, the integrator and transfer functions.

B. Neural network models

The definition of the computational NN given in DARPA (1988, p. 60) is: "a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes". The network structure (or topology) refers to the way the nodes are connected to each other; the connection strengths are the weight vectors $\{\mathbf{\omega}_j\}$; and node processing refers to *local* computations done by Φ at any node in the network.

Figure 4.71 shows a general NN architecture, with no particular assumptions made about the node functions that are used. The network topology in Figure 4.71 has feed-forward, feed-backward and cyclic connections between and among its nodes. Most NN models used in pattern recognition are feed-forward only, a simplification that seems necessary for both computational and analytical tractability.

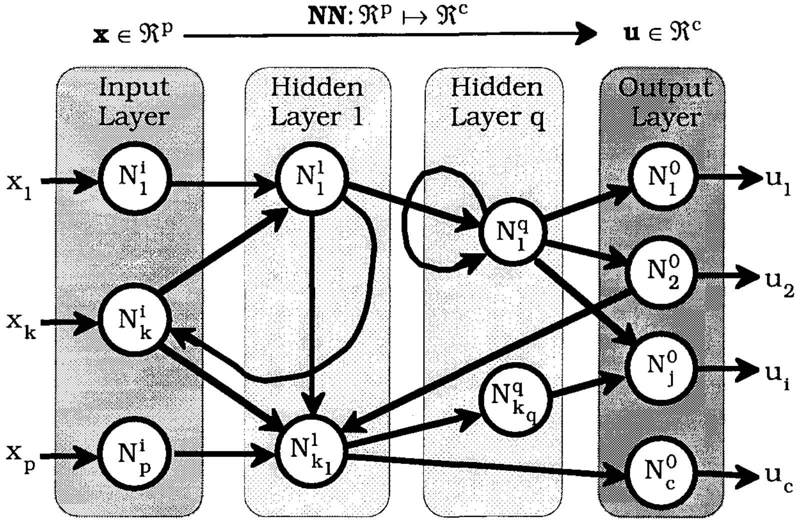


Figure 4.71 A computational neural network

See Table 1 in Hecht-Nielsen (1988) for an early list of the thirteen (supposedly, in 1988) most common NNs as well as a tabulation of neurocomputers for each model built as of that date. The most complete current listing of NN architectures and software is perhaps the *Handbook of Neural Computation* (Fiesler and Beale, 1997). A good recent list of hardware implementations of NN architectures is given in Chapter 27 of Chen (1996).

Most NNs have *layers*. In Figure 4.71 there is an *input layer*, whose nodes $\{N_j^i : j = 1,\ldots,p\}$ almost always "perform" no computations. The purpose of the input layer is to indicate how the data enters the network in p parallel input streams, and to show how the input features are distributed to the *first hidden layer*, whose nodes are indicated by the notation $\{N_j^1 : j = 1,\ldots,k_1\}$. Hidden from what you may ask? Hidden from the input and output layers, so we are told. Integer k_1 is the number of nodes in the first hidden layer. The qth *hidden layer* has k_q nodes $\{N_j^q : j = 1,\ldots,k_q\}$, etc., and the nodes of the *output layer* are $\{N_j^o : j = 1,\ldots,c\}$. Thus, superscripts indicate the layer, and subscripts indicate the node number within each layer.

The output layer usually has computations at each node. When we discuss a general NN, we may omit the superscripts and speak about node N_i for simplicity. We call the hidden and output layers in a NN the *computing layers* of the NN. The architecture in Figure 4.71 is symbolically denoted by the sequence of numbers representing the number of nodes in each layer as $p : k_1 : \cdots : k_q : c$.

If the *last* functions applied to values flowing through the output layer in Figure 4.71 are of the form (4.97), the output of the NN is a possibilistic label vector, $\mathbf{u} = (u_1,\ldots,u_c)^T \in N_{pc}$ (see equation (1.1)). This hardly justifies calling such a network a possibilistic NN, so don't be tempted to interpret it that way unless there is enough semantic justification to entitle the network to this descriptor.

It is convenient to have a notation for the set of *all* parameters of a NN that must be "learned" (acquired during training). For example, if a node has linear integrator and logistic functions, $\Phi_{LH} = F_L \circ f_H$, then the ith node weight vector has the form $\boldsymbol{\omega}_i = (\lambda_i, \beta_i, \mathbf{w}_i, \alpha_i)$. When the total number of nodes in the network is N, we call $\mathbf{W} = (\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_N)$ the *network weight vector*. For example, if there are 8 input nodes, 2 hidden layers with 3 and 5 nodes, respectively, followed by an output layer with 4 nodes, i.e., an 8:3:5:4 architecture, then there are $(8+3)\cdot 3 + (3+3)\cdot 5 + (5+3)\cdot 4 =$ 95 parameters to learn assuming feed forward connections only.

The cardinality, $|\mathbf{W}|$ of $\mathbf{W}$, which is the number of parameters to estimate, is important, because it influences the size of minimally acceptable training sets. Theoretical guidance for network size (as measured by $|\mathbf{W}|$) for a given set of IO data is limited to cases where very idealized assumptions are made about the training data. For example, Baum and Haussler (1989) give the bound

$$\left(\frac{32\cdot|\mathbf{W}|}{E_{\mathbf{D}}(X_{te}|X_{tr})}\right)\log_e\left(\frac{32\cdot M}{E_{\mathbf{D}}(X_{te}|X_{tr})}\right)\le n_{tr}=|X_{tr}| \quad \text{, where} \qquad (4.100)$$

M is the total number of hidden nodes in a single layer and $E_{\mathbf{D}}(X_{te}|X_{tr})$ is the *desired* fraction of errors that you will tolerate on the test set. They assert that a single layer neural network with bipolar output nodes will "almost certainly" generalize [to $E_{\mathbf{D}}(X_{te}|X_{tr})$ on similar input data] if the fraction of errors committed on the training data is less than half of the desired test error, $E_{\mathbf{D}}(X_{tr}|X_{tr})\le E_{\mathbf{D}}(X_{te}|X_{tr})/2$, and (4.100) is satisfied. Haykin (1994) calls (4.100) a distribution-free worst-case bound on the size of the training data. Ignoring the logarithmic term and the multiplier 32 in (4.100) gives the simpler first order estimate,

$$\frac{|\mathbf{W}|}{E_{\mathbf{D}}(X_{te}|X_{tr})}<n_{tr} \quad , \qquad (4.101)$$

which Haykin (1994) asserts is a good rule of thumb in practice. Equations (4.100) and (4.101) bound the size of the network in terms of the number of samples, but not the number of total values (number of features times number of samples), in X_{tr}. Another rule of thumb that involves p, the number of features per sample, is $10\cdot|\mathbf{W}|\le n_{tr}\cdot p$. For example, $|\mathbf{W}|\le 150\cdot 4/10=60$ for resubstitution training (i.e., n_{tr} = n = 150) of the Iris data, which limits the network weight vector to a total of 60 parameters. None of these bounds account for the variability that real data possess, and there are as many of them as you have time to read about, but you should always be cognizant of the "power" of your training data - its size certainly limits the total number of parameters you should estimate with it, be the design a NN or some other type of classifier.

Just above the diagram in Figure 4.71 you see $\mathbf{NN}:\Re^p\mapsto\Re^c$. This emphasizes that mathematically the NN is just a vector field. We use the notation **NN** when the role of the **NN** as a function is being emphasized; of course, **NN** is a computational transformation realized *only* by computer implementation. And we use (unbold) NN when talking about a NN generally, or in the engineering design sense, as an input - output system. In this regard **NN** is exactly like the classifiers based on decision trees and fuzzy systems discussed in section 4.6. Here, **NN** will be a feature selector, clustering algorithm or classifier function depending on the discussion at hand. When it is a classifier, we use our standard notation $\mathbf{D}_{\mathbf{NN}}$.

One touted advantage of neural networks is that their parameters can be "learned" from labeled training data. But this is true of every classifier - that's what supervised learning means. The real power of NNs lies in the way they build up functional approximations to IO mappings that underlie the training data. Kreinovich et al. (1998) provide a very nice discussion of this aspect of NNs in the context of universal approximation theory. Parametric learning by a NN is based on an *update function or strategy* that converts the current set of weights $\mathbf{W}_t$ at the t-th training cycle or iteration into a new or updated set $\mathbf{W}_{t+1}$. The action of the update or *learning rule* can be written symbolically as $\mathbf{W}_{t+1} = U(\mathbf{W}_t)$. Updating is done during training whenever the NN system output(s) do not correspond well enough to the desired labeled outputs. For pattern recognition, this usually means that the NN is operating as a classifier.

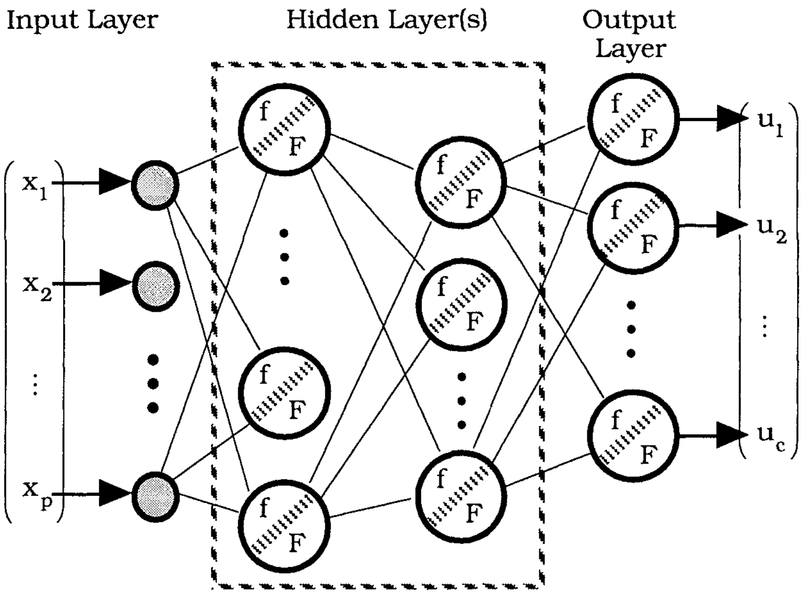


Figure 4.72 The FF network with node functions $\Phi = F \circ f$

There are many principles that guide the choice of a learning strategy. Different learning rules are chosen to match a specific network architecture; most update rules attempt to optimize some function of the observed error(s) between the desired and observed outputs of the network. By far the most popular and pervasive NN to date is the *feed forward* (FF) network, (Zurada, 1992, Haykin, 1994). Figure 4.72 shows a typical representation for a FF network. The main difference between this structure and the one shown in Figure 4.71 is in the topology of the interconnections between the nodes. In a FF network there are no self loops or feedback connections; data at each stage of the network in Figure 4.72 can only flow forward (to the next layer) from left to right. The standard algorithm for updating in the FF case is the back-propagation technique invented by Werbos (1974). Given the central importance of back-propagation in the training of FF networks, these networks are referred to as *feed-forward back-propagation* (FFBP) networks. They are usually called *multi-layer perceptrons* (MLP). Many authors, including us, reserve the term MLP for the special case of the FFBP network in which every node function is $\Phi_{LH} = F_L \circ f_H$.

In the BP method, the error function for a given IO pair $(\mathbf{x} \in \Re^p, \mathbf{y} \in \Re^q)$ is the sum of squared errors between the desired and target outputs, $E(\mathbf{x}, \mathbf{W}_t) = \|\mathbf{NN}(\mathbf{x}, \mathbf{W}_t) - \mathbf{y}\|^2$. E is regarded as a function of the current network weight vector, and differentiation of this function of $\mathbf{W}_t$ leads to necessary conditions for adjustments of the weights by gradient descent. The input is fed forward, and the error it causes produces updates to the current weights that are then propagated backwards through the network layer by layer - hence, FFBP. We will not repeat the formulae for this well known procedure here. If a specific need arises in connection with fuzzification of some part of the FFBP design, we will discuss what seems most appropriate at that juncture.

It is impossible for us to estimate how many fuzzy variants of the FFBP structure shown in Figure 4.72 have been discussed in the literature of so-called "fuzzy-neuro (aka neuro-fuzzy)" systems in the last decade. Suffice it to say that there are least a half dozen textbooks whose titles suggest that they deal exclusively with this: for example, *Neural Fuzzy Systems* (Lin and Lee, 1996),*Neuro-Fuzzy and Soft Computing* (Jang et al., 1997) and *Neuro-Fuzzy Controllers* (Godjevac, 1997). In order to appreciate some of the extensions of the NN to be developed subsequently, we present in Example 4.21 the results of using the standard FFBP network to design a crisp classifier with the Iris data.

Example 4.21 The Iris data is labeled, and can be used to estimate the parameters of a FF network in many ways. Here we show the results of training the same MLP network with three different training and testing strategies. Specifically, we train the network shown in Figure 4.72 with node functions $\Phi_{LH} = F_L \circ f_H$ using back-propagation with these three protocols:

A. $X_{tr} = X =$ Iris, leading to the resubstitution error estimate $E_{\mathbf{D}_{NN}}(X|X)$.

B. X_{tr} = the union of the first 25 points from each of the three labeled classes; X_{te} = the union of the remaining 25 points from each class, leading to the estimate $E_{\mathbf{D}_{NN}}(X_{te}|X_{tr})$. Cross validation was not done for this example.

C. Finally, we illustrate the leave one out procedure by building, for $k = 1,\ldots 150$, the classifiers $\{\mathbf{D}_{NN,k}\}$ by constructing the training and test sets $X_{tr,k} = X - \{\mathbf{x}_k\}$ and $X_{te,k} = \{\mathbf{x}_k\}$. From these we can compute an average error rate, $E_{\bar{\mathbf{D}}_{NN}}(X_{te}|X_{tr}) = \sum_{k=1}^{150} E_{\mathbf{D}_{NN,k}}(X_{te,k}|X_{tr,k})/150$.

The MLP we used for all three experiments was a simple one: it had two hidden layers with 6 nodes each and an output layer with c = 3 nodes. Since the Iris data is 4 dimensional, this gives a 4:6:6:3 configuration. The 15 computing nodes all use the linear integrator function f_H and the logistic function F_L. For simplicity we fixed $\lambda = 1$ and $\beta = 0$ for the logistic functions at all 15 nodes. Consequently, the only parameters that must be estimated are the weight vectors $\{\mathbf{w}_i\}$ and bias constants $\{\alpha_i\}$ of the 15 hyperplanes at the computing nodes. That is, the cardinality of the weight vector for this structure is $|\mathbf{W}| = (4+1)\cdot 6 + (6+1)\cdot 6 + (6+1)\cdot 3 = 93$ parameters. Do we have enough data to expect good generalization with this structure? Since (4.100) and (4.101) are for single (hidden) layer networks, they don't apply to our topology. The only guideline we have is the rule of thumb $10\cdot|\mathbf{W}| \le n_{tr}\cdot p$. Solving this inequality for n_{tr} with the values $p = 4$ and $|\mathbf{W}| = 93$ gives $n_{tr} \ge 10\cdot 93/4 = 232.5$. Since n = 150 for Iris, no scheme for training and testing this network can satisfy this rule of thumb. Let's see how good the rule is.

Each of the networks for experiments A and B was initialized randomly. Training was terminated when the MSE on the training data was less than 0.01 for 10 consecutive passes (epochs) through it, or at the maximum specified limit of 200,000 epochs. For experiment C, the network was initialized randomly at k = 1, 51 and 101, i.e., at the start of each new class, and the weights from the training runs at these three k's were retained and used to initialize the remaining 49 training sessions for points in that class. Without this "jump start" for better initialization, some of the experiment C runs ran to the iteration limit of 200,000 passes without satisfying the (successive iterates) termination criterion. In other words, carrying the final weights from the previous run forward to initialize the next training session in the leave one out tests helped network training a lot.

All networks used learning rate and momentum factors of 0.5. The learning rate refers here to a multiplier in the update rule for the weights, and has exactly the same meaning as the term did in earlier sections. We have not discussed momentum, since there has been little work that we know of on "fuzzy momentum". Momentum is a term that is added to the update rule for the network weights, and it is often able to accelerate back-propagation learning towards termination. See Section 4.5 in Zurada (1993) for an excellent discussion of this topic.

Experiments A and B aren't very exciting, and are easy to report. Training method A with n_{tr} = 150 led to a MLP classifier with a resubstitution error rate of zero, and training method B with n_{tr} = 75 terminated at a network with 1 testing error. Thus given all (experiment A) or half (experiment B) of the Iris data for training, it is not hard to find a network for which $E_{\mathbf{D}_{NN}}(X|X) = 0; E_{\mathbf{D}_{NN}}(X_{te}|X_{tr}) = 1$. Don't forget that the resubstitution estimate is usually optimistically biased as you evaluate this result.

Experiment C is more interesting, for here we use n_{tr} = 149 of the 150 points in Iris for training (which is almost the same training data as resubstitution uses), and test the resultant classifier on the point held out (which is not resubstitution). In our set of 150 trials using this scheme, the classifier built with $X_{tr,k} = X - \{\mathbf{x}_k\}$ and tested on $X_{te,k} = \{\mathbf{x}_k\}$ gave the wrong classification for k = 2, 42 (called class 2|were class 1), 57 (called class 1|was class 2) and 71 (called class3|was class 1). Thus, for this network configuration, $E_{\mathbf{D}_{NN,k}}(\{\mathbf{x}_k\}|X - \{\mathbf{x}_k\}) = 4/150 = 2.66\%$.

How did our rule of thumb about the number of parameters versus the size of the training data do? Well, this rule of thumb is not tied to a specific error rate like (4.100) is, so we cannot say that it failed (except in case A - no one would argue that a perfect score is undesirable). On the other hand, the worst case, experiment C, produced an average error rate of 2.66% with less training data than the rule of thumb recommends. So? Take rules of thumb for what they are - general guidelines or heuristics that work sometimes, for some algorithms, and some data sets.

The errors committed in experiment C are particularly interesting in that class 1 (Sestosa) is usually the subspecies that is handled perfectly by classifiers, but here, the leave one out networks committed two errors on class one test points, while class 3

(Virginica) here showed no errors. This suggests that the decision functions built by the network are more complex than the simpler ones we have studied so far. The leave one out error rate is really a little misleading because it is not for just one classifier; nonetheless, it is often taken as a "representative" best (and most pessimistic) estimate of the error rate that can be expected using a similar design on the same type of data. We also remind you that these results depend on the initialization used, and for a different set of starting points, something entirely different could happen.

Recalling Table 4.9, we know that the experiment C error rate (2.66%) can be achieved with c = 7 LVQ prototypes. However, the error rate shown in Table 4.9 is the resubstitution error rate on all 150 points, and experiment A in this example shows that the FFBP NN easily achieves 0% errors in the resubstitution case. Since the leave one out error rate is the most pessimistic one we can compute, and here we have 2.66% for it, we are tempted to conclude that a simple MLP of the type represented by the 152 classifiers designed in this example is, for this data set at least, more likely to produce lower generalization errors than the nearest multiple prototype classifiers in Section 4.3.

Finally, we comment on the training time it took for each of the 150 leave one out classifiers designed in experiment C. In 57 of the 150 designs, termination was achieved in less than 1000 passes through the training data, which took about 5 seconds on a SUN workstation. This was the case for all 50 points in class 3. At the other end of the scale, the 32nd point in class 1 took more than 46,000 passes through the 149 training data to satisfy the termination criterion. This sounds like a lot, but this run only used about 4 minutes of CPU time. Moreover, once trained, NN classifiers are fast, and (for small data sets anyway) we think some type of network design should always be tried when you start building classifiers with labeled data.

To show the versatility of the FFBP network, we give another example of its use for an entirely different pattern recognition problem - feature extraction. This idea had its origins in the work of Cottrell et al. (1989). Many papers have been written that use the basic idea illustrated in Example 4.22.

Example 4.22 We seek a 2-dimensional data set *extracted from* the Iris data by a FF design. Figure 4.73 shows the architecture of the network to be used. There are 4 input nodes, one hidden layer with 2 computing nodes that use $\Phi_{LH} = F_L \circ f_H$, and the output layer has 4 nodes, thus making a 4:2:4 configuration. The wrinkle here is that the target outputs for this application are the *input vectors*. That is, we want the NN to function like the identity map, so $\mathbf{NN}(\mathbf{x}_k) \approx \mathbf{x}_k \; \forall k$. The idea that underlies this design is that if NN does function as the identity, then the data flowing through every layer of the network will, by and large, possess the same "information" as the inputs themselves. And in particular, the vectors $\mathbf{y}_k = (y_{1k}, y_{2k})^T \; \forall k$ that are copied from the output of the hidden layer should be a good pair of extracted features in this loosely defined sense.

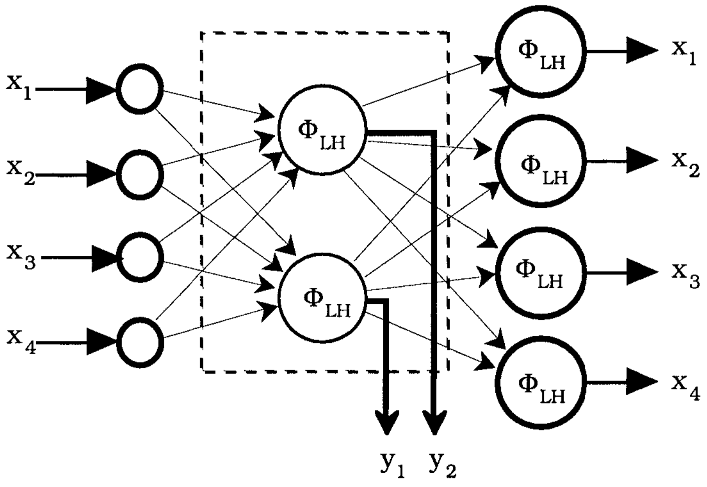


Figure 4.73 A MLP approach to feature extraction from the Iris data

As shown in Figure 4.73, the basic structure of the six computing nodes is that each used a linear integrator and logistic transfer function. We use *bipolar* logistic transfer functions in this example to demonstrate that the only difference between these and their unipolar relatives is a matter of scaling. Since the range of the Iris data is [0.1, 7.9], each logistic function was scaled by 10, enabling the output of each node to range over the interval [-10, 10]. Thus, the specific form of each node function is $10\Phi_{L,biH}$.

As in Example 4.21, the parameters of all 6 bipolar logistic functions were fixed at $\lambda = 1$, $\beta = 0$, so the parameters acquired during learning in this example are again the weight vectors $\{\mathbf{w}_i\}$ and bias constants $\{\alpha_i\}$ of the 6 hyperplanes at the computing nodes. Now there are only $|\mathbf{W}| = (4+1)\cdot 2 + (2+1)\cdot 4 = 22$ network parameters, so $n_{tr} \geq 10 \cdot 22 / 4 = 55$. Our rule of thumb says we can use Iris, or any subset of it with at least 55 samples, for training. Moreover, this network satisfies the requirements laid out for equations (4.100) and (4.101). With M = 2 and $|\mathbf{W}| = 22$, we can either pick n_{tr} and compute $E_D(X_{te}|X_{tr})$, or fix $E_D(X_{te}|X_{tr})$ at some desired level, and solve (4.100) for n_{tr}. Suppose we insist that the generalization error in (4.100) be less than 10%, $E_D(X_{te}|X_{tr}) = 0.1$. Then with (4.100) we compute

$$\left(\frac{704}{.1}\right)\log_e\left(\frac{64}{.1}\right) = 45,489 >> 150,$$

so by (4.100) it will impossible to attain a 10% test error. Indeed, using (4.100) with 100% test errors leads to

$$\left(\frac{740}{1}\right)\log_e\left(\frac{64}{1}\right) = 3,078 > 150.$$

These results suggest that we can't hope for the success we reported in Example 4.21 for this problem. But to be fair to the error bounds, we point out that the bound in (4.100) also assumes that during training we can obtain a resubstitution rate on the training data that is no more than half of the testing rate, and we did not conduct this experiment.

All 150 points in Iris were fed sequentially through this network during training to acquire the network weight vector. Training was terminated when the overall sum of squared errors between the inputs and outputs of the network was less than 17 misclassifications of the hardened outputs. At termination, the resubstitution MSE was 16.971. By (4.100), this means the best generalization error we could expect is about 34%. Putting 0.34 into the denominator of (4.100) gives n_{tr} = 10,845. Hmmmmm...........

Returning to the problem at hand, after termination, each point in X = Iris is fed through the network one more time, generating $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_{150}\} \subset \Re^2$, a labeled set of 2D vectors that can be used to represent the 4D Iris data. Figure 4.74 is a scatter plot of the 150 points in Y found by this technique. Each $\mathbf{y}_k$ automatically acquires the same label as $\mathbf{x}_k$ in the original data set, so the class labels of the 50 points in each of the three clusters can be illustrated by different symbols.

The vertical lines $y_1 = 2$ and $y_1 = -1.03$ represent a linear classifier that separates the extracted data into three groups. It is easy to see that the 4 "×'s" to the left of the vertical line $y_1 = -1.03$ are the only resubstitution errors committed by this classifier. Thus, the two dimensional data set Y extracted by the FFBP network provides a substantial improvement over the resubstitution error rate that can be achieved by a set of hyperplanes in the original four dimensional data set X. In fact, Figure 4.74 shows that only one feature, y_1, is needed to achieve this error rate.

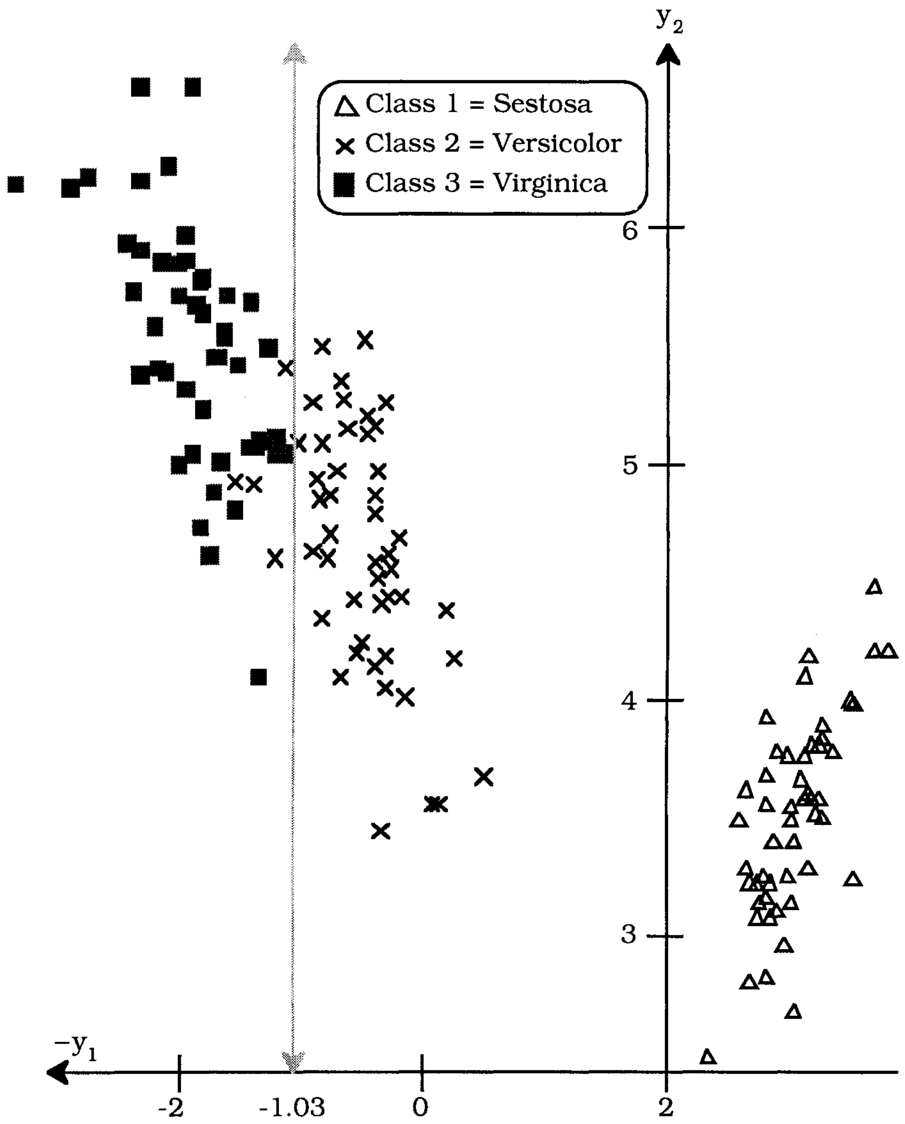


Figure 4.74 A NN approach to feature extraction from the Iris data

Before you get really excited about the NN method, we want to show you the result of feature extraction on Iris using the standard linear transformation known as *principal components analysis* (PCA). We aren't going to discuss this topic; instead you are referred to the wonderfully readable treatment of PCA in Johnson and Wichern (1992). Figure 4.75 shows the projection of Iris onto its first two principal components. Comparing this view to Figure 4.74 shows that PCA and the NN in this example extract very similar features. We are again able to construct a pair of hyperplanes in Figure 4.75, by eye, that commit either 3 or 4 resubstitution errors - depending on how good your eye is - on the extracted data (if you make the calculations, it turns out to be 4 errors).

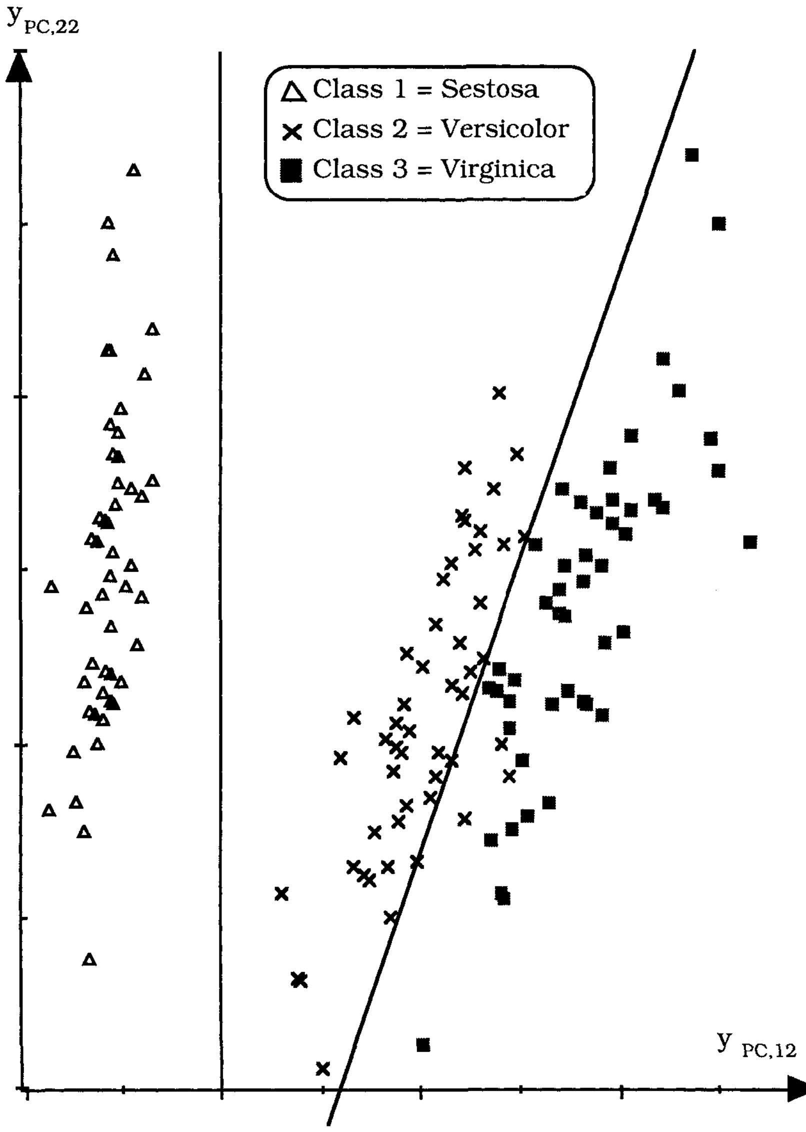


Figure 4.75 Feature extraction from Iris with principal components

And finally, let's have a closer look at Iris$_{34}$, the data scatterplotted in Figure 4.12. Figure 4.76 is another plot of the same data, now shown with the extra information that accrues from having crisp class labels. This also depicts a feature extraction method - the special case called orthogonal projection illustrated in Figure 2.22. And again, it's easy to construct a piecewise linear classifier with the pair of hyperplanes in Figure 4.76 that commits only 3 or 4 resubstitution errors.

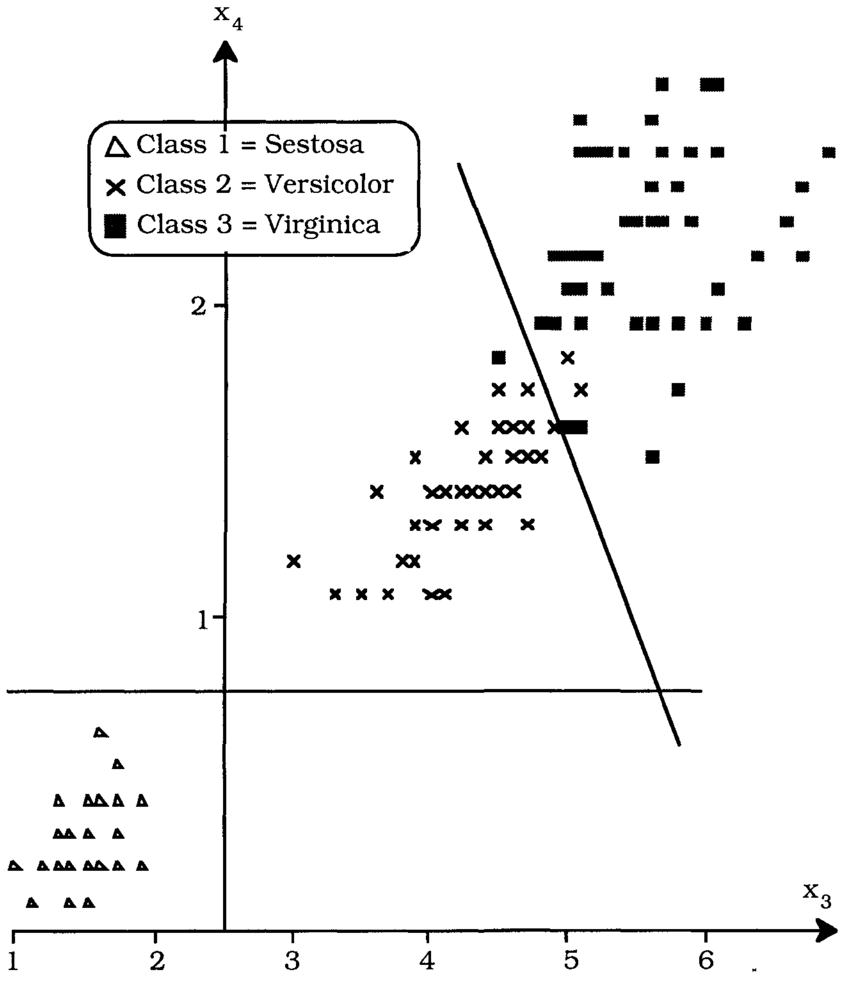


Figure 4.76 Feature extraction from Iris by orthogonal projection (selection) to get Iris$_{34}$

Given Figure 4.76, you must be wondering - why bother with these complicated classifier designs when I can just project the data into

$\Re^2$ and eyeball a pretty good linear classifier onto its scatterplot? Well, if you can do it this way, you *should* do it this way - as Einstein once said "simple is best - but only simple enough to work". But the reason it works here is because Iris is nice to us. For one thing, p = 4, so there aren't a lot of pairs of features to look at. But suppose you have p = 100 features. Then there will be 4,950 pairs of features to scatterplot, so selection by visual inspection becomes pretty tough. We think you should always try simple tricks like this, but don't count on them too much. After all, you don't expect the hare to void turd the size of elephant dung.

The MLP can also be used to *select* (instead of extract, as in Example 4.22) a good subset of features. For example, Pal and Chintalpudi (1997) made a simple modification of the conventional MLP for feature selection. Each *input* layer node becomes a computing node by associating it with a multiplier which lies between 0 and 1. If the multiplier is zero then that feature does not pass into the network, while if the multiplier is 1 then the associated feature passes into the network unattenuated. For intermediate values of the multiplier, the feature is partially attenuated. Pal and Chintalpudi realized the multiplier using a multiplier function with a tunable parameter.

Using our terminology, let f(**x**) = **x** be the identity function, let $G_\lambda : \Re \mapsto [0,1]$ be a monotonic, non-decreasing real-valued function parametrized in the real number λ (e.g., the unipolar sigmoid), and define $F(y) = y \cdot G_\lambda(y)$. Thus the effect of $\Phi(\mathbf{x}) = F \circ f(\mathbf{x}) = y \cdot G_\lambda(y)$ is to multiply **x** by a multiplier function g with a tunable parameter λ, where $G_\lambda(y)$ is in [0,1]. If $g_\lambda(y)$ of an input node is 1, then the corresponding feature is important and passed unattenuated into the net; if $G_\lambda(y)$= 0, then that feature is irrelevant or harmful and is not allowed to enter the network. The Type I fuzzy neuron depicted in the lower half of Figure 4.77 is very similar to the input node structure proposed by Pal and Chintlapudi, but the Pal and Chintlapudi neuron does not necessarily produce outputs that lie in the interval [0, 1].

In Pal and Chintlapudi the non-input layers are exactly like those in the conventional MLP. The multiplier parameters λ_i i = 1,..., p, are learned along with the connection weights using the usual back-propagation algorithm. The training starts with all multiplier functions set to almost zero, i.e. with almost 100% attenuation. Thus, at the beginning of training, practically none of the features are allowed to pass into the network. As the network trains, it selectively allows only some important features to be active by adjusting their multiplier values as dictated by the gradient descent.

The training can be stopped when the network has classified satisfactorily i.e., when the training error rate has gone down to a tolerable value. Features with high values of the attenuation factor (i.e., small multipliers) may be eliminated from the feature space.

Ghosh et al. (1993) discuss the conversion of a multilayer perceptron to an unsupervised network by the introduction of concepts from fuzzy theory. This fuzzy neural network can extract objects in a noisy environment in a completely unsupervised manner by minimizing a measure of fuzziness computed on the output of the network.

Yan (1993) presents a scheme for extracting multiple prototypes from crisply labeled training data, $X_{tr} = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\} \subset \Re^p$, using a 3 layer perceptron that is very similar in spirit to the method presented in Example 4.22. Yan's objective is to reduce the size of X_{tr} through the transformation $\mathbf{V}_c = \hat{X}_{tr} = \Omega(X_{tr})$, exactly as depicted in Figure 4.2. Yan uses a p+2 : c: $\hat{c}$ multilayered perceptron as the function Ω. We remind you that in the setting of multiple prototypes in Definition 4.2, $\hat{c}$ is the number of classes in X_{tr}, $\hat{c} \le c$. Yan's desire is find a set of multiple prototypes, called $\mathbf{V}_c$ in Definition 4.2, for which c << n, and for which the resubstitution error of the 1-nmp classifier $\mathbf{D}_{\mathbf{V}_c,\mathbf{E}_{\hat{c}},\delta}$ in equation (4.7) is zero (we have also called this consistency). While consistency is a stated objective in Yan (1993), no guarantee is claimed; the method is consistent for one of the numerical examples given, and may be for the second one too, but this is not stated.

Did you notice that Yan's MLP structure was p+2 : c: $\hat{c}$? In this interesting paper Yan increases the dimension of the input space by 2, adding the number $\|\mathbf{x}_k\|^2/2$ as the p+1- st coordinate of each $\mathbf{x}_k$, and the constant 1 as its p+2-nd coordinate. Yan argues that the number $\|\mathbf{x}_k\|^2/2$ is chosen so that this MLP - without sigmoids in the computing nodes and before training - can be regarded as an *approximation to* the 1-nn rule in equation (4.38).

The $\hat{c}$ output nodes are completely fixed, using a linear integrator function that has user-defined weight vectors depending on just two parameters of opposite signs. The activation function in each output node is the unipolar sigmoid F_L in (4.97) with $\lambda = 1$, $\beta = 0$. Thus, there is no adjustment in the output layer during training.

The desired prototypes in Yan's scheme are the weight vectors (in $\Re^p$) of the hidden layer nodes, each of which uses the standard node function Φ_{LH} with fixed sigmoidal parameters as in the output layer. Backpropagation training adjusts the initial prototypes (which are specified in the paper by a third user-defined parameter), and at the end of training, the n points in X_{tr} that have $\hat{c}$ classes are replaced by the c point-prototypes in the vector $\mathbf{V}_c$. At this state Yan has the basic equipment needed to build the 1-nmp classifier $\mathbf{D}_{\mathbf{V}_c,\mathbf{E}_{\hat{c}},\delta}$, needing only to pick a distance measure (Euclidean in the paper). Yan calls the prototypes obtained by this MLP "optimized prototypes".

Two numerical examples are given in Yan (1993), both for 2D data. The first example is an "XOR-like" data set of 162 points in the plane that form (roughly) four clusters in the shape of an "X", but the clusters have only $\hat{c} = 2$ crisp labels. These 162 labeled data points are replaced by c = 4 2D "optimized prototypes" with the result that, when used in the 1-nmp rule in (4.7), they give zero resubstitution error. The second example uses 500 labeled (image) data that are distorted digits for training, and another 500 data for testing. Yan states that the 1-nn rule achieves 99% accuracy using all 500 training data to label the test data, and that 10 optimized prototypes obtained with his method achieve a testing success of 99.4%, slightly better than the full 1-nn rule, using only 2% (10/500) as many points. We will return to this interesting method in Section 4.8.

The objective of Section 4.7.B has been to introduce the terminology associated with the most popular NN models. The remainder of this chapter is devoted to specific ways that one or more components of a NN model can be altered to accommodate and manipulate fuzziness. Any and all of the modifications we describe can justifiably be called *fuzzy neural networks*. As we mentioned earlier, this has been done in so many ways that it is impossible for us to lead you through the forest; the best we can hope for is to give you a glimpse at some of the trees.

C. Fuzzy Neurons

Example 4.21 (and a metric ton of papers over the last 15 years) demonstrate how good plain old FFBP and especially MLP nets can be in pattern recognition, and in particular, for classifier design. Fuzzy sets were created to deal with linguistic information and provide an interface between linguistic and numeric descriptions. So, we hope you see in what follows that two advantages can be realized by networks that have Type I fuzzy neurons as defined in equation (4.102). The most important contribution of adding fuzziness to a NN structure is that, after training, the nodes of a fuzzy neural network can admit a linguistic interpretation, i.e., some insight can be gained into how the features combine to make a class decision. The opaqueness of FF nets has always been an issue; it's hard to trust a "black box". The second advantage is that for many types of data, we have found that networks of fuzzy neurons train in many fewer epochs (although the calculations done during training may increase).

Keller and Hunt (1985) first introduced fuzzy sets into the training of a single perceptron (don't we just love to cite ourselves?). We have not discussed the perceptron, nor do we intend to, for it is arguably the most well known linear classifier that had its roots in a desire to mimic the BNN. But for the record, it's a linear classifier that was originally designed for c = 2 class problems, and the perceptron learning rule is an iterative procedure that finds estimates of the parameters of the sought after separating hyperplane. When the training data have two linearly separable classes, the perceptron convergence theorem guarantees us that the iterative learning procedure converges to a separating hyperplane in finitely many steps (Duda and Hart, 1973).

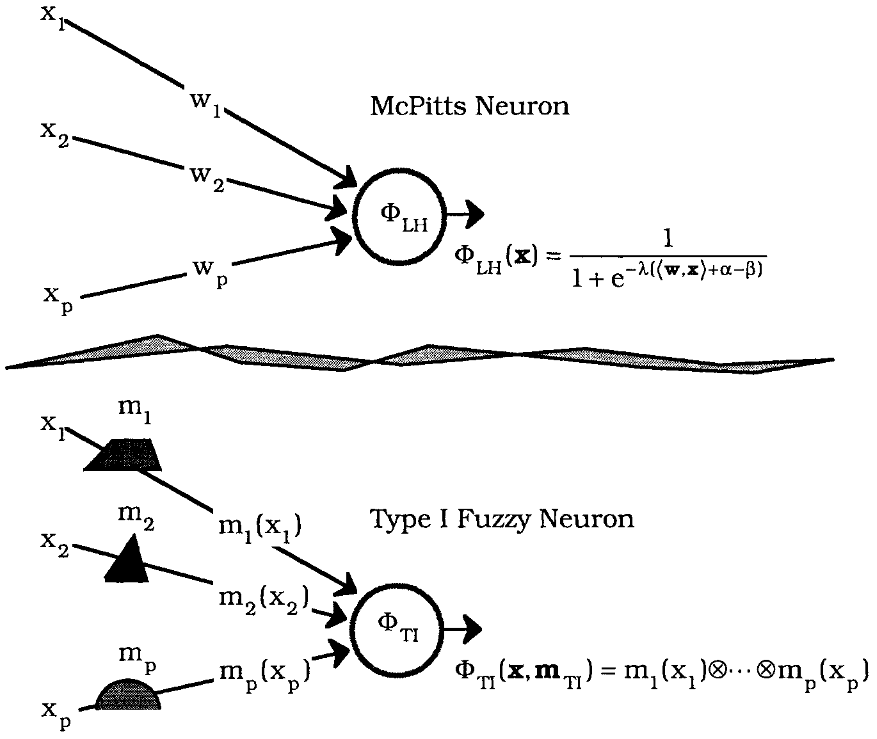
Keller and Hunt's fuzzification of the perceptron training rule generally resulted in faster convergence, also guaranteed a separating hyperplane if one existed, and produced good results when the data were not linearly separable - a big problem with the classical perceptron training algorithm (Rosenblatt, 1957). Fuzzification of the training rule was extended to MLP nets by Pal and Mitra (1992). These connections to NN training, along with the use of neural networks to perform operations like fuzzy inference will not be pursued here. The reader is referred to Lin and Lee (1996) for development of these and other relationships between fuzzy sets and neural networks.

There are many ways to integrate fuzzy sets into a neuron model. Most of these methods involve changing the integrator and activation functions of the standard McPitts neuron. Some involve changing the input data and/or the weight values from real numbers to fuzzy sets. There are even multiple taxonomies developed to describe the various possible modifications. Lee and Lee (1970, 1975) were the first to postulate and describe fuzzy neurons and they analyzed fuzzy neural networks based on their fuzzy neurons from the standpoint of fuzzy automata theory. See (Lin and Lee, 1996, Jang et al., 1997, Godjevac, 1997, Pedrycz et al., 1998) for extensive details about many forms of fuzzy neurons. Gupta and Rao (1994) discuss various principles of fuzzifying neurons and neural networks; and Buckley and Hayashi (1994) provide a nice summary of fuzzy neural nets that process fuzzy signals and/or have fuzzy weights.

We begin with the basic mathematical neuron model in Figure 4.68. Most of the fuzzy variants of this node change the form of one or more of the input vector $\mathbf{x}$, the weight vector $\mathbf{w}$, or the integrator/activation functions f/F. The fuzzy neuron which seems to be encountered most often in pattern recognition is called the *Type I neuron*. Each input x_i is still a real number, but the weight w_i is "thought of" as a fuzzy set whose membership function is m_i. The integrator/activation function pair $\Phi = F \circ f$ is replaced by some fuzzy aggregation function. That is, the output of a Type I fuzzy neuron is given by

$$\Phi_{TI}(\mathbf{x}, \mathbf{m}_{TI}) = m_1(x_1) \otimes m_2(x_2) \otimes \cdots \otimes m_p(x_p) \quad , \qquad (4.102)$$

where the vector $\mathbf{m}_{TI} = (m_1, \cdots, m_p)^T$ of input edge membership functions effectively become the "weights" for the node. The symbol $\otimes$ in (4.102) is used to represent a fuzzy set connective operator: union (OR neuron), intersection (AND neuron), generalized mean, or hybrid. If $\otimes$ is a T-norm, then the Type I fuzzy neuron simply computes the LHS activation or firing strength of a fuzzy rule $\alpha_i(\mathbf{x})$ exactly as in equation (4.72). Figure 4.77 shows you the conceptual difference between the McPitts and Type I fuzzy neurons.



x_1
x_2
x_p
w_1
w_2
w_p
McPitts Neuron
Φ_{LH}
$\Phi_{LH}(\mathbf{x}) = \frac{1}{1 + e^{-\lambda(\langle \mathbf{w}, \mathbf{x} \rangle + \alpha - \beta)}}$
m_1
m_2
m_p
$m_1(x_1)$
$m_2(x_2)$
$m_p(x_p)$
Type I Fuzzy Neuron
Φ_{TI}
$\Phi_{TI}(\mathbf{x}, \mathbf{m}_{TI}) = m_1(x_1) \otimes \cdots \otimes m_p(x_p)$

Figure 4.77 Standard and Type I fuzzy neurons

We have added graphs of specific membership functions on the input edges to the Type I fuzzy neuron for illustration, but these functions can be any membership functions. In view of Figure 4.77 you see why we say that the weights are "thought of" as fuzzy sets. They are not weights at all in the normal neural networks sense, but are used to convert real inputs into membership values that lie in [0, 1]. Thus, the use of $\mathbf{m}_{TI} = (m_1, \cdots, m_p)^T$ serves to normalize the input features. These functions allow you to choose membership function shapes that weight different features and feature values differently, and by this device you can build a lot of factual as well as heuristic knowledge about the process generating the data into the model.

From the pattern recognition standpoint the input edge membership functions also serve as a mechanism to convert raw input features into degrees of satisfaction of a class hypothesis. Yamakawa et al.'s (1992) fuzzy neuron, on the other hand, has a fixed membership function and also a tunable real weight for every input link to a neuron.

You can see from Figure 4.77 how a layer of AND neurons can be used to emulate the LHS of a fuzzy rule base. If the AND layer is followed by other layers of specialized neurons, like OR neurons and "weighted averaging" neurons, then you can view the action of a fuzzy rule base as a special case of fuzzy neural networks. The so-called *adaptive-network-based fuzzy inference system* (ANFIS, Jang et al., 1997) is one such realization, although there are many in the literature. Our purpose here is not to study all the interconnections between fuzzy sets and neural-like structures and get thereby be caught in the jungle of function approximation (see the excellent article by Dubois et al., 1998, reproduced in part in section 4.10). We are interested in the value-added to the pattern recognition problem when standard neurons are replaced by fuzzy neurons. What could that possibly be?

Using a parametrized operator (such as Yager unions and intersections, weighted generalized means, etc.), a fuzzy neuron can be defined which affords the opportunity to be trained (i.e., it's parameter(s) can be learned). A back-propagation algorithm can be devised since partial derivatives of these families of operators can be computed. Krishnapuram and Lee (1988, 1989, 1992a) used these basic operators in a network that has Type I fuzzy neurons to do multicriteria decision making. One basic problem is that the category of neuron must be determined (does the data represent conjunction, disjunction, or compensatory criteria?). This makes the training algorithm cumbersome. In what follows, we develop the use of hybrid fuzzy connectives, first for a single Type I fuzzy neuron, and then in the next section, for networks of such neurons.

Fuzzy set theoretic connectives, i.e., unions, intersections, generalized means, and hybrid operators, are useful for aggregating memberships functions. The resulting membership depends on the type of aggregation operator used, and this type is dictated by the "attitude" that we expect the aggregation connective to have. These operators are very useful in decision analysis and decision making. You are referred to (Dubois and Prade, 1985, Mizumoto, 1989, Klir and Yuan, 1995, Dyckhoff and Pedrycz, 1984) for a more complete description of fuzzy set connectives, which is what a Type I fuzzy neuron computes.

First, we need to develop some aggregation operators that can be used for unions and intersections. The most well known and oldest of these is the *generalized* (or weighted) *mean of order q* (which was called a q-norm in older mathematics books such as Beckenbach and Bellman, 1961). This function combines a set of p positive inputs, say $\mathbf{x} = (x_1, \ldots, x_p)^T ; x_i > 0 \, \forall \, i$, with a set of p convex weights, say $\mathbf{w} = (w_1, \ldots, w_p)^T$, as follows:

$$M_q(\mathbf{w}, \mathbf{x}) = \left(\sum_{i=1}^{p} w_i x_i^q \right)^{1/q}, \; q \neq 0 \qquad , \qquad (4.103a)$$

where the weights $\mathbf{w} = (w_1, \ldots, w_p)^T$ satisfy the constraints

$$w_i > 0 \, \forall \, i; \; \sum_{i=1}^{p} w_i = 1 \qquad . \qquad (4.103b)$$

M_q can be interpreted as a *node function*, say $\Phi_{M_q} = F_{1/q} \circ f_q$, where the integrator and transfer functions are defined as $f_q(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{p} w_i x_i^q$ and $F_{1/q}(y) = y^{1/q}$. The parameter q may or may not be part of the unknowns that must be estimated when using (4.103b) as a fuzzy neuron. When q is unknown, it corresponds in some very loose sense to the offset parameter α of the linear integrator function used by the McPitts neuron.

The generalized means of orders -1, 1 and 2 are, respectively, the *harmonic*, *arithmetic* and *RMS* means of $\mathbf{x}$. The weights $\{w_i\}$ may be (usually *are*) chosen to be equal to (1/p), and then $M_q(\mathbf{1}/p, \mathbf{x}) \propto \|\mathbf{x}\|_q$, the Minkowski q-norm in equation (1.11). Here are the most important properties of M_q (Beckenbach and Bellman, 1961):

$$M_0(\mathbf{w}, \mathbf{x}) \equiv \underbrace{\lim}_{q \to 0} \left\{ M_q(\mathbf{w}, \mathbf{x}) \right\} = \prod_{i=1}^{p} x_i^{w_i}, \; q \neq 0 \qquad ; \qquad (4.103c)$$

$$M_\infty(\mathbf{w}, \mathbf{x}) \equiv \underbrace{\lim}_{q \to \infty} \left\{ M_q(\mathbf{w}, \mathbf{x}) \right\} = \underbrace{\max}_{1 \le i \le p} \left\{ x_i \right\} \qquad ; \qquad (4.103d)$$

$$M_{-\infty}(\mathbf{w},\mathbf{x}) \equiv \underbrace{\lim}_{q\to-\infty}\left\{M_q(\mathbf{w},\mathbf{x})\right\} = \underbrace{\min}_{1\le i\le p}\left\{x_i\right\} \quad ;\text{ and} \qquad (4.103e)$$

$$M_{-\infty}(\mathbf{w},\mathbf{x}) \le M_q(\mathbf{w},\mathbf{x}) \le M_{\infty}(\mathbf{w},\mathbf{x}) \;\; \forall q \in \Re \quad . \qquad (4.103f)$$

M_0 is called the *geometric mean*. If we relax the positivity constraint on the $\{x_i\}$, and require only that these numbers be non-negative, we must restrict q to be positive, or define $M_q(\mathbf{w},\mathbf{x})$ to be 0 for all $q \le 0$. When the $\{x_i\}$ are all positive, $M_q(\mathbf{w},\mathbf{x})$, is a non-decreasing function of q, and when they are all distinct, $M_q(\mathbf{w},\mathbf{x})$ is strictly increasing,

$$p < q \Rightarrow M_p(\mathbf{w},\mathbf{x}) < M_q(\mathbf{w},\mathbf{x}),\;\; x_i > 0 \; \forall i \quad . \qquad (4.103g)$$

For the special case when $x_i \in (0,1]$ for all i, equations (4.103e) and (4.103f) show that for all real q, $M_q(\mathbf{w},\mathbf{x})$ is pinched in-between the (largest) T-norm and smallest T-conorm of $\mathbf{x}$:

$$\cap\text{'s} \leftarrow T_3(\mathbf{x}) < M_q(\mathbf{w},\mathbf{x}) < S_3(\mathbf{x}) \;\rightarrow\; \cup\text{'s},\;\; x_i \in (0,1] \, \forall i. \qquad (4.103h)$$

Because of (4.103h), all of the $M_q(\mathbf{w},\mathbf{x})$'s can be *interpreted* either as intersections or unions. Here, the weight w_i associated with x_i can be thought of as the relative importance of x_i. In the context of aggregation of fuzzy evidence, when we use the mean of order q, we attempt to choose q to suit the required (or desired) degree of optimism or pessimism we have about the values concerned. For example, $M_q(\mathbf{w},\mathbf{x})$ can be used to approximate behaviors such as "at least" and "at most" (Krishnapuram and Lee, 1988). When used in fuzzy aggregation networks (which will be discussed in Section 4.7.D) the generalized mean is useful for determining redundant features.

In the *hybrid connective*, high input values are allowed to compensate for low ones. For example, the *additive* and *multiplicative* γ-operators are defined pointwise with respect to a common argument, respectively, for fuzzy sets whose membership functions are m_A and m_B, as weighted arithmetic and geometric means of any fuzzy set union and intersection:

$$m_A \oplus_\gamma m_B = (1-\gamma)\cdot(m_A \cap m_B) + \gamma\cdot(m_A \cup m_B) \quad ;\text{ and} \qquad (4.104a)$$

$$m_A \otimes_\gamma m_B = (m_A \cap m_B)^{(1-\gamma)} \cdot (m_A \cup m_B)^{\gamma} \quad . \qquad (4.104b)$$

Both of these operators can act as a pure intersections or unions at the extremes of the parameter γ: $\gamma = 0$ gives the intersection, while $\gamma = 1$ gives the union in both connectives. But these families of connectives also allow the intersection and union to compensate for each other when $0 < \gamma < 1$. Thus γ can be regarded as the parameter that controls the degree of compensation afforded by its connective. Any union and intersection operator can be used in equations (4.104); see Dubois and Prade (1985) or Klir and Yuan (1995) for more extensive discussions on this point.

For pattern recognition applications, the aggregation operators in (4.103) and (4.104) are often used as integrator functions in Type I fuzzy neurons. The definitions given below are for individual membership values of the inputs, which can also be interpreted as degrees of satisfaction of some criteria for class labeling.

Zimmermann and Zysno (1983) introduced an exponentially weighted multiplicative hybrid operator that they called the *multiplicative γ-model.* To write the formula in the style of a Type I neuron we add the *exponential* weight vector, $\mathbf{w} = (w_1 \cdots w_p)^T$ to equation (4.102),

$$\Phi_M(\mathbf{x}, \mathbf{m}_{TI}, \mathbf{w}) = (\prod_{i=1}^{p} m_i(x_i)^{w_i})^{1-\gamma} \cdot (1 - \prod_{i=1}^{p} (1 - m_i(x_i))^{w_i})^{\gamma}, \text{ with} \quad (4.105)$$

$$\sum_{i=1}^{p} w_i = p \text{ and } 0 \le \gamma \le 1 \quad . \quad (4.106)$$

Here w_i is the weight associated with input x_i and is related to the importance of x_i. The degree of compensation between the union and intersection parts of the operator is controlled by $\gamma \in [0,1]$. The parts of this connective are not strictly unions and intersections (the exponential weights prevent them from being commutative). However, the factors in (4.105) function in much the same way. The summation in (4.106) insures that the "union" part is always larger than the "intersection" part. Krishnapuram and Lee (1988, 1989, 1992a, 1992b) studied some properties of Type I neurons that used the generalized mean, the γ-model, and Yager's (1980) union and intersection operators. These authors developed back-propagation training algorithms for FFBP networks that used all these neurons.

The *additive γ-model* neuron is defined as:

$$\Phi_A(\mathbf{x}, \mathbf{m}_{TI}, \mathbf{w}) = (1-\gamma) \cdot (\prod_{i=1}^{p} m_i(x_i)^{w_i}) + \gamma \cdot (1 - \prod_{i=1}^{p} (1 - m_i(x_i))^{w_i}) \quad (4.107)$$

While this γ-model incorporates weights that can be estimated with training methods, increased flexibility can be obtained by replacing the fixed multiplicative union and intersection parts with a parametrized family of union and intersection operators, for example, Yager operators (Yager, 1980). These union and intersection connectives can be inserted directly in equation (4.107). However, in order to match more closely the way in which a traditional neural network handles inputs and weights, we can also incorporate exponential weights into the new operators. Using (4.104a), the output value of the Type I fuzzy neuron takes the form

$$\Phi_Y(\mathbf{x}, \mathbf{m}_{TI}, \mathbf{w}) = (1-\gamma) \cdot y_1 + \gamma \cdot y_2 \qquad \text{, where} \qquad (4.108a)$$

$$y_1 = 1 - (1 \wedge \left[\sum_{i=1}^{p} \left(1 - m_i(x_i)^{w_i}\right)^{\beta}\right]^{\frac{1}{\beta}}, \ \beta \in [0, \infty) \qquad \text{, and} \qquad (4.108b)$$

$$y_2 = 1 \wedge \left[\sum_{i=1}^{p} \left(m_i(x_i)^{w_i}\right)^{\beta}\right]^{\frac{1}{\beta}}, \ \beta \in [0, \infty) \qquad \text{, with} \qquad (4.108c)$$

Since the weights $\{w_i\}$ are tunable, such neurons may be realized using the fuzzy neurons of Yamakawa et al. (1992) with a different aggregation function than they used. The fuzzy neuron of Yamakawa et al. has a membership function and a real weight associated with each input connection. Strictly speaking, neurons with hybrid operators or the fuzzy neurons of Yamakawa et al. are not of Type I. The additive hybrid operator at (4.108) is constructed from a Yager union and intersection (1980) of the exponentially weighted inputs. In this case, the constraint $\sum_{i=1}^{p} w_i = p$ (which ensures that the union part is always greater than the intersection part in the multiplicative γ-model of (4.105)) is no longer needed. All that is required is that each weight is greater than or equal to zero (Keller et al., 1994b). The $m_i(x_i) \in [0,1]$ are the inputs or criteria to be aggregated, w_i represents the weight associated with the input $m_i(x_i)$ and is related to the importance of that input, and $\gamma \in [0,1]$ controls the degree of compensation between the union and intersection parts of the operator.

Additive hybrid Type I fuzzy neurons are studied in Keller and Chen (1992a, b) and Keller et al. (1994b), and training algorithms similar to those for multiplicative hybrids are developed in these papers. We will not include the details of the training algorithms since they are essentially back-propagation, except with more complicated partial derivatives. To reduce the complexity of the derivatives, Keller and Yang (1995) modified the Yager operators to include multiplicative instead of exponential weights. As will be seen in the example that ends this section and the examples in the next section (as well as those in Chapter 5), the actual form of the Type I fuzzy neuron does not have a particularly strong affect on the overall quality of approximation provided by the associated fuzzy neural network. The speed of training and semantic interpretation of the nodes and weights are where the advantage of this type of fuzzification of NNs really lies.

Hayashi et al. (1991) develop training algorithms for similar hybrid operators (non weighted inputs, with other choices for the union and intersection parts), for use in an information retrieval scheme. This is similar to the use of additive hybrid operators in Keller et al., (1994b), which concerns itself with general decision making.

Example 4.23 This example considers the training of a single neuron to approximate the IO relationship of an empirical data set which is conjunctive in nature. The operator and data were studied in (Thole and Zimmermann, 1979). The first two columns of Table 4.40 show the input data, n = 20 vectors $\mathbf{x} = (x_1, x_2)^T \in \Re^2$, while column 3 displays the desired output d($\mathbf{x}$). We call this IO data X_{ZZTop}.

Table 4.40 X_{ZZTop} and outputs of single neuron approximations

x_1	x_2	d($\mathbf{x}$)	Φ_M	Φ_A	Φ_Y	$\Phi_{M_{-1.02}}$	Φ^1_{LH}	Φ^5_{LH}
0.00	0.99	0.01	0.02	0.19	0.05	0.00	0.31	0.30
0.91	0.42	0.52	0.59	0.50	0.56	0.57	0.55	0.57
0.22	0.15	0.17	0.10	0.09	0.07	0.18	0.09	0.07
0.55	0.80	0.67	0.61	0.53	0.65	0.66	0.57	0.57
0.02	0.45	0.01	0.06	0.09	0.03	0.04	0.11	0.09
0.50	0.44	0.49	0.38	0.32	0.45	0.47	0.30	0.29
0.69	0.40	0.54	0.46	0.39	0.51	0.51	0.40	0.40
0.85	1.00	1.00	0.91	0.87	0.89	0.92	0.82	0.85
0.42	0.62	0.46	0.43	0.36	0.49	0.51	0.36	0.36
0.32	0.21	0.14	0.17	0.14	0.19	0.25	0.13	0.11
0.48	0.31	0.40	0.30	0.25	0.35	0.38	0.23	0.21
1.00	0.00	0.00	0.03	0.19	0.05	0.00	0.35	0.35
0.63	0.34	0.44	0.39	0.32	0.43	0.44	0.32	0.31
0.28	0.45	0.24	0.26	0.22	0.31	0.35	0.20	0.18
0.13	0.51	0.10	0.17	0.16	0.19	0.21	0.16	0.14
0.33	0.24	0.30	0.18	0.16	0.21	0.28	0.14	0.12
0.97	0.26	0.33	0.48	0.40	0.41	0.40	0.49	0.50
0.48	0.01	0.02	0.05	0.10	0.03	0.02	0.12	0.10
0.55	0.96	0.71	0.69	0.60	0.66	0.70	0.65	0.67
0.13	0.98	0.19	0.31	0.28	0.26	0.23	0.39	0.38
MSE			0.006	0.014	0.004	0.003	0.025	0.025
Epochs			42	73	15	29	1000	5000

Three Type I fuzzy neurons, Φ_M (4.105), Φ_A (4.107), Φ_Y (4.108), the generalized mean neuron Φ_{M_q} (4.103a), and one standard neuron, Φ_{LH}, were trained via back-propagation to approximate this data - that is, weights were sought during training that minimized $MSE_{\Phi_*}(X_{ZZTop}) = \sum_{k=1}^{20}\left(\Phi_*(\mathbf{x}_k,*) - d(\mathbf{x}_k)\right)^2 / 20$.
For the generalized mean Φ_{M_q}, the initial weights and exponent were $w_1 = w_2 = 0.5$, $q = 1$, and as shown in Table 4.40, the final value of q after learning is q= -1.02. The exponential weights started at 1.0 and the initial value of γ was 0.5. The standard McPitts neuron was trained for 1000 (Φ^1_{LH}) and 5000 (Φ^5_{LH}) epochs. All training runs were terminated when the maximum change in any parameter was less than 0.0001.

The outputs upon resubstitution are shown in Table 4.40. The mean squared errors for all six neurons are similar. The fuzzy neurons (we include the generalized mean as a fuzzy neuron, and notice that it enjoyed the smallest MSE of the lot) enjoy a slight advantage in terms of smaller training errors over either of the McPitts neuron. We think the real payoff, however, is that the number of epochs needed to reach the error rates shown was much less for the fuzzy neurons than the standard ones. Interestingly, the MSE of the standard neuron stabilized at 0.0248 (rounded to 0.025 in the table) in about 3500 epochs - that is, the MSE for Φ^5_{LH} did not decrease in the last 1500 passes through X_{ZZTop}. Of course, it is possible that if training were extended, or if a smaller learning rate, or a different initialization were used, this level might have decreased.

Table 4.41 Final parameters of the fuzzy Type I neurons

Neuron	γ (or q)	w_1	w_2
Φ_M	0.46	1.04	0.96
Φ_A	0.19	1.04	0.96
Φ_Y	0.05	0.65	0.62
Φ_{M_q}	-1.02	0.49	0.51

Table 4.41 displays the final values of the parameters of the Type I fuzzy neurons. Both Φ_A and Φ_Y ended with a γ near zero, indicating that the overall aggregation was intersection-like. The negative value for q in the generalized mean indicates that it acts here as an intersection-like operator (tending slightly towards the min = T_3 norm and just a little to the "left" of the harmonic mean, which is

realized at q = -1). It is interesting that Φ_M remained almost completely compensatory (γ = 0.46). Our ability to interpret the nature of the neuron (e.g., intersection-like) is sometimes offered as an additional advantage of using fuzzy neurons. It's hard to draw much stronger conclusions from this simple example, but it gives you an idea of one way that fuzziness can be injected into the atomic unit of a neural network.

D. Fuzzy aggregation networks

When the Type I fuzzy neurons from Section 4.7.C are put into a network structure, the resultant configuration has been called a *fuzzy aggregation network*, or FAN (Krishnapuram and Lee, 1988, 1989, 1992a, 1992b, Keller et al., 1994b, Keller and Chen, 1992a). These networks have the advantage that, after training, the nodes can be interpreted as "mini-rules", i.e., there is a higher degree of transparency in what the network learned than is available in traditional neural networks. There are many variations of the material presented in this section - too many to cover in any logical fashion. What we hope to do instead is show, by example, how you might develop a fuzzy network approach that is tailored to the application you are interested in.

With this approach, the decision process can be viewed as a hierarchical network, where each node in the network "aggregates" the degree of satisfaction of a particular criterion from the observed evidence. The inputs to each node are the degrees of satisfaction of each of the sub-criteria, and the output is the aggregated degree of satisfaction of the criterion. Such networks can be utilized to address the object classification problem, and as we shall see in Chapter 5, they are quite effective for image segmentation, which is also just a pattern recognition problem "in the small".

The classification problem using the fuzzy aggregation net framework reduces to : (i) determining the structure of the aggregation network to be used; (ii) determining the nature of the connectives at each node of the network; and (iii) computing the input supports (degrees of satisfaction of criteria) based on observed features (This should "ring-a-bell"! It's a critical step). The structure of the aggregation network depends on the problem at hand. Krishnapuram and Lee (1988, 1989, 1992a, 1992b), and Keller et al. (1994b) developed learning procedures based on back-propagation, so that both the type of connective at each node, as well as the parameters associated with the connective, can be learned from training data. Besides general pattern recognition and decision making problems, these fuzzy aggregation networks have also been used in network structures for fuzzy logic inference (Keller et al., 1992, 1994c).

Example 4.24 In this example from Krishnapuram and Lee (1992a, b), we consider the problem of recognizing two classes based on four features. The features for each class were generated using a multivariate Gaussian probability density distribution. The mean and variance of the first two features in both classes were the same. The third and fourth features had different means and variances in each class. A total of 121 sets of features were generated for each class. Out of these 242 vectors, approximately 90% were used for training and 10% were used for testing, repeating this process 10 times (10-fold cross validation).

From the training data, the mean and variance of each feature in each class were calculated. This gives 8 means and 8 variances. The membership value (or the degree of satisfaction of the criterion) of each feature in the two classes was calculated assuming a Gaussian membership function. Specifically, the membership value m_{ij} of x_i (the ith feature) in class j was given by

$$m_{ij}(x_i) = \exp^{-((x_i-\mu_{ij})^2/(2\cdot\sigma_{ij})^2)} \quad , \qquad (4.109)$$

where μ_{ij} and σ_{ij} are the sample mean and the standard deviation of the ith feature in the jth class. This gives two membership values for each feature (one for each class) or a total of 8 membership values per data vector. These are the input membership functions on each edge of the input side of the two fuzzy Type I output nodes used in this example.

Krishnapuram and Lee (1992a) used a single hidden layer aggregation network with 8 nodes (h1, h2, ..., h8) and 2 fuzzy Type I output nodes (o1 and o2) for this classification problem. This network is shown in Figure 4.78(a) with inputs at the bottom of the sketch. Notice that this is not a fully connected network - feature value x_1 is distributed only to h1 and h2, x_2 to h3 and h4, etc. Here nodes h1 and h2 in the hidden layer tell us to what extent feature 1 supports class 1 and class 2, etc. In the training phase, the desired value of the output nodes o1 and o2 were chosen to be equal to 1 and 0 respectively, if the data point came from class 1, and they were chosen to be equal to 0 and 1 respectively, if the data point came from class 2.

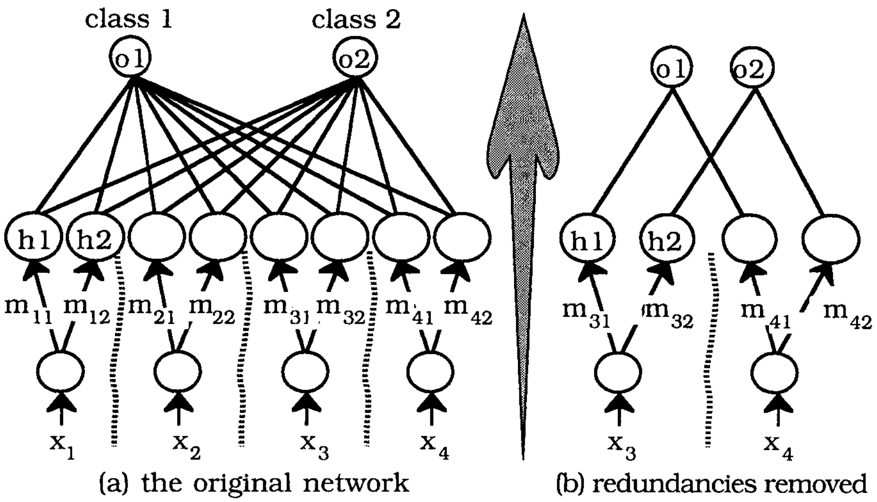


Figure 4.78 Fuzzy aggregation network for a two class problem

A modified gradient descent algorithm (Krishnapuram and Lee, 1992a, 1992b) was used (to account for the constraints on the weights) with the multiplicative γ-model as the aggregation operator for training the parameters γ and δ associated with o1 and o2. As the training proceeded, the weight values associated with all connections emanating from the first four nodes of the hidden layer gradually decreased toward zero. Four other weights also dropped towards zero, producing the much simpler network shown in Figure 4.78(b). The final parameter values were: for node o1, $\gamma = 0.922$, $w_1 = 4.66$, $w_2 = 3.34$; for node o2, $\gamma = 0.923$, $w_1 = 4.64$, $w_2 = 3.36$. This indicates that these features were redundant. Krishnapuram and Lee state that this suggests that the weights associated with x_1 and x_2 should be reduced relative to other weights in the network. Constraints on the weights such as the ones in (4.103b) or (4.106) are crucial if this is to be achieved.

The overall rate of correct classification for ten-fold cross validation was 92%, the same performance attained by an optimal Bayes classifier trained with the standard mixture model equations (which in this case are decoupled and can be computed non-iteratively because the data have crisp training labels). Thus, this aggregation scheme performs as well as a Bayesian classifier on data that match the assumptions for an optimal Bayesian design.

Aggregation networks using Φ_M as the node function in Type I fuzzy neurons have been successfully applied to a variety of problems (both two-layer and multi-layer) including the determination of creditworthiness, and recognition of tanks, armored personnel carriers and false alarms with excellent results (Krishnapuram and Lee, 1989, 1992a, 1992b).

One important aspect of the training procedure for aggregation networks is that the resulting networks can be interpreted linguistically, since the final parameters allow us to loosely characterize each node as a conjunction, disjunction or mean of the values being aggregated. Based on the parameter values, it is even possible to say something about the strength of the operation, i.e., that the node is a strong or weak conjunction of evidence. For example, the generalized mean behaves like an intersection (union) operator for negative (positive) values of q. Similarly, Φ_M and Φ_A behave like intersection (union) operators for values of γ close to 0 (1). Interpretation of these operators can provide insight into the nature of the decision process, and into the nature of the training data itself. This could be used to advantage, for example, if the designer had the information necessary to heuristically tune the performance of the network. We will see "mini rules" such as these used for image segmentation in Example 5.10 in Chapter 5.

Another important aspect of these networks is that they can be used to identify redundant features. Krishnapuram and Lee (1989, 1992a, 1992b) define three kinds of redundant features: (i) uninformative, (ii) unreliable, and (iii) superfluous. Uninformative features are those whose values are approximately the same in all feature vectors. Unreliable features are those whose means and variances are roughly the same in all classes. And finally, superfluous features are those that are highly correlated. Features that have the first two of these three characteristics are not very useful for classifier design. Features with the third characteristic can increase the reliability of a classifier, since they all provide similar information. Krishnapuram and Lee (1989, 1992a, 1992b) use numerical experiments to support their assertion that aggregation networks using node functions based on either Φ_{M_q} or Φ_M can eliminate uninformative and unreliable features, as indicated by the training weights for such features tending towards 0. Example 4.24 shows an instance where unreliable features (1 and 2) are eliminated this way.

Because the weights for Yager-type nodes (Φ_Y) are exponents of feature values which lie in the interval [0,1], and because γ determines the "mixing" of union and intersection components for the desired node, it is possible to detect features which do not contribute to the decision using Φ_Y as well. The farther away from 1.0 a weight becomes during training, the less impact that feature has in the combination of values made at the node. When γ is greater than 0.5, the node leans more towards a union, and so, large exponents tend to signify unimportant features, while for more intersection-like nodes ($\gamma < 0.5$), a small exponent keeps a feature from contributing to the node output (Keller and Chen, 1992b). This allows features in the training data to be investigated as to their potential for contributing to the final aggregation.

We end this section with an example of a two level network showing how fuzzy aggregation networks can be used to solve recognition (or decision) problems in high level computer vision. In one sense, this use is similar to employing neural networks to model fuzzy rules, a technique investigated by several authors (Wang and Mendel, 1992, for example). We include this example in Chapter 4 because the discovered "rules" are object recognition decisions - that is, the network is performing classification, and hence, is a classifier.

Example 4.25 This example combines some results discussed in (Krishnapuram and Lee, 1992a) and Keller et al. (1994b). We will show how the multiplicative and additive hybrid operators can be used in a two layer network to simulate a classification problem in computer vision. The goal is to recognize a stool from an arbitrary viewpoint. The stool is assumed to have four cylindrical legs and a top that can be square or circular. However, depending on the viewpoint, the top may be perceived as a parallelogram or an ellipse. The strategy here is that a strong fuzzy classification should result if either group of features (legs or top) is present. The two layer network shown in Figure 4.79 was used by both (Krishnapuram and Lee, 1992a) and Keller et al. (1994b).

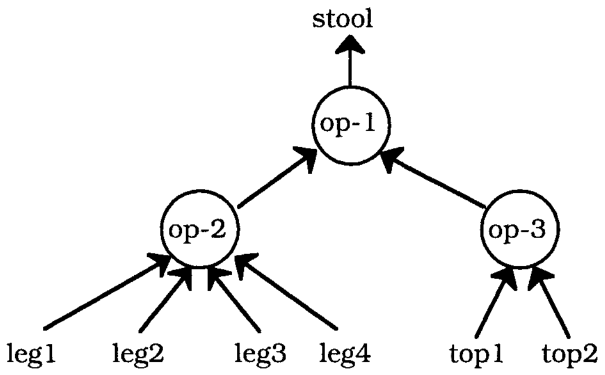


Figure 4.79 Aggregation network or "stool recognition" problem

In the first layer, the simulated evidence associated with the four legs is combined to make a hypothesis at op-2, and the simulated evidence supporting the shape of the top is combined at op-3. In the second layer, the hypothesis concerning the existence of the stool is made at op-1 by aggregating the evidence coming from op-2 and op-3. Different aggregation operators such as Φ_M, Φ_A and Φ_Y were used for op-1, op=2 and op-3 ("op" stands for operator in this example). The training data set was constructed synthetically using the following decision strategy.

IF	all four legs exist in the current view
OR IF	there is one of the two possible types of tops
THEN	accept the hypothesis that the object is a stool.

Because of the way the desired outputs were assigned in the synthetic data, it is expected that after training, the connectives for op-1, op-2, and op-3 will be disjunctive, conjunctive, and disjunctive, respectively. If we use Φ_M, Φ_A or Φ_Y for the three nodes, the three γ values at the nodes γ_{stool}, γ_{legs} and γ_{top} should be large, small, and large, respectively. A subset of the synthetic input data, vectors $\mathbf{x} \in \Re^6$, together with their desired output values d($\mathbf{x}$), is shown in Table 4.42. Only a subset of the 48 input/output tuples are shown. The entire data set, which satisfies the above stipulations, consists of n = 48 IO pairs which are all symmetric combinations of those listed in Table 4.42.

Table 4.42 Sample inputs and desired output d(x)

leg1 $m_1(x_1)$	leg2 $m_2(x_2)$	leg3 $m_3(x_3)$	leg4 $m_4(x_4)$	top1 $m_5(x_5)$	top2 $m_6(x_6)$	d($\mathbf{x}$)
0.10	0.10	0.10	0.10	0.10	0.10	0.01
0.10	0.10	0.10	0.10	0.10	0.90	0.90
0.10	0.10	0.10	0.90	0.10	0.10	0.05
0.10	0.10	0.10	0.90	0.10	0.90	0.93
0.10	0.10	0.90	0.90	0.10	0.10	0.10
0.10	0.10	0.90	0.90	0.10	0.90	0.95
0.10	0.90	0.90	0.90	0.10	0.10	0.20
0.10	0.90	0.90	0.90	0.10	0.90	0.98
0.90	0.90	0.90	0.90	0.10	0.10	0.98
0.90	0.90	0.90	0.90	0.10	0.90	0.99

The results of training (i.e., resubstitution errors incurred after training) are displayed in Table 4.43. In this example, the parameters were initialized as follows: γ was set to 0.5, all of the w_i's to 1.0 and β (for the Yager connectives) to 2.0. All training runs were terminated when the maximum change in any parameter was less than 0.0001.

Table 4.43 Desired output and results using 5 networks

d(**x**)	Φ_M	Φ_A	Φ_Y	NN5	NN20
0.01	0.18	0.19	0.04	0.004	0.02
0.90	0.90	0.90	0.90	0.90	0.94
0.05	0.18	0.19	0.08	0.03	0.05
0.93	0.93	0.90	0.96	0.93	0.92
0.10	0.19	0.20	0.08	0.07	0.07
0.95	0.90	0.90	0.96	0.96	0.95
0.20	0.26	0.25	0.20	0.27	0.23
0.98	0.91	0.91	0.96	0.97	0.98
0.98	0.74	0.71	0.96	0.79	0.93
0.99	0.97	0.97	0.96	0.99	1.00
Epochs	385	5000	1248	5000	20000
MSE	0.0059	0.0068	0.0004	0.0016	0.0003

For comparison, we trained a standard FFBP network with configuration 6:3:1 where the parameters of the logistic functions were fixed at $\lambda = 1$, $\beta = 0$, so the parameters acquired during learning in this example are the weight vectors $\{\mathbf{w}_i\}$ and bias constants $\{\alpha_i\}$. We trained this network to 5000 and 20000 iterations through the training data. The outputs of these two networks are in Table 4.43 in columns labeled "NN5" and "NN20", respectively. The final parameters for the fuzzy aggregation networks were as follows (the weights $\{w_i\}$ are numbered from left to right):

Network of Φ_M neurons

op-1: $\gamma_{stool} = 1.0$; $w_7 = 1.03$; $w_8 = 0.97$

op-2: $\gamma_{legs} = 0.06$; $w_1 = 1.00$; $w_2 = 1.00$; $w_3 = 1.00$;. $w_4 = 1.00$

op-3: $\gamma_{top} = 1.0$; $w_5 = 1.00$; $w_6 = 1.00$

Network of Φ_A neurons

op-1: $\gamma_{stool} = 0.99$; $w_7 = 0.985$; $w_8 = 1.015$

op-2: $\gamma_{legs} = 0.00$; $w_1 = 0.999$; $w_2 = 0.999$; $w_3 = 1.059$; $w_4 = 0.944$

op-3: $\gamma_{top} = 0.99$; $w_5 = 1.000$; $w_6 = 1.000$

Network of Φ_Y neurons

op-1: $\gamma_{stool} = 0.96$; $\beta = 0.41$; $w_7 = 1.172$; $w_8 = 1.227$

op-2: $\gamma_{legs} = 0.03$; $\beta = 1.94$; $w_1 = 0.98$; $w_2 = 0.98$; $w_3 = 0.99$; $w_4 = 0.97$

op-3: $\gamma_{top} = 0.92$; $\beta = 2.24$; $w_5 = 1.639$; $w_6 = 1.639$

As can be seen, for all three fuzzy models, op-1 and op-3 turned out to be primarily union connectives, while op-2 leaned toward intersection. The results in all cases matched the desired values pretty well, especially considering the small amount of training data.

Furukawa and Yamakawa (1998) recently proposed a 4 layer feed-forward fuzzy NN which can extract local features (structural information) directly from inputs such as handwritten characters and employ them for recognition. The interesting aspect of this method is that each layer uses a different type of fuzzy neuron and each performs a different task. The first layer gets the local features from the input image. The second layer filters off dispensable features and integrates the local features obtained by the first layer into more global features. The third layer compresses the size of the map of local features. The fourth layer is self-organized by learning and gives similarities of the input image to all of the learned images in the output layer.

E. Rule extraction with fuzzy aggregation networks

There are many methods to generate fuzzy (and crisp) rules automatically from training data through the use of fuzzy network structures (Wang and Mendel, 1992, Berenji and Khedhar, 1993, Lin and Lee, 1996; Jang et al., 1997, Lin and Cunningham, 1995). We discuss a method developed by Krishnapuram and Rhee (1993a) which uses the FAN from section 4.7.D to induce a set of fuzzy rules which are used for classification (see Example 5.10). This technique is fairly general, and can be applied to any classification problem.

Krishnapuram and Rhee (1993a) describe an automatic rule generation procedure which they used for supervised image segmentation (i.e., pixel classification). The procedure consists of the following three stages: estimation of the class membership functions $\{m_{ij}\}$, where $m_{ij}(x_i)$ represents the membership value of feature i for each class j; estimation of the membership functions $\{m_{ik}^{\mathcal{R}}\}$ of the linguistic labels to be used in rule base $\mathcal{R}$ to describe each feature i; and generation of the rules in $\mathcal{R}$ that best describe the training data.

Suppose there are p features and c classes. In the first stage, Krishnapuram and Rhee use the smoothed histogram of each feature in each class to generate the membership functions $\{m_{ij}$, i=1,..., p; j=1,...,c}. The smoothed histograms $\{m_{ij}\}$ play the role of the Gaussian membership functions used in Example 4.24 (see (4.109)). Krishnapuram and Rhee use a network similar to the one in Figure 4.78 to eliminate uninformative and unreliable features. The generalized mean was used as the node function in the output nodes. At the end of this stage the remaining features are used in the rule generation process.

The first step in the second stage is to generate the membership functions $\{m^R_{ik}\}$ for the various linguistic values (such as *low*, *medium* and *high*) that each non-redundant feature can take. This is done by first computing a smoothed histogram of a given feature. Unlike the computation of m_{ij}, data from all classes are used for this purpose. This process generates p smoothed histograms. Each of these histograms is then approximated in terms of a weighted sum of Gaussians. Krishnapuram and Rhee (1993a) use a polynomial fit to the histogram to determine the number of peaks in the histogram; this information is used to establish the number of Gaussian functions needed, as well as their initial means and covariance matrices. Then, a gradient descent procedure that minimizes the error between the smoothed histogram and the weighted sum of Gaussians is used to fine tune estimates for the means and variances of the Gaussians.

Each Gaussian function that appears in the weighted sum approximation of the feature i histogram is treated as the membership function of a linguistic label associated with feature i. The membership values for an observed feature value in each of the labels can be calculated using these membership functions. The final step is to obtain a compact set of rules with conjunctive and disjunctive antecedent clauses. To achieve this, Krishnapuram and Rhee use a three-layer fuzzy aggregation network. They initially start with an approximate structure for the aggregation network which is then pruned after training.

Figure 4.80 shows the structure of the approximation network for generating rules (jump ahead to Example 5.10 if you want to see this approach used in an actual problem). In the approximation network, the bottom layer consists of p groups of nodes, with the i-th group consisting of r_i nodes, where r_i is the number of linguistic values (granularity) defined on the i-th feature. We denote the linguistic values associated with feature i by $\ell_{i1},\ldots,\ell_{ir_i}$. Node k of group i (which is associated with ℓ_{ik}) in the bottom layer uses the membership function m^R_{ik} of the linguistic label (which is a Gaussian function determined in the previous stage) as the activation function.

The hidden layer consists of p groups of c nodes each. The jth node in group i in the bottom layer is connected to the kth node in the corresponding group in the hidden layer if a small α-cut (e.g., 0.05) of m^R_{ij} has a non-empty intersection with the support of m_{ik}. The rationale behind this connection is that if the support of the membership function of a linguistic label has no intersection with m_{ik}, then it cannot appear in the antecedent clause of a rule that describes class i (remember that m_{ik} is the smoothed histogram of feature i values from class k). This connection process is repeated for all the groups in the bottom layer.

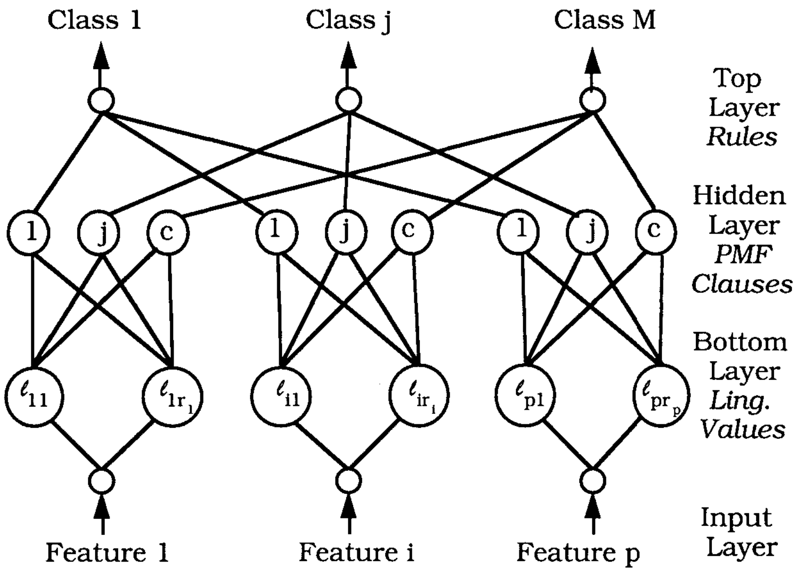


Figure 4.80 An approximation network for generating rules

The kth node of all groups in the hidden layer is connected to the kth node of the top layer for k = 1,...,c. All hidden and top-layer nodes use a suitable fuzzy aggregation operator such as the generalized mean or the γ-model as the activation function. The i-th feature (x_i) is fed to the i-th group of bottom-layer nodes as input. This completes the construction of the initial fuzzy aggregation network for this method.

The target values (crisp class labels) in the training data are chosen to be 1 for the class from which the training data was extracted, and 0 for the remaining classes. The aggregation operators (such as the generalized mean) used in the hidden and top layers have weights associated with each of their inputs. Each node in the hidden layer represents a combination of atomic premise clauses (e.g., feature i is *low* and feature j is *high*). However, the nature of the combination depends on the aggregation operator (e.g., generalized mean) chosen, and is not necessarily as in (4.72a).

As the network is trained, the weights corresponding to redundant antecedent clauses in the hidden layer become insignificant. This happens because typically there is a constraint on the weights. Each node in the top layer in Figure 4.80 represents a combination of rules for a class. The weights for redundant rules also become insignificant during training. The network is then pruned by removing all connections with very low weights (e.g., < 0.0001); the thresholds chosen in the top layer are usually data dependent.

The resulting network is interpreted as a set of fuzzy decision rules. The nodes in the hidden and top layers can represent either conjunctive, disjunctive or compensatory nodes, depending on the final values of the parameters of the aggregation function. Also, the connection weights determine the relative importance of the antecedent clauses in a rule. Since all the rules are determined simultaneously, an optimal set of rules is obtained, as opposed to individually optimal rules that would result from a serial process. In the notation of Section 4.6.D, the final network represents the rule base $\mathcal{R}$. Another attractive feature of this method is that it automatically identifies redundant features in the first stage. For example, this method eliminates the first two features of Iris (Rhee, 1993, Krishnapuram and Rhee, 1993a), leaving the third and fourth features (scatterplotted in Figure 4.12) to support the classifier. The rules discovered for Iris look like this:

R_1 : IF feature 3 is *low* OR feature 4 is *low*
THEN class = Sestosa

R_2 : IF feature 3 is *med* OR feature 4 is *med*
THEN class = Versicolor

R_3: IF feature 3 is *high* OR feature 4 is *high*
THEN class = Virginica

The structure just described, like ANFIS (Jang et al., 1997) and many others, is applicable for many types of data and problems - both in classifier design and elsewhere. Example 5.10 in chapter 5 illustrates this approach to learn a small set of fuzzy rules for an image segmentation problem.

4.8 Adaptive resonance models

Competitive learning models (besides the ones already discussed in Section 4.3) have been studied by many researchers (Grossberg, 1976a, 1976b, Rumelhart and McClelland, 1982, Carpenter and Grossberg, 1987a, Rumelhart and Zipser, 1985). This section is about Carpenter and Grossberg's *adaptive resonance theory* (ART) and some fuzzy relatives of it.

The original model was called ART1 by Grossberg (1976a, b). There are some interesting parallels between the evolution of crisp decision trees and ART1. Like Quinlan's ID3 (Section 4.6), ART1 was developed in a somewhat broader context than clustering or classifier design; Quinlan was interested in rule extraction for semantic explanations of rule-based decisions, while Grossberg wanted to mimic rudimentary connections believed to operate in our biological neural networks. When people began to use these two crisp models strictly for pattern recognition purposes, both were found deficient because both were created for a special type of data (ID3 for any discretely valued inputs, and ART1 for binary inputs, which are also discretely valued, with only two values) that are relatively rare in everyday pattern recognition. So, both developers generalized their original designs to accommodate continuously valued features : Quinlan's 1983 ID3 was imbedded in his 1993 C4.5; while Grossberg's 1976 ART1 was generalized to ART2 in Carpenter and Grossberg (1987b).

ART1 and many of its unsupervised relatives are nothing more than sequential point prototype generators, so a logical place for this subsection from the pattern recognition point of view would be somewhere in Section 4.3. On the other hand, unlike some of the CL models discussed in that section, many investigators besides Grossberg have invested substantial effort in connecting ART to (presumed) elements of the biological neural network, so we decided to defer this section until after our brief discussion about the BNN in case this aspect of ART interests you. Our aim here is to make sure that you understand the basic structure of ART1 from the pattern recognition viewpoint. Following Moore (1988), we separate the algorithmic component of ART1 from its architectural design, and present only the algorithmic aspects of ART1, Grossberg's (1976a, b) original model.

A. The ART1 algorithm

Recall from Section 4.3.A that any c point prototypes $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_c\} \subset \Re^p$ can be substituted into (2.6a), and the result is the crisp partition $U_{np}(\mathbf{V})$, the nearest prototype partition of X. As pointed out in Section 4.3.A, subsequently applying (2.6b) to the rows of $U(\mathbf{V})$ results in the sample means, $\mathbf{V} = \overline{\mathbf{V}}$, so it is not incorrect to regard the prototypes $\mathbf{V} = \overline{\mathbf{V}}$ as a *representation* of $U_{np}(\mathbf{V})$. Much of the ART literature uses this alternate way to describe crisp clusters in terms of their nearest prototypes, so we will follow this convention in this section.

Most competitive learning models suffer from a problem we can loosely call "unstable learning". Grossberg (1976b) proved a theorem about the competitive learning model described in Grossberg (1976a) which essentially states that if not too many input vectors are presented to the algorithm relative to the number of categories, or if the inputs do not form too many clusters, then the prototype that represents each class eventually stabilizes. This competitive learning model was also analyzed by Rumelhart and Zipser (1985), whose simulations confirmed Grossberg's theorem. However, non-frivolous counterexamples demonstrate that such competitive learning models cannot learn temporally stable prototypes in response to arbitrary inputs (Grossberg, 1987). For these counterexamples, system response to the same input data on successive presentations can be different due to prototype updates that take place in response to intervening data (Shih et al., 1992, Baraldi and Alpaydn, 1998). Consequently, the response to a given input pattern might never stabilize. Carpenter and Grossberg (1987a, 1988b) demonstrated several environments in which periodic presentation of just four inputs can cause instability.

Moore (1988) characterizes the *stability* of CL models in terms of two properties she calls stable$_1$ and stable$_2$. Specifically, a CL model is *stable*$_1$ in case no prototype returns to a previous position during training; and it is *stable*$_2$ when only finitely many prototypes are created during learning. The assumption for these two definitions is that there is an infinite supply of data. Stable$_1$ is a property possessed by individual prototypes, while stable$_2$ is a property of the entire prototype set $\mathbf{V}$.

Stability is one of two problems that Grossberg's ART1 was designed to address. The second problem was called *plasticity* by Grossberg. Plasticity refers to the ability of a CL model to learn new inputs after it has stabilized on previous training data. To understand both the problem and Grossberg's method of fixing it with ART1, suppose that a CL learning model has been running for a while, and its prototypes are fairly stable. Most of the CL models we have discussed so far (Section 4.3) use an update equation with the general form of equation (4.11), rewritten here to save you the trouble of looking it up:

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}), i = 1,\ldots, c; t=1,\ldots,T \quad . \tag{4.11}$$

The plasticity problem is related to the learning rate distribution in (4.11) - that is, the numbers $\{\alpha_{ik,t}\}$. In almost every case we know of, the standard method of achieving stability under (4.11) is to begin with values for the $\{\alpha_{ik,t}\}$ close to, but less than, 1; and then to decrease the $\{\alpha_{ik,t}\}$ towards zero as time (iteration number t) increases. The plasticity problem arises because smaller learning rates may disable the model's ability to respond appropriately to new inputs that have not been seen by the algorithm until the $\{\alpha_{ik,t}\}$ are small. To understand this, we assume that $\mathbf{v}_{\text{old}}$ is any prototype that will be updated with (4.11) for the current input $\mathbf{x}$, and rewrite (4.11) in a more suggestive form:

$$\Delta\mathbf{v} = (\mathbf{v}_{new} - \mathbf{v}_{old}) = \alpha(\mathbf{x} - \mathbf{v}_{old}) \quad . \tag{4.11'}$$

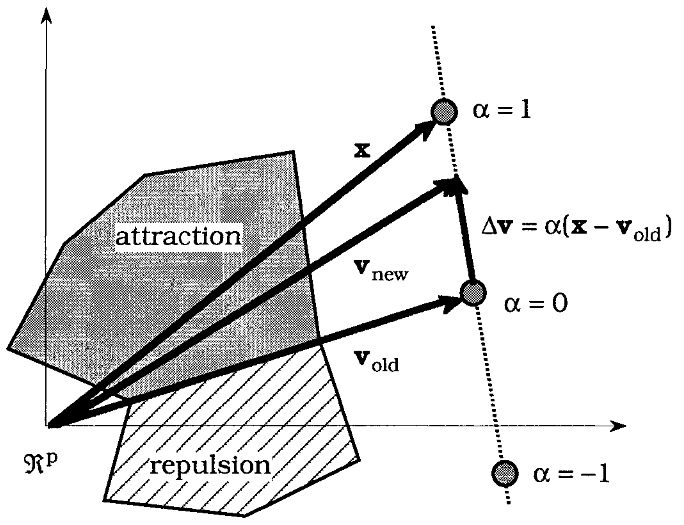


Figure 4.81 The update geometry of CL models that use (4.11)

Figure 4.81 illustrates the update geometry of (4.11'), and hence, of all the algorithms in Section 4.3 that use (4.11) to update prototypes. The vector $\Delta\mathbf{v}$ takes its general direction from the difference vector $(\mathbf{x} - \mathbf{v}_{old})$; its magnitude depends on the value of α; and the sign of α determines whether the update moves $\mathbf{v}_{old}$ towards $\mathbf{x}$ (attraction, the region "above" vector $\mathbf{v}_{old}$) or away from $\mathbf{x}$ (repulsion, the region "below" the vector $\mathbf{v}_{old}$).

When $\alpha = 1$, $\mathbf{v}_{new} = \mathbf{x}$; when $\alpha = 0$, $\mathbf{v}_{new} = \mathbf{v}_{old}$, i.e., the prototype is unchanged; and most importantly for plasticity, when α is positive but close to 0, $\Delta\mathbf{v}$ will be very small. Under these circumstances algorithms that use (4.11) to update their prototypes don't have much choice - they become stable as $\alpha_{ik,t} \to 0$ because $\Delta\mathbf{v}$ is so small that they can take only tiny steps. If the learning rates actually get to zero, the updates stop, and the prototypes are completely stable. Now suppose that $\alpha_{ik,t} \approx 0$, and that a new input arrives in the system that has not participated in the update scheme. The impact of this point on the locations of the $\{\mathbf{v}_i\}$ may be insignificant, even though the new input itself is importantly related to the structure of the overall input data. This is the plasticity problem, and it tugs CL models in the opposite direction from stability.

ART1 is motivated by this so-called *stability-plasticity dilemma* of competitive learning (Carpenter and Grossberg, 1987a). Apparently, the best situation would be if the CL system could switch between the plastic and stable states and vice-versa as the need arose. Such characteristics can be built into a network by adding a feedback mechanism between the competitive layer and the input layer. This philosophy has resulted into two well known prototype generation architectures, ART1 (permits only binary inputs) and ART2 (suitable for analog / gray level inputs). In ART-type networks outputs of the processing elements reverberate back and forth between layers, resulting in a stable oscillation when proper prototypes develop - a kind of resonance - hence the name ART. Study of the structure that achieves this takes us into the architectural details of ART networks, which is not covered in this book., We will follow Moore (1988) by presenting a simple description of ART1 in the language of Section 4.3.

ART1 assumes that inputs are binary valued p-vectors, that is, input data have the form $\mathbf{x} = (x_1, \ldots, x_p)^T \in \{0,1\}^p$. While the general case is to assume an infinite input stream, we will always deal with finite data sets $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \{0,1\}^p$, $|X| = n$. ART1 uses two similarity measures between the binary input vectors and the prototypes it constructs. Let $\mathbf{x}_k$ be the current input vector, $\mathbf{x}_k \in \{0,1\}^p$, and let $\{\mathbf{v}_i\}$ be a set of c *binary-valued* prototypes (we shall see later that ART1 guarantees this). Define

$$s_1(\mathbf{x}_k, \mathbf{v}_i) = \frac{\langle \mathbf{x}_k, \mathbf{v}_i \rangle}{\beta + \|\mathbf{v}_i\|_1}, \quad i = 1, \ldots, c\,;\ \beta > 0 \qquad \text{, and} \qquad (4.110a)$$

$$s_2(\mathbf{x}_k, \mathbf{v}_i) = \frac{\langle \mathbf{x}_k, \mathbf{v}_i \rangle}{\|\mathbf{x}_k\|_1}, \quad i = 1, \ldots, c \qquad . \qquad (4.110b)$$

The closest prototype to $\mathbf{x}_k$ maximizes s_1, and for small values of β, (4.110a) is an estimate of the ratio of overlap between $\mathbf{x}_k$ and $\mathbf{v}_i$ (recall that these are binary-valued vectors, so the dot product simply computes the number of matches between $\mathbf{x}_k$ and $\mathbf{v}_i$) and the magnitude of the prototype. Using a small value of β helps with the "all zero prototypes" problem. This measure is sometimes called a *search* parameter in the ART1 literature.

Similarity measure s_2 is used to evaluate the extent to which $\mathbf{x}_k$ and $\mathbf{v}_i$ are matched: this number will range between 0 and 1, being 0 if there are no matches, and 1 if $\mathbf{x}_k \le \mathbf{v}_i$, where ordering of vectors is in the usual component by component sense. In other words, (4.110b) computes the fraction of matches between the input and the prototype. This measure is compared to a threshold ρ called the *vigilance* parameter. We will describe the role each of these measures plays in determining (nearest prototype) cluster shapes after we discuss the operation of the ART1 algorithm, which is summarized in Table 4.44.

Table 4.44 The ART1 algorithm

A. Training phase : find $\mathbf{V}$ without U_{tr}	
Store	Unlabeled binary-valued data $X \subset \{0,1\}^p, \lvert X \rvert = n$
Pick	☛ maximum number of iterations: T ☛ search parameter $\beta : 0 < \beta << 1$ ☛ vigilance parameter $\rho : 0 < \rho \le 1$
Iterate	see below

Iterate

```
V ← {x_1}
For t = 1 to T
    For k = 1 to n
        V' ← V
        REPEAT
            $s_1(\mathbf{x}_k, \mathbf{v}_i) = \underbrace{\max}_{\mathbf{v}_j \in \mathbf{V}'}\{s_1(\mathbf{x}_k, \mathbf{v}_j)\}$        (4.111a)
            $\mathbf{V}' \leftarrow \mathbf{V}' - \{\mathbf{v}_i\}$
            $\mathrm{IF}\left(\mathbf{V}' = \varnothing \text{ and } s_2(\mathbf{x}_k, \mathbf{v}_i) < \rho\right)$        (4.111b)
                Then $\mathbf{V} \leftarrow \mathbf{V} \cup \{\mathbf{x}_k\}$
            IF $s_2(\mathbf{x}_k, \mathbf{v}_i) \ge \rho$        (4.111c)
                Then $\mathbf{v}_i \leftarrow \mathbf{v}_i \wedge \mathbf{x}_k$ (bitwise AND)        (4.111d)
        UNTIL ($\mathbf{V}' = \varnothing$)
    Next k
Next t
```

B. Prototype relabeling of $\mathbf{V}$ with U_{tr} using, e.g., (4.13)

C. Optional (crisp) clusters if U_{tr} is unknown, with, e.g., (2.6a) :

$$u_{ik} = \begin{cases} 1; & \lVert \mathbf{x}_k - \mathbf{v}_i \rVert < \lVert \mathbf{x}_k - \mathbf{v}_j \rVert \text{ , } 1 \le j \le c, j \ne i \\ 0; & \text{otherwise. Resolve ties arbitrarily} \end{cases} \Bigg\} \forall i, k$$

The prototypes built by ART1 are accumulated in Table 4.44 using our standard notation - $\mathbf{V}$ is the set of prototypes at any point during training. Table 4.44 has the same general organization as Table 4.4 - it is set up so that if you have labels for the training data, these can be ignored in step A, and then used in step B to create labeled prototypes. Thus, ART1 can be used to design prototype classifiers just as we did with other CL models in Section 4.3. The usual specification of ART1 (e.g., Moore, 1988) does not give a termination criterion. Carpenter and Grossberg (1988b) show that ART1 terminates after a finite number of iterations (remember, an iterate is one pass through all of X, sometimes called one epoch) in the sense that no new clusters will be formed, and the prototypes of existing clusters will stop changing - a point at which ART1 is said to be *stabilized.* We have added an iterate limit T in Table 4.44 as a matter of practicality, since the finite number of passes needed to achieve stability for a particular data set is not known, and might be very large.

Many ART papers call s_1 a search criteria because it controls search through the current prototypes, beginning with the closest (winner). Equation (4.111a) shows that ART1, like LVQ and SHCM, *begins* as a winner take all CL model - it selects $\mathbf{v}_i$ - the closest prototype to input $\mathbf{x}_k$ - for *possible* updating. If there are no prototypes, the input is declared a new prototype (and hence, ART1 creates a new cluster). If there are prototypes, and the winner fails to achieve resonance, the "second best" (next closest prototype to the input) is tested by (4.111a); and so on, until one of the existing prototypes gets updated or, failing this, a new prototype is created. Consequently, the "winner" in ART1 - that is, the prototype that gets updated - is the one that exhibits maximum response among the subset of prototypes that satisfy the vigilance test. Thus, ART1 is a CL model which only updates one prototype per input, but not necessarily the "winning" one in our previous sense of the word winner as used, e.g., in connection with Kohonen's LVQ. Dynamic creation of new prototypes by ART1 seemingly frees it from the problem of how many to look for, but like the mountain and subtractive clustering methods of Section 4.3, the terminal value of c, the number of prototypes chosen by the model, depends implicitly on the choice of the ART1 parameters β and ρ.

At the beginning of ART1 there are no prototypes, so without loss, we initialize the prototype set by $\mathbf{V} \leftarrow \{\mathbf{x}_1\}$. Whenever ART1 creates a new prototype, it is one of the input vectors - in other words, the first instance of each prototype in $\mathbf{V}$ is a binary valued vector, $\mathbf{v} \in \{0,1\}^p$. Consider the prototype update equation $\mathbf{v}_i \leftarrow \mathbf{v}_i \wedge \mathbf{x}_k$ in (4.111d). Both arguments of the bitwise AND are binary vectors, so the new updated prototype will again be binary. Moreover, taking the bit-pair minimum in each of the p coordinates of the two vectors means that whenever a 1 is removed during this operation, it cannot be restored by a later update of the same prototype.

While the updated prototype vector $\mathbf{v}_{new}$ in Figure 4.81 for algorithms such as LVQ and SHCM can move in any direction in $\Re^p$, the updated ART1 prototype vector $\mathbf{v}_i \leftarrow \mathbf{v}_i \wedge \mathbf{x}_k$ can only gravitate towards the origin, and can only move parallel to the axes of the lattice $\{0,1\}^p$. A side effect of this asymmetric updating strategy is that ART1 is biased towards creating a lot of prototypes if the input data are strings with a lot of 0's. ART1 imposes this constraint on the directions that prototype updates can take in an attempt to control the stability (motion) of the prototypes during learning. This stands in sharp contrast to the method used, for example, in LVQ, where stability is achieved by scheduling the learning rates so that $\{\alpha_{ik,t}\} \rightarrow 0$. See Baraldi and Alpaydn (1998) for a discussion related to conditions on the $\{\alpha_{ik,t}\}$ under which ART1 may converge.

If ART1 is terminated before stabilization, the crisp partition U(**V**) associated with **V** is not *guaranteed* to be $U_{nn}(\mathbf{V})$ unless optional phase C in Table 4.44 is used - that is, a last pass through X after termination of ART1 is needed to construct $U_{nn}(\mathbf{V})$ with equation (2.6a). However, if ART1 is stabilized at termination, a theorem due to Carpenter and Grossberg (1988b) guarantees that $U(\mathbf{V}) = U_{nn}(\mathbf{V})$, without using step C in Table 4.44. Carpenter and Grossberg call this situation "direct access by perfectly learned patterns". To be sure you have $U_{nn}(\mathbf{V})$, just use step C, which always guarantee it. We summarize some other properties of ART1 derived by Carpenter and Grossberg (1988b) that are also paraphrased in Moore (1988) :

- The vigilance parameter essentially controls the diameter of the clusters. Consequently, increasing ρ usually results in more clusters (higher c) with decreased cardinalities. Carpenter and Grossberg call this the *self-scaling property* (the word "self" may be a little misleading, since *you* pick ρ) ;

- Distinct clusters have distinct prototypes ;

- ART1 clusters are stable$_1$;

- For $X \subset \{0,1\}^p, |\mathbf{V}_{ART1}| \le 2^p$, so ART1 is stable$_2$ on finite or infinite input sets. (However, ART1 is not stable$_2$ for $X \subset \Re^p$, but remember that ART1 was not designed for vectors in $\Re^p$) ;

- After stabilization, $\mathbf{v}_i \subseteq \mathbf{x}_k \ \forall \ \mathbf{x}_k \in$ class i. Here $\mathbf{v}_i \subseteq \mathbf{x}_k$ means that $\mathbf{x}_k$ has a 1 wherever $\mathbf{v}_i$ does. Moreover, each $\mathbf{x}_k$ belongs to the jth crisp (nearest prototype) cluster if and only if $\mathbf{v}_j$ is the largest subset of $\mathbf{x}_k$ among the c prototypes.

ART1 is not particularly attractive as either a prototype generator and/or crisp clustering algorithm because of its limited applicability (binary valued data). Moore (1988) asserts that it might be useful for binary character recognition, but that it may not be suitable for signal processing problems where the 0's possess as much information as the 1's. Nonetheless, ART1 is significant for three reasons: first, the issue of plasticity versus stability is both interesting and important, and ART1 was the first model to clearly identify this problem and propose a solution to it; second, relatives and generalizations of ART1 can handle continuously valued data, and these extensions are as good as any other model you might choose to try - but as usual, the proof will always be in the pudding; and lastly, the attempt to connect this model to elements of the biological neural network has a certain charm, even though, in our opinion, the actual connection between any computational NN and the BNN will never be known.

The ART1 architecture is equipped to deal with only binary input vectors; ART2 can handle both analog and binary data. All of the basic building blocks of ART1 are used in ART2. The main difference between the two schemes is in the architecture of the input layer L_1, which is split into a number of sub-layers containing both feedback and feed forward connections. The processing in both the input layer L_1 and output layer L_2 of ART2 is similar to that in ART1. For further details, see Carpenter and Grossberg, (1987b). There is yet another version of ART called ART3 (Carpenter and Grossberg, 1990) for parallel search of learned patterns.

B. Fuzzy relatives of ART

The ART1 model can handle only binary inputs. The usual interpretation of a binary feature value in ART1 is that 1 indicates the presence of some quality, and 0 indicates its absence. In real life many descriptive features are fuzzy, or partially present to some degree. This is, of course, the raison d'être for fuzzy sets in the first place. The most prominent generalization of ART1 to continuously valued data is based on this observation (Carpenter et al., 1991a). To get an ART model for continuously valued data, Carpenter et al. proposed a generalization of the ART model they called *fuzzy ART* (FART) which assumes input data in $[0,1]^p$, and uses the fuzzy set aggregator we call the T_3 norm for the computation of activities and the adaptation of weights. We briefly discuss the changes needed to convert the ART1 algorithm in Table 4.44 into the FART algorithm.

FART begins with the output nodes (which are in what is often called layer 2, denoted here as L_2, with nodes $\{L_{2j}\}$) initialized at the value 1; i.e., $\mathbf{v}_j = (v_{1j},\ldots,v_{pj})^T = (1,\ldots,1)^T = \mathbf{1}$, j = 1,...,c. For this initialization, each category is initially uncommitted, and when an output node wins, it becomes committed. The similarity measure s_1 at (4.110a) used by ART1 to control the search process when input $\mathbf{x}_k$ is presented to the FART algorithm is given by

$$s_{1,\mathrm{FART}}(\mathbf{x}_k, \mathbf{v}_i) = \frac{\|\wedge(\mathbf{x}_k, \mathbf{v}_i)\|_1}{\beta + \|\mathbf{v}_i\|_1}, \ i = 1,\ldots,c \,;\, \beta > 0 \quad , \tag{4.112}$$

where $\beta > 0$ is a constant. The AND operation in (4.112) is defined component wise using T_3, i.e., $\wedge(\mathbf{x}, \mathbf{v}) = (x_1 \wedge v_1, \ldots, x_p \wedge v_p)$. The winner node J is selected, as in ART1 step (4.111a), by maximizing the modified search criterion,

$$s_{1,\mathrm{FART}}(\mathbf{x}_k, \mathbf{v}_J) = \underbrace{\max}_{\mathbf{v}_j \in \mathbf{V}'}\{s_{1,\mathrm{FART}}(\mathbf{x}_k, \mathbf{v}_j)\} \quad . \tag{4.113}$$

If there is more than one winning node, Carpenter et al. (1992) suggest using the winner with the smallest index. The output is subsequently computed by

$$\mathbf{S} = \begin{cases} \mathbf{x}_k, & \text{if } L_{2J} \text{ is inactive} \\ \wedge(\mathbf{x}_k, \mathbf{v}_J), & \text{if } L_{2J} \text{ is chosen} \end{cases} \quad . \tag{4.114}$$

The vigilance test in Table 4.44, equation (4.111b), is made in FART using a generalization of the matching criterion s_2 at (4.110b),

$$s_{2,\mathrm{FART}}(\mathbf{x}_k, \mathbf{v}_i) = \frac{\|\wedge(\mathbf{x}_k, \mathbf{v}_i)\|_1}{\|\mathbf{x}_k\|_1}, \ i = 1,\ldots,c \quad . \tag{4.115}$$

Using (4.114) and (4.115), when the Jth category is chosen (again, it may or may not actually be the winner), resonance occurs if

$$\|\mathbf{S}\|_1 = \|\wedge(\mathbf{x}_k, \mathbf{v}_J)\|_1 \geq \rho\|\mathbf{x}_k\|_1 \quad . \tag{4.116}$$

When this happens, the update equation in (4.111d) is replaced by

$$\mathbf{v}_J \leftarrow \alpha(\wedge(\mathbf{x}_k, \mathbf{v}_J)) + (1-\alpha)\mathbf{v}_J \quad , \tag{4.117}$$

where $\alpha \in [0,1]$ is the learning rate, $\alpha_{ik,t} = \alpha$, for all i, k and t. Other aspects of the algorithmic operation of FART are very similar to our specification of ART1 in Table 4.44.

Carpenter et al. (1991a) assert that when the data are "noisy", it is better to begin with $\alpha = 1$ when J is an uncommitted node, and then switch to $\alpha < 1$ after the node is committed. Thus $\mathbf{v}_J = \mathbf{x}_k$ if $\mathbf{x}_k$ is the first input at which L_{2J} becomes a winner. Carpenter et al. (1991a) call this strategy *fast commitment* and *slow recoding.*

In a large set of data with many distinct values, the possibility of ending up with a large number of prototypes is high. FART tries to control the proliferation of categories by normalizing the input data. The simplest choice is to convert each incoming vector to a unit vector using the standard procedure, $\mathbf{x}_k \leftarrow \mathbf{x}_k / \|\mathbf{x}_k\|$. Another type of normalization discussed by Carpenter et al. can be achieved by *complement coding.* Let the complement of the input vector $\mathbf{x}$ be $\mathbf{x}^c$ where $x_i^c = 1 - a_i$, i=1,..., p. The *complement coded* input $\mathbf{c(x)}$ is defined as $\mathbf{c(x)} = (\mathbf{x}, \mathbf{x}^c) = (a_1, \ldots, a_p, a_1^c, \ldots, a_p^c)^T$. Note that for any $\mathbf{x}$, $\|c(\mathbf{x})\|_1 = p$, and hence complement coding imposes an automatic normalization to the fixed length of p (see the denominator of (4.111b) to understand the motivation for this). A neural realization of FART is discussed in Carpenter et al. (1991b).

So far we have discussed ART models only in the context of prototype generation associated with either nearest prototype crisp clusters, or possibly, as a basis for the design of nearest prototype classifiers (as in Section 4.2). A class of neural architectures for incremental supervised learning that results in a crisp classifier is known as the *adaptive resonance theoretic MAP* (ARTMAP, Carpenter et al., 1991c). An ARTMAP system has two ART1 modules, $\mathrm{ART_a}$ and $\mathrm{ART_b}$, that can create stable categories in response to an arbitrary sequence of input presentations.

ARTMAP is trained in the usual way using (X_{tr}, U_{tr}) for design, and (X_{te}, U_{te}) to test the classifier. The subnet $\mathrm{ART_a}$ receives an input datum $\mathbf{x}_k \in X_{tr}$, while the $\mathrm{ART_b}$ subnet uses the corresponding crisp output label $\mathbf{u}_k \in U_{tr} \in M_{hcn}$. $\mathrm{ART_a}$ and $\mathrm{ART_b}$ are connected by an associative learning network and an internal controller to ensure autonomous operation in (near) real time. The learning rule attempts to simultaneously minimize predictive error while maximizing predictive generalization. The learning scheme increases the vigilance parameter of $\mathrm{ART_a}$ by the minimal amount needed to correct a resubstitution error at $\mathrm{ART_b}$. A prediction failure at $\mathrm{ART_b}$ increases the vigilance parameter of $\mathrm{ART_a}$ by the minimum amount necessary to initiate hypothesis testing by $\mathrm{ART_a}$. This process is known as *match-tracking.*

Carpenter et al. (1992) also generalized ARTMAP for "fuzzy inputs" - that is, input vectors in the hypercube $[0,1]^p$. *Fuzzy ARTMAP* (FARTMAP) replaces the ART1 modules $\mathrm{ART_a}$ and $\mathrm{ART_b}$ of ARTMAP by fuzzy ART or FART modules. These two FART modules are connected by a module called the *map-field*, F^{ab}. The map-field has the same number of nodes as the L_2 layer of ART_b, which are connected to the L_2 layer of ART_a. For ARTMAP, inputs to both ART_a and ART_b are presented in the complement code form. Let $\mathbf{x}$ be an input vector and $\mathbf{u}$ be its corresponding crisp output label vector; then ART_a is given the input $\mathbf{c}(\mathbf{x}) = (\mathbf{x}, \mathbf{x}^c)^T$ and ART_b receives $\mathbf{c}(\mathbf{u}) = (\mathbf{u}, \mathbf{u}^c)^T$ as its input. The map-field F^{ab} is activated whenever one of the ART_a and ART_b categories is active. If both of them are active, then F^{ab} becomes active only when ART_a predicts the same category as that of ART_b via connection weights between F^{ab} and ART_a. The output vector of F^{ab} is $\mathbf{0}$ when the category found by ART_a is disconfirmed by ART_b and in that case ART_a searches for a better category. Readers interested in this scheme are referred to Carpenter et al. (1992) for a detailed discussion of FARTMAP.

Baraldi and Parmiggiani (1995) proposed a crisp variant of ART1 called *simplified ART* (SART). SART is a self-organizing feed-forward network that uses a soft competitive learning scheme to update the prototype vectors associated with the output (L_2 layer) nodes. The *Fuzzy SART* (FSART) model, also proposed by Baraldi and Parmiggiani (1997b), integrates the SART architecture with a soft learning strategy employing a fuzzy membership function. Similar to SART, FSART is also a self-organizing feed-forward network. While processing, FSART adds a new node to the output layer whenever the system fails to categorize a data point, and removes previously allocated nodes whenever they are no longer able to win the competition for any input vector. One advantage of FSART is that it does not require any preprocessing such as normalization or complement coding, and it is quite stable with respect to small changes in the input parameters and the order of data feed. But FSART is computationally expensive compared to ART1 because it needs to determine the "neighborhood-ranking" (Baraldi and Parmiggiani, 1995) whenever it considers a new input.

Blonda et al. (1998) discuss an application of a fuzzy hybrid neural network called the *fully self-organized simplified adaptive resonance theory* (FOSART) model of Baraldi and Parmiggiani (1997a). The FOSART model is a member of the family of neural networks called *radial basis function* (RBF) networks, and we want to include an example from Blonda et al. in this section because it provides us with a very different type of fuzzy NN structure that also ties together several models discussed in previous sections. Towards this end, we take a short excursion into the world of RBF networks.

C. Radial basis function networks

Haykin (1994) provides a nice discussion of RBF networks, so we will not spend a lot of time reviewing them, but Example 4.26 will make more sense to you if we spend just a few pages discussing RBF networks, which are very interesting in their own right. The most important difference between an MLP and an RBF network is that the computing nodes in the first (and only) hidden layer of a typical RBF network use radial basis functions as node functions, instead of the more familiar linear integrators followed by sigmoids as used, for example, in all computing layers of MLPs (Section 4.7). Apparently Broomehead and Lowe (1988) were the first authors to employ RBFs instead of "standard" node functions in a feed forward network architecture.

When $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \Re^p$ is a set of n distinct points, the functions $\mathbf{RBF} = \mathbf{RBF}(\varphi, \|*\|) = \left\{\varphi\left(\|\mathbf{x} - \mathbf{x}_i\|\right)\right\}$ are called a set of *radial basis functions*. In the older literature of classifier design, families such as these were often the kernel functions for classifiers such as Parzen's window (Duda and Hart, 1973). Since any norm can be used for **RBF**, there are infinitely many sets of RBFs for each choice of φ (sometimes called the *generating function* of **RBF**, which at this point is an arbitrary mapping from $\Re^+$ to $\Re$). The points in X are called the *centers* of the basis functions. RBF functions are linearly independent as long as the points in X are distinct.

The function $\varphi\left(\|\mathbf{x} - \mathbf{x}_i\|\right)$ is *radial* because the norm is radially symmetric about $\mathbf{x}_i$; and **RBF** is a "basis" only in some ill-defined sense - viz., that some linear combination of the functions in **RBF** will approximate IO data XY arbitrarily well. In the language of Section 4.6.D, certain families of RBFs are universal approximators (Park and Sandberg, 1991, 1993), so if the "power" of a network is measured by its UA ability, RBFs are equally "as powerful" as MLP networks, and MA and TS fuzzy systems as well.

As mentioned at the beginning of Section 4.6.D, radial basis functions are one of the leading choices for families that are used for "conventional" function approximation (Powell, 1990). Once a norm and generating function φ are chosen, the general form of an approximating RBF family is

$$\mathbf{S}(\mathbf{x}:\boldsymbol{\theta}) = \sum_{k=1}^{n} w_k \varphi\left(\|\mathbf{x} - \mathbf{x}_k\|\right) = \left\langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{x}) \right\rangle \quad , \qquad (4.118)$$

where the unknown parameters or weights which must be estimated with IO training data XY are $\mathbf{\theta} = \mathbf{w} = (w_1, \ldots, w_n)^T \in \Re^n$, and we define $\boldsymbol{\varphi}(\mathbf{x}) = \left(\varphi(\|\mathbf{x} - \mathbf{x}_1\|), \ldots, \varphi(\|\mathbf{x} - \mathbf{x}_n\|)\right)^T$.

Chen et al. (1991) assert that the shape of φ for **RBF** is not as crucial to good approximations of **S** as the choice of centers. A more realistic view is that the quality of the approximation in (4.118) depends jointly on four things: φ, the norm, the centers, and the data used to build the approximation. Chen et al. (1991) identify four families of suitable generating functions: $\varphi(t) = t^2 \log t$; $\varphi(t) = \left(\beta^2 + t^2\right)^{\pm\frac{1}{2}}$, and $\varphi(t) = e^{-(t/\beta)^2}$. Thus, the most familiar, but certainly not the "best" or only choice for the i-th function of an RBF set is a multiple of the univariate Gaussian density $\varphi(t_i) = e^{-\left(t_i/\beta\right)^2}$, where $t_i = \|\mathbf{x} - \mathbf{x}_i\|^2_{\Sigma^{-1}}$ for p-dimension inputs, with each of the n data points used as the mean, $\boldsymbol{\mu}_i = \mathbf{x}_i$ for all i as in (2.18), for example. It is fairly common to assume a circular covariance structure for each function, $\Sigma_i = \sigma_i^2 I$. Under these circumstances we have the *Gaussian radial basis functions*

$$\mathbf{GRBF} = \left\{ \varphi(\|\mathbf{x} - \mathbf{x}_i\|) = e^{-\frac{1}{2\sigma_i^2}\|\mathbf{x}-\mathbf{x}_i\|^2} ; i = 1, \ldots, n \right\} \quad . \tag{4.119}$$

Substituting (4.119) into (4.118) gives

$$\mathbf{S}(\mathbf{x}:\mathbf{\theta}) = \sum_{k=1}^{n} w_k e^{-\frac{1}{2\sigma_i^2}\|\mathbf{x}-\mathbf{x}_i\|^2} \quad , \tag{4.120}$$

which provides approximations to **S** by linear combinations of p-variate Gaussian functions centered at the data with spreads (or widths) $\{\sigma_i^2\}$. If the $\{\sigma_i^2\}$ are unknown, they become part of the parameter vector $\mathbf{\theta}$ that must be estimated.

When n is large (as it will be in almost all interesting real data sets), the approximation in (4.118) gets pretty unwieldy, so we abandon X as the set of centers of the RBFs, and use our old friends $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_q) \in \Re^{qp}$, a set of q prototypes in $\Re^p$ instead of the n data points, as centers of a set of q functions that are sometimes called *generalized radial basis functions* (Haykin, 1994). Using **V** instead of X in (4.118) gives the approximation

$$\mathbf{S}(\mathbf{x}:\boldsymbol{\theta}) = \sum_{i=1}^{q} w_i \varphi(\|\mathbf{x} - \mathbf{v}_i\|) = \langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{x}) \rangle \quad . \qquad (4.121)$$

If φ and the norm in (4.121) have been chosen, the parameters that need to be estimated are now the weight vector $\mathbf{w}$, the q prototypes $\{\mathbf{v}_i\}$, and any other parameters needed by the node function ϕ (such as the width parameter if the RBFs are Gaussian). We can easily cast the approximation problem (finding $\hat{\boldsymbol{\theta}}$ in (4.121)) in a network architecture. Define the integrator and activation functions at node i, i = 1,...,q, in the *hidden layer* as

$$f_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{v}_i\| \quad ; \qquad (4.122a)$$

$$z_i = \phi_i(\mathbf{x}) = F_i(f_i(\mathbf{x})) = \varphi(\|\mathbf{x} - \mathbf{v}_i\|) \quad . \qquad (4.122b)$$

In the notation of Section 4.7, the node functions for the q hidden layer nodes are then $\{\phi_i = \varphi \circ f_i ; i = 1,\ldots,q\}$, the hidden layer node weight vectors (assuming that the parameters of the functions specified by $\boldsymbol{\varphi}$ do not need to be estimated) are the q prototypes **V**, and the output of the hidden layer is the vector $\mathbf{z} = (z_1,\ldots,z_q)^T$. For convenience, let $\mathbf{w}_i' = (w_{1i},\ldots,w_{qi},\alpha_i)^T$ be the weight vector for the output node o_i, and denote the vector obtained by adding a 1 as $\mathbf{z}$'s last (new) coordinate by $\mathbf{z}'$, $\mathbf{z}' = (z_1,\ldots,z_q,1)^T$. Now define the integrator and activation functions at c output nodes (o_i, i = 1,...c), which comprise the output layer, as

$$f_{oi}(\mathbf{z}) = \langle \mathbf{w}_i, \mathbf{z} \rangle + \alpha_i = \langle \mathbf{w}_i', \mathbf{z}' \rangle \quad ; \qquad (4.123a)$$

$$F_{oi}(f_{oi}(\mathbf{z})) = \langle \mathbf{w}_i', \mathbf{z}' \rangle = f_{oi}(\mathbf{z}) \quad . \qquad (4.123b)$$

Equation (4.123a) shows that the i-th output node uses a standard linear integrator function with weight vector $\mathbf{w}_i'$ to be estimated, and uses the identity map for activation. In other words, the output layer comprises a set of c nodes that use node functions that in the early literature were called *continuous perceptrons* (without sigmoidal activation functions, Zurada, 1992). In our reserved terminology for MLPs, the node functions are specifically $\Phi_{LH} = F_L \circ f_H$, so we hesitate to call this output layer a *single layer perceptron* (SLP), but, following Haykin (1994), we cautiously do so here. The output layer has q inputs coming from the hidden layer and c outputs. Combining the equations in (4.122) and (4.123) in a network architecture gives the structure shown in Figure 4.82.

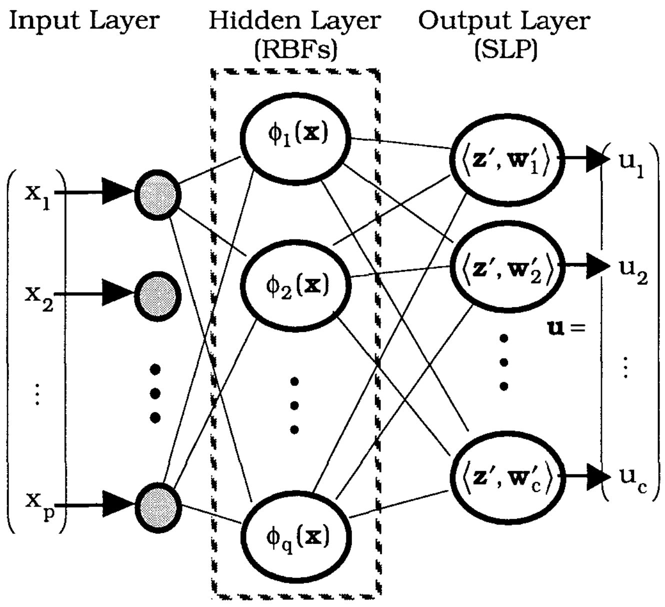


Figure 4.82 A typical radial basis function network

We have shown the hidden layer (or kernel nodes) in Figure 4.82 with q nodes, so the layering architecture of the RBF network in Figure 4.82 is compactly described as $p:q:c$. Like the feed forward network in Figure 4.71, the network in Figure 4.82 is a vector field, $\mathbf{RBF}: \Re^p \mapsto \Re^c$, i.e., the network in Figure 4.82 realizes the approximation $\mathbf{S}(\mathbf{x}:\boldsymbol{\theta}) = \mathbf{RBF}(\mathbf{x}:\mathbf{W})$, where $\boldsymbol{\theta} = \mathbf{W}$ is the network weight vector. If all of the parameters in the hidden layer are assumed known, then $\mathbf{W} = (\mathbf{w}_1', \ldots, \mathbf{w}_q')$; if there are unknown parameters associated with the hidden layer nodes (e.g., the spatial locations and shape parameters of the RBF centers), then **W** includes these parameters as well. Notice that the $p:q:c$ multiple output RBF network can be separated into c single output networks of size $p:q:1$.

The i-th output node of the RBF network produces the real number $u_i = \langle \mathbf{w}_i', \mathbf{z}' \rangle; i = 1, \ldots, c$. When the target output vectors in Y are crisp label vectors, the usual method of converting the network output vector $\mathbf{u} = \mathbf{RBF}(\mathbf{x})$ to a crisp label is to first normalize the outputs of the SLP so that each value lies in the closed interval [0, 1] or the open interval (0,1). One convenient way to do this is to replace the identity in (4.123b) by, for example, a unipolar sigmoid function, so that the output layer becomes a single layer perceptron in the sense of section 4.7 with node functions $\Phi_{LH} = F_L \circ f_H$. This converts the output of **RBF** to a possibilistic label vector which can, if desired (and must, if training error rates are to be computed), be hardened with the function **H** in (1.15). Hardening is not necessary when using the MSE between the computed and target outputs as a measure of generalization quality of the network in question.

Comparing Figures 4.72 and 4.82, we see that the RBF and MLP networks are similar in that both are feed forward structures, but they also have some differences. The main distinctions are that the RBF network usually has only one hidden layer, whereas the MLP very often has two or more; the node functions in the hidden layers are very different; and while the output layer in the MLP usually has nonlinear node functions, the output layer in the RBF network usually has linear node functions as shown in Figure 4.82. There are other differences between the two architectures; we leave discussion of these to Haykin (1994).

There are several training strategies for an RBF network. The simplest way to proceed is to assume the number q of fixed RBFs in the hidden layer, centered at q points selected randomly from the data. The fixed RBFs are often chosen as a special case of the Gaussian functions in (4.119) which have fixed and equal standard deviations that are proportional to the maximum distance between the chosen centers, so that (4.119) becomes

$$\mathbf{GRBF} = \{\varphi_i(\mathbf{x}) = e^{-\left(q\|\mathbf{x}-\mathbf{v}_i\|^2/\delta^2_{\max}\right)}; i = 1,\ldots,q\} \quad , \qquad (4.124)$$

where $\delta_{\max} = \underbrace{\max}_{i \neq j}\left\{\left\|\mathbf{v}_i - \mathbf{v}_j\right\|\right\}$. This fixes the width of each function in (4.124) at $\sigma = \delta_{\max}/\sqrt{2 \cdot q}$. With this approach, the only parameters that need to be learned by training are the c weight vectors of the hyperplanes used for the output node functions, and this can be done (for c = 1 at least) with a technique such as the psuedoinverse (Broomheade and Lowe, 1988).

A second method for training RBF networks, called *hybrid learning*, is a two stage process. This method begins by temporarily regarding the RBF network in Figure 4.82 as two "separate" networks that are trained independently, and then "put together" for testing and operation. If you imagine temporarily breaking all the edges in Figure 4.82 between the hidden and output layers, the network on the left ("*left half-net*") will be a p:q layer network with p input and q RBF "output" nodes; while the network on the right ("*right-half-net*") will be a q:c single layer perceptron whose inputs are the left half outputs.

How will we train the left half-net? Moody and Darken (1989) first suggested that any unsupervised method could be used to get the RBF centers. Methods for doing this fall into the two groups depicted in our Figures 4.1 and 4.2: (i) *selection* to find q centers among the n training data ; or (ii) *extraction*, using, for example, any point prototype clustering algorithm (Chapter 2), or any other point prototype generator (Chapter 4) to find centers for the q RBF node functions. If clustering is used, you will need to settle the issue of how many centers (q) to look for (cluster validity, again). Since we are dealing with labeled IO data, the number of classes (c) is given, but this will be the number of output nodes in the right half-net. We have dealt extensively with this issue in previous sections: suffice it to say that if you choose to train the left half-net with unsupervised learning, the target output set Y is simply ignored during training. Once the centers are obtained, Moody and Darken then used "nearest neighbor heuristics" (which are *not* the k-nn rules discussed in Section 4.4) to find the width of each (Gaussian) RBF.

Once the left half net of size p:q is trained, we know the exact structure of the input and output layers in the right half-net because this network is a q:c single layer perceptron, which may be trained in the usual way (for example, with the *least mean squared* or LMS algorithm, Widrow and Stearns, 1985). This results in estimates for the q weight vectors $\{\mathbf{w}'_j\}$ of the hyperplanes residing in the output nodes of the right half net. During training, outputs of the left half-net on X_{tr} become inputs for training the right half-net against the desired target outputs Y_{tr}.

When this "two-part" hybrid approach to training the network in Figure 4.82 is completed, the left and right half-nets are "pasted together" (or, if you prefer, operated in cascade). Now the p:q:c RBF network can be tested with $X_{te}Y_{te}$ (recall our notation XY for IO data in Section 4.6), and then operated as a network classifier or function approximator in the usual way. The two stage hybrid approach to training an RBF network *might* be superior to the fixed, selected centers training method, but we say might because, as we have emphasized many times, the success of clustering algorithms at discovering good point prototypes for clusters in X depends on whether the data possess clusters that match the clustering model chosen to search for them. And, as always, the ubiquitous cluster validity problem is there to haunt you.

In the third training method, the entire network weight vector **W**, which includes the free parameters in both the hidden layer and output layers, is learned. This is sometimes done by standard back propagation training based on gradient descent conditions (Poggio and Girosi, 1990). Chen et al. (1991) discuss a supervised learning scheme which incrementally *selects* a number of centers in the data for RBFs using a method based on *orthogonal least squares* (OLS). Key features of the OLS method are that it adds centers chosen from X one at a time, always maximizing the increment to the explained variance between the desired and observed outputs, and it is relatively stable in the sense that it can be done without illconditioning problems when using the orthogonal projection theorem. This is much like using principle components analysis for feature extraction, where each additional component used in the linear combination accounts for a successively smaller amount of the remaining total variance in the input data.

The basic model in Blonda et al. (1998) is the RBF network of Figure 4.82, and their training method is the two part, hybrid approach that we couched in terms of "left half" and "right half" nets. These authors compare three classifier designs that differ principally in the method used to train the left half-net. Specifically, they describe two fuzzy schemes and a non-fuzzy approach that uses Kohonen's unsupervised *self-organizing feature map* (SOFM, subsection 4.3.D). The right half-net in all three classifiers is the single layer perceptron with node functions $\Phi_{LH} = F_L \circ f_H$ as in Section 4.7 (i.e., hyperplanes followed by the unipolar sigmoid), trained with the standard LMS method.

The first of the three classifiers discussed by Blonda et al. (1998) uses the *fully self-organized simplified adaptive resonance theory* (FOSART) model (Baraldi and Parmiggiani, 1997a) to build the left half-net. This procedure is basically a heuristic point prototype generating algorithm that combines certain aspects of ART1, competitive learning, and fuzzy c-means clustering to determine the number of nodes q and the positions of the q RBF centers (prototypes **V**). The hidden layer (the output layer in the left half-net) consists of a variable number of RBF nodes, which are analogous to the L_2 layer in ART1. Unlike ART1, however, FOSART nodes can be created or deleted during training. Initialization of the FOSART centers in training the left half-net is also a little different than ART1; FOSART starts with two nodes, $\mathbf{v}_1$ and $\mathbf{v}_2$, taken as the first pair of distinct inputs submitted to the network.

Each FOSART hidden layer node uses a fixed width Gaussian radial basis function as in (4.124), but with a different fixed width than the value determined by $\delta_{max} = \underbrace{\max}_{i \neq j}\left\{\left\|\mathbf{v}_i - \mathbf{v}_j\right\|\right\}$ in that equation. In Blonda et al., the spread of the RBF functions is fixed at $\sigma = 1/\kappa$, $\kappa \in [0,1]$. κ is a user defined parameter that is functionally equivalent to the (normalized) vigilance parameter ρ in ART1 and FART, since it controls the proliferation of nodes in the output layer of the left half-net built by the FOSART algorithm. While the nodes in the standard RBF network (Figure 4.82) are not viewed as "competitive" in the sense of the competitive learning models discussed in Section 4.3, the hidden layer nodes in FOSART are made competitive in Baraldi and Parmiggiani (1997a) in a way that is somewhat similar to the competition in the ART1 L_2 layer.

More specifically, let $\mathbf{x}_k$ be the current input vector at iterate t, and suppose that the FOSART hidden layer currently possesses s nodes (and hence, s prototypes). We are using t here to index passes (epochs) through X, but FOSART also keeps track of the "age" of each node created in the output layer (of the left half-net). Let τ_i denote the age of the i-th node, i=1,...s. (we will explain shortly how this set of parameters is manipulated.) We indicate this extra "time" variable with an additional subscript, so $\mathbf{v}_{i,\tau_i,t}$ is the prototype for node i which has age τ_i at iterate t. FOSART first computes the values

$$\phi_{i,t}(\mathbf{x}_k) = e^{-\left(\kappa^2\left\|\mathbf{x}_k - \mathbf{v}_{i,\tau_i,t-1}\right\|^2 / 2\right)}; i = 1,\ldots,s \quad . \quad (4.125)$$

The largest value in (4.125) is used to identify the winner node, say $\phi_{w,t}(\mathbf{x}_k)$ for the winning node $\mathbf{v}_{w,\tau_w,t-1}$, for this input. The smallest distance in the exponent of (4.125) gives the largest value of the RBF, and conversely, so this terminology makes sense. The value just below the winner value in (4.125) identifies the prototype that is called the *second place* node (second best neuron), since it comes in second in the competition for $\mathbf{x}_k$.

FOSART compares $\phi_{w,t}(\mathbf{x}_k)$ to the "vigilance" parameter, and if $\phi_{w,t}(\mathbf{x}_k) \geq \rho = 1/\kappa$, resonance occurs. When this happens, some of the nodes in the RBF layer will be updated. When $\phi_{w,t}(\mathbf{x}_k) < \rho = 1/\kappa$, FOSART creates a new node $\mathbf{v}_{s+1} = \mathbf{x}_k$ with an RBF centered at $\mathbf{v}_{s+1}$.

When resonance occurs so that learning rates are needed, these are computed by substituting the function $D_{wk,t} = 1 - \phi_{w,t}(\mathbf{x}_k)$ into equation (2.7a), the necessary condition for memberships in fuzzy c-means, with m = 3. So that you don't have to thumb back to Chapter 2, the explicit construction is

$$u_{ik,t} = \left(\frac{1}{1-\phi_{i,t}(\mathbf{x}_k)}\right) \Big/ \sum_{j=1}^{s}\left(\frac{1}{1-\phi_{j,t}(\mathbf{x}_k)}\right), i=1,\ldots,s \quad . \quad (4.126)$$

Since values computed with (4.125) lie in (0,1) the value in (4.126) always exists. The winning node in the output layer of ART1 has lateral connections to all the other nodes in its layer. In FOSART, only some of the nodes in the RBF layer are updated. Blonda et al. call the non-winner nodes that get updated "synaptically linked" to the winning node. (This feature is borrowed from the SOFM, subsection 4.3.D, hence the "SO" part of FOSART.) In other words, FOSART maintains an update neighborhood $\mathcal{N}(\mathbf{v}_{i,\tau_i,t})$ in SOFM style for each node in the RBF layer, but unlike SOFM, it is not the inverse image $\mathcal{N}^{-1}(\mathbf{d}_{i,\tau_i,t})$ of a topologically connected display space. The topological connectivity of the prototypes is maintained in FOSART by a distance rule which can form topology preserving maps (Martinetz et al., 1994).

Update neighborhoods of the RBF nodes expand and contract during training as nodes and node links are created and deleted using the following heuristics. Given a winner and second place nodes for any input, a synaptic link between these two nodes is created ($\mathcal{N}(\mathbf{v}_{i,\tau_i,t})$ grows) if the distance between their prototypes is "fairly similar" to the set of existing distances between pairs of linked centers that already exist for both nodes.

More specifically, the links from any RBF node to other nodes in its current update neighborhood satisfy a *link constraint*. Let $\{\mathbf{v}_{j,\tau_j,t}\} = \mathcal{N}(\mathbf{v}_{i,\tau_i,t})$; FOSART requires the ratio of the maximum to the minimum pairwise distances of the prototypes in neighborhood $\mathcal{N}(\mathbf{v}_{i,\tau_i,t})$ to satisfy $\underbrace{\max}_{s \neq q}\left\{\left\|\mathbf{v}_{q,\tau_q,t} - \mathbf{v}_{s,\tau_s,t}\right\|\right\} \Big/ \underbrace{\min}_{s \neq q}\left\{\left\|\mathbf{v}_{q,\tau_q,t} - \mathbf{v}_{s,\tau_s,t}\right\|\right\} \le \chi$. The threshold χ is chosen by the user, and FOSART currently employs the value $\chi = 1.6$. Now suppose $\mathbf{v}_{w1,\tau_{w1},t}$ and $\mathbf{v}_{w2,\tau_{w2},t}$ to be the current first and second place winners. If nodes w and w2 are not already linked, a link is established between them if the ratio of their internode distance, $\left\|\mathbf{v}_{w1,\tau_{w1},t} - \mathbf{v}_{w2,\tau_{w2},t}\right\|$, to the minimum of the two sets of distances for pairs of nodes in $\mathcal{N}(\mathbf{v}_{w1,\tau_{w1},t})$ and $\mathcal{N}(\mathbf{v}_{w2,\tau_{w2},t})$ is less than or equal to $\chi = 1.6$. If the link from w1 to w2 is inserted, new sets of distances are computed over both neighborhoods, and any links in either one that no longer satisfy the link constraint are deleted. Moreover, links that have not been used for an entire pass through X_{tr} are now deleted (i.e., $\mathcal{N}(\mathbf{v}_{i,\tau_i,t})$ shrinks).

At resonance, learning rates are computed for all s nodes in the output layer as follows:

$$\alpha_{ij,t} = \begin{cases} \left(u_{w1,t}\right) \cdot \left(u_{w1,t}\right)^{\frac{\tau_{w1}}{\tau}} & ; i = w1 \\ \left(u_{jk,t}\right) \cdot \left(u_{jk,t}\right)^{\frac{\tau_{w1}+\tau_j}{\tau}} & ; \mathbf{v}_{j,\tau_j,t} \in \mathcal{N}(\mathbf{v}_{w1,\tau_{w1},t}) \\ 0 & ; \text{otherwise} \end{cases} , \tag{4.127}$$

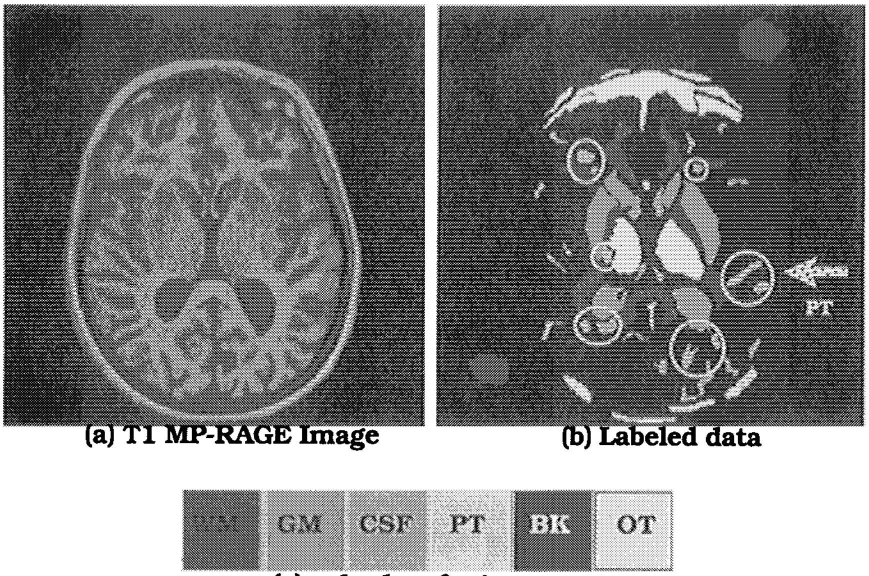
As we have said, τ is a user specified constant that controls the time available for learning. In Blonda et al. (1998) $\tau_i \in [0,\infty)$ is a real number which begins at 0 and simply accumulates the sum of the learning rates applied to the i-th RBF node. Thus, after computing (4.127), the ages of the s nodes are reset using $\tau_i \leftarrow \tau_i + \alpha_{i,\tau_i,t}$. Baraldi and Parmiggiani (1997a) call the winner node w1 stable (and updates of this node stop) when $\tau_{w1} \geq 3 \cdot \tau$, for at this point the exponent of the second factor for the winning node in (4.127) is ≥ 3. After computing (4.127), the nodes in the RBF layer are updated with equation (4.11),

$$\mathbf{v}_{i,\tau_i,t} = \mathbf{v}_{i,\tau_i,t-1} + \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,\tau_i,t-1}),\ \ \tau_i < 3 \cdot \tau; i = 1,\ldots,s. \tag{4.128}$$

Notice that a condition for updating is that the node has not reached its stabilization age. If a current node in the RBF layer is never a winner (in the ART sense) for an entire epoch *but others are*, and it has not already "stabilized" (so that updates on it have stopped), the node is then deleted from the network. Updating stops when either (i) *all* of the nodes have stabilized (none are updated for an entire epoch); or (ii) when $\|\mathbf{V}_t - \mathbf{V}_{t-1}\| < \varepsilon$ in some convenient matrix norm. At this point FOSART has created a set of q RBF nodes and provided estimates for the centers of the RBF node functions in the output layer of the left half-net.

After the output layer of the left half-net is determined by FOSART, the right half-net is trained. Blonda et al. (1998) do not specify how the weights of the output layer are initialized, what the parameters of the unipolar sigmoids are, nor how training is terminated. The two independently trained half-nets are then coupled, and the resultant p:q:c RBF network, structured as in Figure 4.82 (except for the sigmoids in the c output nodes) is called the FOSART-SLP classifier. With this as background, we present an example abstracted from several papers about FOSART.

Example 4.26 Blonda et al. (1998) consider classifier design in the context of lesion detection in MR images taken from a patient diagnosed with multiple sclerosis (MS). An interesting ancillary aspect of this work is the comparison of results using standard MR images with a new type of MR imagery called *magnetization-prepared rapid gradient echo* (MP-RAGE), which can produce thinner slices than standard MR devices. Figure 4.83(a) shows one slice from the T1 MP-RAGE sequence for the patient with MS. Unlike brain tumors, which produce very visible lesions (see Figures 4.16 and 4.19), multiple sclerosis produces small heterogeneous lesions that are in some sense similar to microcalcifications in digital mammograms - small, well distributed, and hard to see. Can you find them in Figure 4.83(a)?



(a) T1 MP-RAGE Image **(b) Labeled data**

(c) color key for image (b)

Figure 4.83 (a) Raw MP-RAGE image data, and (b) training data selected from (a) by a neuroradiologist, with (c) color key for view (b).

Figure 4.83(b) shows the training data extracted from the MR slice corresponding to Figure 4.83(a) by an expert neuroradiologist. The labeled data consist of a total of 8627 pixels. The number of pixels in each of six tissue classes are reported in Table 1 of Blonda et al. (1998), repeated here as our Table 4.45. Half of the pixels in each of the $c = 6$ classes were randomly selected for training, and the remaining 50% were reserved for testing the three classifiers discussed by the authors.

Standard 3D spin echo MR images such as those shown in Figures 4.16(a) and Figures 4.19 (a)-(c) result in pixel vectors which have the form discussed in Example 4.5, viz., $\mathbf{x}_{ij} = (T1_{ij}, T2_{ij}, \rho_{ij})$. The MP-RAGE data, which is derived from an extension of the turboflash technique (Brandt-Zawadzki et al., 1992), is a function of the T1 gradient spin echo sequence. MP-RAGE also produces three dimensional data which has been successfully used in brain image analysis (Blonda et al., 1996a). The data used for the images in this example are 3D pixel vectors made by replacing the T1-spin echo intensity with the T1 MP-RAGE intensity in the 3D spin echo data. This gives us pixel vectors $\mathbf{x}_{ij} = (T1\text{-}RAGE_{ij}, T2_{ij}, \rho_{ij})$, leading to the data set $X = \{\mathbf{x}_{11}, \mathbf{x}_{12}, \ldots, \mathbf{x}_{ij}, \ldots, \mathbf{x}_{mn}\}$ in $\Re^3$. The images used by Blonda et al. (1998) had spatial dimensions $m = n = 256$.

Table 4.45 Pixels in tissue classes selected by a neuroradiologist (after Blonda et al., 1998, Tables 2-4)

Tissue	Abbr.	# of pixels
white matter	(WM)	1675
gray matter	GM)	1294
cerebro-spinal fluid	(CSF)	1251
pathology	(PT)	479
background	(BK)	848
other	(OT)	3080

The color key in view (c) of Figure 4.83 would enable you to see a total of 10 isolated regions in view (b) that are labeled *pathology* (PT) - if you could see view (b) in color. Reproduced in shades of gray, however, it is pretty hard to see the regions labeled PT, so we have superposed an arrow in the center right of view (b) pointing to 2 of the 10 pathology regions (which are enclosed by one circle), and circled the rest of them without arrows in little ellipses so you can find them. There are 6 circled regions: 4 of them contain 2 pathology regions each, and the other two contain just one.

The first classifier discussed in Blonda et al. (1998) is the RBF network discussed prior to this example, with the left half-net trained by FOSART, and the right half-net trained by the standard LMS rule. Protocols for the runs made will be given shortly.

The second classifier discussed in Blonda et al. (1998) is based on the same two layer p:q:c RBF structure that FOSART finds. In the second design the number of hidden layer nodes is fixed at the FOSART determined value of q, and the centers of the fixed width Gaussians specified in (4.125) are found by applying a modified version of the batch FLVQ algorithm (subsection 4.3.H) to the training data. Baraldi et al. (1998) recommend 3 heuristic modifications of descending FLVQ based on conclusions they drew from 10 numerical experiments. The recommended modifications to the algorithm of Table 4.12 are that: $m_f = 1.05$ (instead of 1.1); that termination criterion $\varepsilon = 0$ (that is, the recommendation is to abandon the computation of E_t in Table 4.12, and always run descending FLVQ to the final value $m_f = 1.05$); and finally, that a value for $\Delta m = (m_t - m_0)$ always be chosen in the range [0.01, 0.05], regardless of the values selected for m_0 and T. The classifier discussed in Blonda et al. (1998) that is illustrated in this example used all of these modifications to descending FLVQ while determining the prototypes for the RBFs in the hidden layer of the FLVQ based classifier. The structure of and weights for the output layer were determined in exactly the same fashion as for the FOSART-SLP design. This second classifier will be called the FLVQ-SLP network.

Finally, a third two stage p:q:c RBF design that was structurally identical to the FOSART-SLP and FLVQ-SLP networks was built by Blonda et al. using Kohonen's *self organizing feature map* (SOFM) approach to find the centers of the RBFs in the output layer of the left half-net. In the experiments below the initial learning rates were $\alpha_{ik,0} = 0.5 \forall i$; and these rates, applied uniformly across the nodes being updated, decreased monotonically with the formula $\alpha_{ik,t} = \alpha_{ik,0}(1-(t/T))$. The value of T was the total number of training data times the number of epochs run. For example, the first run used 47 epochs on 8,627/2 training data, so T=202,734.

The update neighborhoods for the four unsupervised SOFM runs shown in Table 4.46 were linear arrays in this example, and the initial sizes (radii) of the update neighborhoods for the four runs discussed in Table 4.46 were 10, 6, 6 and 5, respectively. For example, if the neighborhood size is 5 and $\mathbf{v}_{14,t}$ is the current winner, the prototypes that get updated are the 11 consecutively indexed centers $\{\mathbf{v}_{9,t},\ldots,\mathbf{v}_{19,t}\}$. The radius of the neighborhood in SOFM display space was also decreased monotonically with the equation $\lceil r_t = r_0(1-(t/T)) \rceil$. The SLP layer was built and trained as the other two classifiers were. Initialization and termination conditions for the SOFM prototypes and SLP weight vectors are not specified in Blonda et al. (1998).

Now we are finally ready to discuss the results shown in Blonda et al. (1998). Four training runs with X_{tr} were made with FOSART-SLP using four different values for the "vigilance" parameter, $1/\rho = \kappa =$ 0.044, 0.010, 0.120 and 0.147. The total number of RBF hidden layer nodes at termination in each of these four runs was then fixed as the number of RBF nodes in the other two networks (that is, all three classifiers had the same architecture in each run, initially determined by the FOSART-SLP runs). The number of hidden RBF nodes for the 4 FOSART runs was q = 22, 109, 160 and 254. In other words, all three classifiers had 3:22:6 configurations for run 1, etc.

The number of passes through the training data to termination for the FOSART runs was also forced on the other two classifiers. The FOSART runs terminated in 47, 14, 12 and 15 passes, respectively, so the other two classifiers were designed using prototypes (that were possibly still being updated) at the same number of passes. Table 4.46 combines the information reported in Tables 2, 3 and 4 in Blonda et al. Protocols for the FOSART-SLP runs were $\tau = 100$, $\varepsilon = 1$; and for FLVQ-SLP, $m_0 = 2$ and $m_f = 1.05$. Table 4.46 shows the MSE achieved on the training and test sets, as well as the percent correct classification for both training and testing in each of the four runs. In terms of the MSE criterion FLVQ does consistently better (is lower) than SOFM, and is lower than FOSART in 5 of the 8 cases shown. In terms of error rates (here shown as percent correct), FOSART is a few percent lower than both of the other designs in all 4 resubstitution cases, and is lower than both of the other classifiers in all of the test error cases except for SOFM, run 2, where it is a little higher than SOFM.

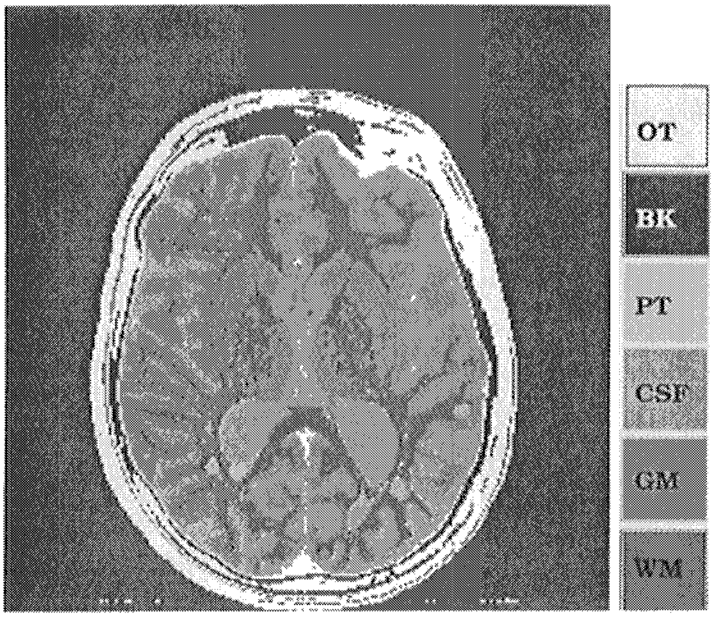
Table 4.46 Training and test results for the image in Figure 4.83(a) (after Blonda et al., 1998, Table 1)

Run	resubstitution MSE (X_{tr})			test MSE (X_{te})		
	FOSART	FLVQ	SOFM	FOSART	FLVQ	SOFM
1	183.4	153.8	183.6	190.1	155.1	178.8
2	57.1	59.2	73.8	67.9	63.9	75.0
3	41.8	49.1	64.3	52.9	52.4	65.8
4	28.3	31.2	43.9	39.8	36.4	47.8

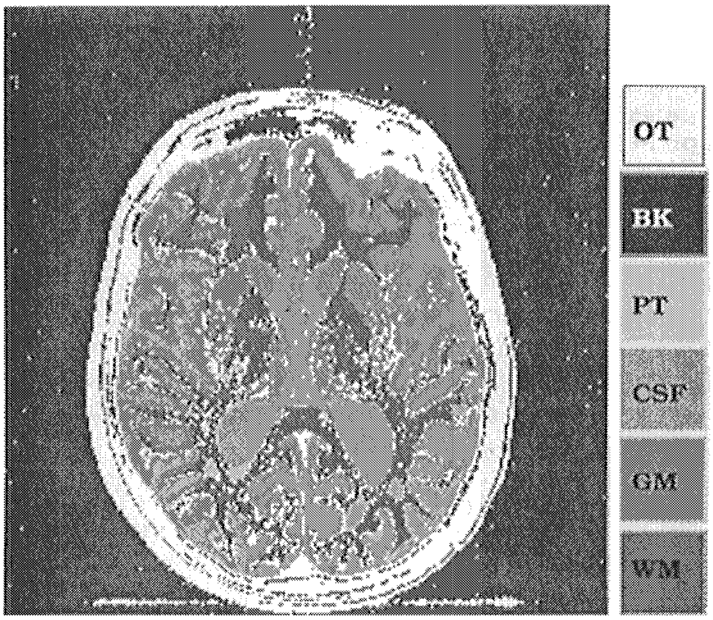
	resubstitution : % correct			test error : % correct		
	FOSART	FLVQ	SOFM	FOSART	FLVQ	SOFM
1	68.4	73.0	72.5	67.0	71.8	70.4
2	75.2	76.9	77.1	74.7	76.2	72.5
3	75.7	78.2	78.3	74.9	77.2	77.6
4	71.1	79.3	78.7	76.3	78.3	78.5

What can be concluded from these experiments? None of the values in Table 4.46 suggest a real advantage to any of the three designs. Rather, and very similarly to Table 4.14, where four algorithms, including LVQ and descending FLVQ also produced very similar results on Iris, we view the three classifiers in this example as being very similar. With a little tuning here and there, it is quite likely that any of the three designs could realize the "best" outputs.

Figure 4.84 shows the final segmentations of the original image made by the three classifiers in run 1, with the same tissue color key as used in Figure 4.83(b) appended to each. These segmentations were made by running the classifiers on the entire 65,536 3D pixel vector image data. Each classifier has c = 6 perceptron nodes with node functions $\phi_{LH} = F_L \circ f_H$ at its outputs, so the overall structure of the trained networks is as possibilistic classifiers that produce a label vector $\mathbf{u} \in N_{p6}$ for each pixel. The labels are then hardened with (1.15) and each pixel was colored using the color assigned to the corresponding tissue class during the initial labeling of the training data (i.e., the colors shown in Figure 4.83(b)).

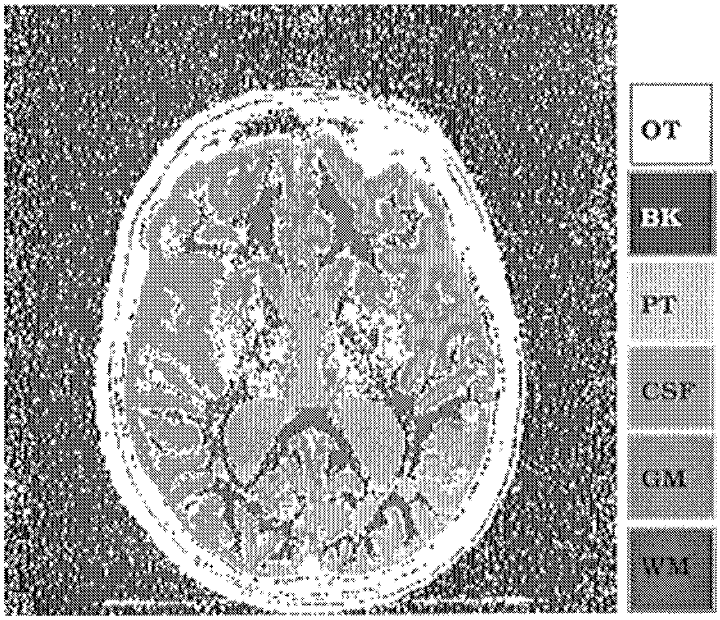


(a) FOSART-SLP



(b) FLVQ-SLP

Figure 4.84 Segmentations of Figure 4.83(a), Blonda et al. (1998)



(c) SOFM-SLP

Figure 4.84 (con't.) Segmentations of Figure 4.83(a)

Visual comparison of these three images to the ground truth - the labeled image in Figure 4.83(b) - is pretty difficult in shades of gray, and is further complicated by the fact that all 65,536 pixels are colored in Figure 4.84, but only a small fraction of them (8,627) are visible in Figure 4.83(b). If you could see the color images, the FOSART segmentation would look a little better than the other two on some of the lesions in the lower half of the image. It's a little hard to understand why this is true, since FLVQ and SOFM both enjoyed lower MSEs and higher percent correct classification than FOSART for run 1. Perhaps a more informative display would be the difference images on just the ground truth pixels in X_{te}, which would show the effectiveness of the three classifiers at labeling MS lesion pixels in the test set.

Blonda et al. (1998) assert that the performance of FOSART recorded in Table 4.46 shows that it is stable to small changes in the parameter κ. They also state that the 3D data used, with the T1 MP-RAGE intensities, produced somewhat better results than the standard 3D spin echo MR data. No evaluation of the medical significance of the images in Figure 4.84 is reported by Blonda et al.

The ART1/FART architecture seems to have minimal influence on the design of FOSART, although several of the basic concepts (node creation, vigilance) of ART1 are certainly evident in FOSART. Our overall assessment of the classifiers in Example 4.26? In the first place, we think that the method of comparison was at best, a little unfair, and at worst, crippling to the FLVQ and SOFM methods. The architecture discovered by FOSART was forced on the other two networks (and hence, not necessarily optimal for their performance criteria).

Comparing the three classifiers built by prototypes obtained from the same number of training epochs - again picked by FOSART - also seems unfair to the FLVQ-SLP and SOFM-SLP designs. After all, the rate of convergence of different algorithms that are looking for a solution to a common problem is often different, but, mindful of the tortoise and the hare, it is certainly possible that slower algorithms can produce better solutions in every sense except the time parameter used to stop them. FOSART was allowed to terminate, while the other two designs were simply sampled at the same time before they terminated in their own right. This seems to prejudice the examples given in favor of FOSART. We appreciate the authors' honest attempt to compare apples to apples, but in this case, some apples seem more equal than others.

Given these disclaimers, it surprises us that FLVQ and SOFM performed better than FOSART under these circumstances. It might be the case that on a level playing field, the FLVQ and SOFM based designs would enjoy an even clearer advantage than is evident in this example. Finally, you have to wonder how a standard RBF network (sans fuzziness) or the crisp MLP with one or two layers would compare to the results in Example 4.26, or for that matter, how well segmentations with a non-neural model, several of which have already been discussed (and see Chapter 5 for more), would compare with the outputs shown in Figure 4.84.

Nonetheless, we like the basic ideas in this example, because the FF networks discussed in Blonda et al. (1998) have a very different flavor than the ones discussed in Section 4.7. Dynamic reconfiguration of the RBF layer during left half-net training seems like a good and clever idea; the authors assert that this has the effect of countering the tendency of ART-like models to overfit the data, and this strategy eliminates dead nodes. The manipulation of the update neighborhood in FOSART (the RBF layer in the left half-net) is also very different from ART1/FART and indeed, SOFM as well. This aspect of the FOSART scheme may provide it with some nice (as yet unproved) local properties akin to topological connectedness of the update neighborhoods. On the other hand, FOSART is a little like the color resulting from mixing 4 or 5 different paints - it might be splendid, or it might black everything out. So, when you try a hybrid scheme like this, add a little bit at a time, stir well, and test often.

Radial basis function networks (crisp or otherwise) provide a fundamentally different approach to classifier design than the MLP model discussed in Section 4.7. However, there is a connection between RBF networks and FAN models. Choose $\phi_i(\mathbf{x}) = e^{-\left(\|\mathbf{x}-\mathbf{v}_i\|_A^2\right)}$ in (4.122b), where A is a positive-definite diagonal matrix. Then $\phi_i(\mathbf{x})$ can be written as a product of p 1D Gaussian functions, which can be interpreted as the membership functions for the linguistic values appearing in the bottom layer of a FAN (refer to Figure 4.80). Consequently, the outputs of the hidden layer units in Figure 4.82 can be interpreted as the conjunctive combination of membership values. The output layer nodes in an RBF network have linear activation functions which can be realized as generalized means. Thus, RBF networks are roughly equivalent to FANs that have conjunctive nodes in the hidden layer and generalized mean nodes at the output layer. Estimating the parameters of $\phi_i(\mathbf{x})$ for an RBF network is roughly equivalent to estimating the parameters of the membership functions in a FAN.

RBF networks are usually easier to train than back-propagation designs, are related to several well known conventional methods - e.g., Parzen windows, and provide approximations that are much more local than FFBP designs (Lippman, 1989). We think that RBF networks are important enough to deserve a whole chapter - but in another book ☹. We will discuss a few other fuzzy NNs in Section 4.11.

4.9 Fusion techniques

Real applications, such as assisted medical diagnosis, handwritten word recognition, automatic target recognition, buried land mine detection, etc., are, unfortunately, not like the Iris data. By this we mean that it is rarely the case that successful systems can be designed using only a few features and almost any classifier, as is the case with the Iris data.

More often, many sources of information are needed to provide partial (soft) classifications, followed by an aggregation function of some type. This strategy has become widely accepted, and is known by many names, including data fusion, information fusion, multistage classifier design or classifier fusion, sensor fusion, and so on. However, the key ingredients in most of these approaches are shared by them all: multiple sources of information provide partial classifications; the classifications are then somehow joined together to give a final (hopefully better) decision than any component classifier could. If you think of features as the sources of information, then many of the classifiers we have discussed in this chapter can be regarded as fusion devices.

We will only scratch the surface of this extremely important topic in the hope that you will use our discussion as an entry point into the literature. Hall (1992) discusses a variety of mathematical techniques that can be useful in the context of sensor fusion. Dasarathy (1994b) focuses on decision fusion and contains over 700 references on this topic, going back to 1981. We will use the terms information fusion and sensor fusion interchangeably, even though there are clearly distinctions between them.

There are several ways to develop a taxonomy of the levels at which information fusion activities can take place. One such hierarchy includes data level, feature level, and decision level fusion (Sims and Dasarathy, 1992, Dasarathy, 1994b). If the data are temporal in nature, such as a sequence of images over time, we should add temporal information fusion to this list.

A. Data level fusion

Data level fusion involves combining sensor outputs directly. A primary example of data level fusion is the combination of precisely registered images from multiple sensors or wavelengths, such as color images, multispectral images, or images acquired using multiple infrared bands. This type of fusion is quite useful *only if* precisely registered information is available. Figure 4.85 shows an example of data level sensor fusion for two registered images from different sensors that contain information about buried land mines. Using the DARPA Backgrounds data, which is based on *ground penetrating radar* (GPR), and a *forward looking infrared* (FLIR) image acquired by Geo-Centers, Inc., suitable image processing techniques can provide complementary evidence of the presence or absence of the mine-like objects. There are three objects of interest in the illuminated scene, two near the top of the frame, and one at the bottom. The GPR image (top left panel in Figure 4.85) had strong returns for the two objects at the top of the image. Since the bottom object is not in the data, subsequent processing of this image alone (shown immediately to the right of the GPR image, with the targets indicated by small white arrows and "T"s), misses the third object, and produces a detected false alarm. However, by combining the registered FLIR and GPR image data before processing, and using morphological operations on the combined data, the three objects were detected and the false alarm eliminated, as shown (very faintly) in the bottom right view of Figure 4.85.

Figure 4.85 is a very simple example that demonstrates the *concept* of complementary sensor fusion at the data (or image) level. It's possible that in processing the one FLIR frame in the bottow left view, all three objects could be found without false alarms, but it is just as likely that 0, 1, or 2 might have been detected. This is one frame (out of thousands) - and is one of the few we could find that showed objects in both the GPR and FLIR images. Hence, Figure 4.85

is really only a conceptual diagram that attempts to answer the question- "what is data level fusion?".

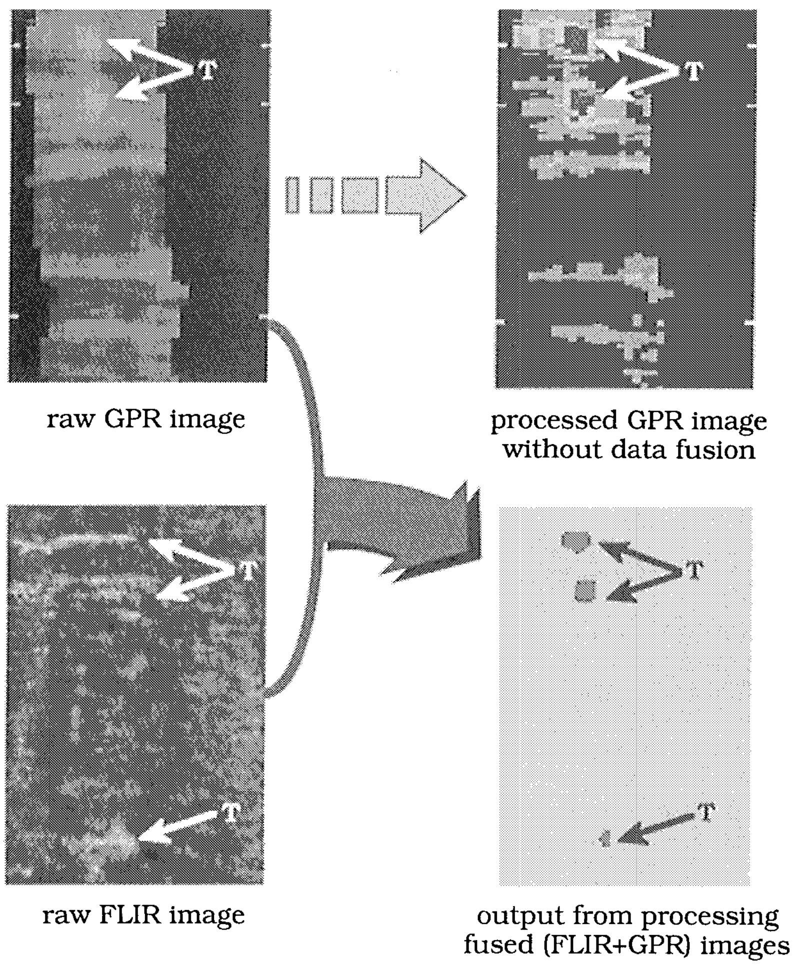

T
T
T
T
T
T
raw GPR image
processed GPR image without data fusion
raw FLIR image
output from processing fused (FLIR+GPR) images

Figure 4.85 Data fusion aids object detection

A main difficulty in fusing image data this way is that the images must be accurately registered in order to perform pixel by pixel data fusion. Due to differences in range and resolution of various sensors, direct data level fusion such as this is usually effective only in carefully controlled situations.

While the potential payoffs of sensor fusion are high, there are many difficulties. Image data are often unregistered and non-collocated. Passive imaging sensors can be registered but they often have different resolutions, causing different intensity distributions which can make registration and matching a difficult problem. The variation in object signatures from different sensing modalities also make it difficult for an algorithm to reliably match potential objects of interest in different image types.

Another problem in sensor fusion is that information may be missing in one sensing modality but available in another. This statement can be true in a partial sense. An object may be partially occluded in one sensing modality but not occluded in another. There may be high contrast between two regions in one modality but not in another, etc. These attributes (occlusion, contrast) are not binary - they are true to some degree. The fusion algorithm must use whatever partial information is available.

Measurements on sensor outputs always contain uncertainty. This uncertainty is caused by inherent physical limitations (resolution, etc.), from the partial information problem, and from imperfections in the algorithms themselves. A practical and effective fusion algorithm must make full use of the available information without being overwhelmed by imprecise and conflicting measurements. The use of fuzzy set theoretic models within the information fusion domain explicitly recognizes this uncertainty and provides mechanisms which often successfully manage the uncertainty and thereby arrive at more realistic answers than crisp, precise models.

Another type of fusion that can be regarded as either data level or sensor level fusion is discussed in a pair of papers by Hathaway et al. (1996) and Pedrycz et al. (1998), who present three models for fusing *heterogeneous fuzzy data* (HFD). The objective of this type of fusion is to convert fuzzy numbers into numerical data vectors (feature vectors in $\Re^p$). Using our standard notation, we let n column vectors in a data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \Re^p$ be arrayed as a $p \times n$ *object data matrix*, which we denote as $\mathbf{X} \in \Re^{pn}$ by letting column k of $\mathbf{X}$ be the column vector $\mathbf{x}_k$, $\mathbf{X} = [\mathbf{x}_1 \ \ \mathbf{x}_2 \ \ \cdots \ \ \mathbf{x}_n]$. Here dimension p is the number of *generalized coordinates* in the chosen representation of the heterogeneous fuzzy data; p will vary as the parametrization of X does.
To understand what type of data this is, Consider the speed s of a vehicle. The features used for describing and classifying vehicle speed (of, e.g., trucks on a highway) can have various representations. If measured precisely at some time instant, speed s is a real number, say s = 100. Figure 4.86(a) shows the membership function, $m_1(s) = 1 \Leftrightarrow s = 100$; otherwise, $m_1(s) = 0$ for this case. This piece of data could be collected by one observation of a radar gun.

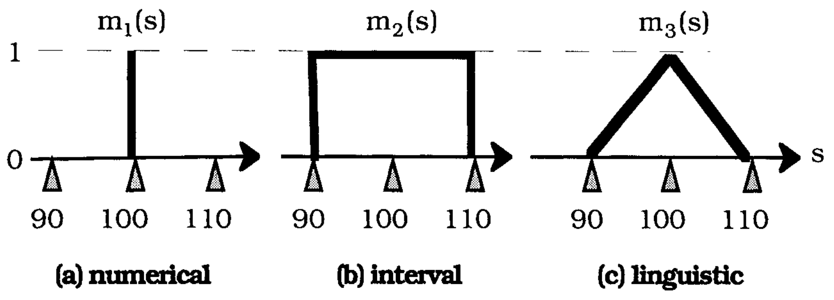


Figure 4.86 Membership functions of crisp and fuzzy data

Next, suppose that two radar guns held by observers at different locations both measure s at the same instant. One sensor might suggest that s = 90, while the second measurement might be s = 110. Uncalibrated instruments could lead to this situation. In this case, several representations of the collected data offer themselves. One way to represent these temporally collocated data points is by the single interval [90, 110], as shown by the membership function $m_2(s)$ in Figure 4.86(b), $m_2(s)=1 \Leftrightarrow 90 \le s \le 110$; otherwise, $m_2(s)=0$. Another plausible representation is to center a small interval of radius ε about each observation, leading to the *pair* of real intervals $[90-\varepsilon, 90+\varepsilon]$ and $[110-\varepsilon, 110+\varepsilon]$. Both representations make the same point - that data can come to us in the form of real intervals.

Finally, it may happen that vehicle speed is evaluated non-numerically by a human observer, who might state simply that "s is very high". In this case the observation can be naturally modeled by a real fuzzy set. The membership function $m_3(s)$ shown in Figure 1(c) is one (of infinitely many) possible representation of the linguistic term "very high", $m_3(s)=\max\{0,1-0.1|100-s|\}$,$s \in \Re$. Taken together, the three forms of data shown in Figure 4.86 are called *heterogeneous fuzzy data* (HFD), and our objective is to find a transformation of these three types of fuzzy numbers so that each of the input data wind up in the numerical feature space $\Re^p$.

Let $\Re$ be the real numbers, $I(\Re)=I$ be all closed real intervals such as [a, b], and $\mathcal{F}(\Re)=\mathcal{F}$ be the real fuzzy subsets of $\Re$. Every element of $\mathcal{F}$ is a membership function $m:\Re \mapsto [0,1]$. Next, let

$$\mathcal{F}^p = \underbrace{\mathcal{F} \times \mathcal{F} \times \cdots \times \mathcal{F}}_{\text{p times}} \quad . \tag{4.129}$$

An element of $\mathcal{F}$ is a function that represents a real number, real interval or fuzzy set of real numbers; an element of $\mathcal{F}^p$ is a p-tuple of them. For example, the vector $\mathbf{x} = (1.32, |\sin(x)|, [-3, 4.5], 2.77, x^2$ for x in [-1,1]) is in $\mathcal{F}^5$. The most general form of HFD is a collection of n vectors $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \subset \mathcal{F}^p$. Hathaway et al. (1996) discuss several parametric representations of this data as a set of generalized coordinates in some real, finite-dimensional vector space.

Because real numbers and intervals can be represented by crisp membership functions, each vector in X can be regarded as a p-tuple of real membership functions. This is the case that is discussed in Pedrycz et al. (1998). We will briefly discuss the simpler case set out in Hathaway et al. (1996), of parametric HFD models by considering here only the more restricted case obtained by constraining the membership functions for each coordinate of $\mathbf{x}_k$ to be *symmetric trapezoidal fuzzy numbers.*

Our notation for any representation of a symmetric trapezoidal fuzzy number is $m_{\mathbf{a}}(x; a_1, a_2, a_3) = m_{\mathbf{a}}(x; \mathbf{a})$, where $\mathbf{a} = (a_1, a_2, a_3)$ is the vector of parameters that specifies m in the chosen representation. We will regard $m_{\mathbf{a}}$ as the *standard* representation of a symmetric trapezoidal fuzzy number, and will refer to a_1 as the *center,* a_2 as the *inner radius,* and a_3 as the *outer radius* of the graph specified by $m_{\mathbf{a}}(x; \mathbf{a})$. Using this standard representation, we let

$$\mathcal{FST}(\mathbf{a}) = \{m_{\mathbf{a}} : \Re \mapsto [0,1] : \mathbf{a} \in \Re^3; a_2, a_3 \geq 0\} \quad . \tag{4.130}$$

$\mathcal{FST}^p(\mathbf{a}) = \underbrace{\mathcal{FST}(\mathbf{a}) \times \mathcal{FST}(\mathbf{a}) \times \cdots \times \mathcal{FST}(\mathbf{a})}_{\text{p times}} \subset \mathcal{F}^p$ is obtained by replacing $\mathcal{F}$ in (4.129) with $\mathcal{FST}(\mathbf{a})$. There are four kinds of symmetric trapezoidal fuzzy numbers in $\mathcal{FST}(\mathbf{a})$, viz., real numbers, intervals, and symmetric triangular and trapezoidal fuzzy numbers.

Every coordinate of a vector $\mathbf{x}$ in $\mathcal{FST}^p(\mathbf{a})$ has a unique representation as an element of $\mathcal{FST}(\mathbf{a})$ for an appropriate choice of $\mathbf{a}$. When every element in a data set X is in $\mathcal{FST}^p(\mathbf{a})$, we call it *parametric* HFD, because each *generalized coordinate* of every $\mathbf{x}_k$ in X has a unique representation as an element of $\mathcal{FST}(\mathbf{a})$ for some choice of $\mathbf{a}$. We abstract a simple example from Hathaway et al. (1996) that shows how their parametric HFD model can be used for data fusion, which in turn enables us to do clustering and classifier design with this type of mixed data.

Example 4.27 Table 4.47 lists, without parentheses or comma delimiters, a set of n = 9 data points in $\mathcal{FST}^2(\mathbf{a})$. The superscript of $\mathcal{FST}^2(\mathbf{a})$ indicates that each data point comprises a *pair* of generalized coordinates; $\mathbf{a}_i$ for i = 1, 2. For example, interpret row one in this table as the generalized coordinates of the vector $\mathbf{x}_1 = (1.1, 0.0, 0.0, 1.5, 0.0, 0.0)^T \in \Re^6$. The first triple of coordinates specifies the symmetric trapezoidal fuzzy number m_a(x; 1.1, 0.0, 0.0), the real number 1.1, and the second triple specifies the symmetric trapezoidal fuzzy number m_a(x; 1.5, 0.0, 0.0), which is the real number 1.5. The pair of generalized coordinates for $\mathbf{x}_2$ in $\mathcal{FST}^2(\mathbf{a})$ specifies a real interval and a real number, and so on.

Table 4.47 A 9 -point parametric HFD set in $\mathcal{FST}^2(\mathbf{a})$

	First Variable : $\mathbf{a}_1$			Second Variable : $\mathbf{a}_2$		
$\mathbf{x}_1$	1.1	0.0	0.0	1.5	0.0	0.0
$\mathbf{x}_2$	1.5	1.0	0.0	2.1	0.0	0.0
$\mathbf{x}_3$	0.2	0.1	0.2	1.7	0.3	0.2
$\mathbf{x}_4$	3.1	0.3	0.0	4.1	0.5	0.0
$\mathbf{x}_5$	2.7	0.0	0.1	3.0	0.0	0.2
$\mathbf{x}_6$	3.5	0.2	0.0	4.7	0.1	0.0
$\mathbf{x}_7$	4.5	0.0	0.0	0.6	0.0	0.0
$\mathbf{x}_8$	4.7	0.0	0.0	0.3	0.0	0.0
$\mathbf{x}_9$	4.6	0.0	0.0	0.6	0.0	0.0

For the representation of **X** in $\mathcal{FST}^2(\mathbf{a})$, p = 6 because each input variable requires 3 numbers for specification as a symmetric trapezoidal fuzzy number. By this device the original set of 18 (2 for each of the 9 input data) membership functions are converted into a set of 9 vectors in $\Re^6$, and this is how the data are fused; all of the HFD inputs are transformed into vectors in 6-dimensional Euclidean space. The nine 6D vectors in Table 4.47 *are* the fused data - that is, the transformation of the inputs to $\Re^6$ accomplishes the fusion. Now we may process the columns of Table 4.47 with any pattern recognition algorithm that uses object data.

Figure 4.87 is a sketch that illustrates the 18 membership functions in $\mathcal{FST}(\mathbf{a})$ that are specified by the HFD set in Table 4.47. The frames in Figure 4.87 are not to scale.Each sketch in Figure 4.87 is 1 unit tall. The horizontal scales vary from one to two units wide so you can only see the approximate relationships of the various data to each other. Moreover, the parameters of the functions are given in a different way than in (4.130) so that the sketches fit in the figure.

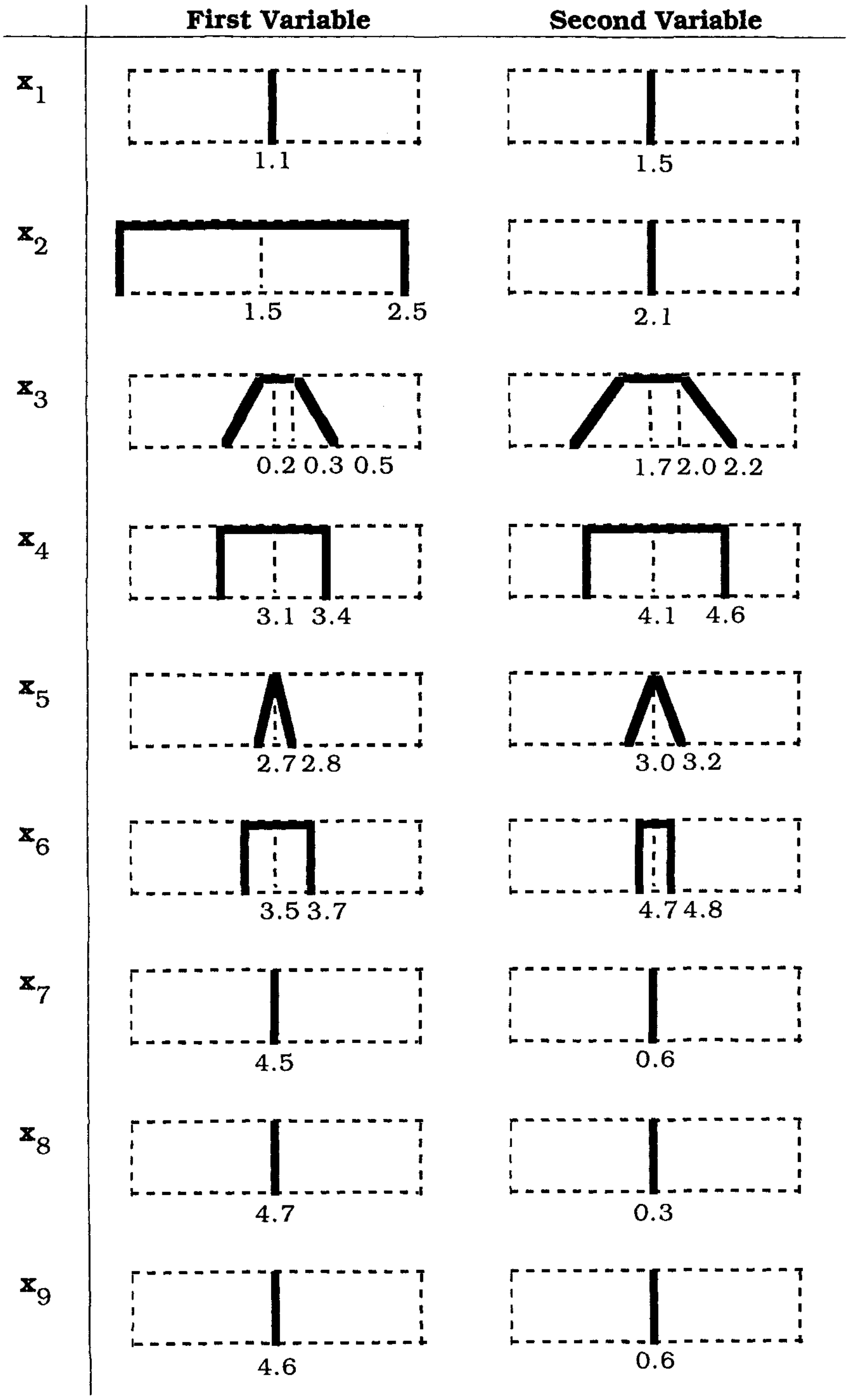


Figure 4.87 Graphical representation of the HFD in Table 4.47

Figure 4.87 illustrates how the parametric HFD model fuses data of the three types discussed in the speed example into a uniform numerical framework. Data points 1,7,8, and 9 are pairs of real numbers; they would be collected by point-valued sensors. $\mathbf{x}_2$ shows a real interval as its first coordinate, and a real number as its second entry; this would result from an instance of the (sensor 2, sensor 1) pair for object 2. $\mathbf{x}_3$ and $\mathbf{x}_5$ could be the result of observations by sensor type 3 on both variables. And so on.

It now makes sense to ask about clusters in HFD. The idea itself is easy to grasp, and the computational means for doing cluster analyses in HFD are available in the setting of $\mathcal{FST}^p(\mathbf{a})$. We can apply *any* object data clustering algorithm to the generalized coordinates of X obtained by this representation, and it will produce clusters. Hathaway et al. applied the fuzzy c-means clustering algorithm (Subsection 2.2.A) to the data in Table 4.47 with m=2, c=3, ε=0.0001 and the Euclidean norm for the objective function. The termination criterion was $E_t = \|U_t - U_{t-1}\|_\infty \le \varepsilon$.

The initialization they used is indicated with subscript 0 in the upper half of Table 4.48, and it grouped points {1,4,7}, {2,5,8} and {3,6,9} into c=3 crisp clusters. The lower half of Table 4.48 contains the final prototypes, indicated by subscript f, which were obtained in 14 iterations of FCM.

Table 4.48 Initial and final FCM prototypes for X in Table 4.47

	First Variable : $\mathbf{a}_1$			**Second Variable : $\mathbf{a}_2$**		
$\mathbf{v}_{1,0}$	2.90	0.10	0.00	2.07	0.17	0.00
$\mathbf{v}_{2,0}$	2.97	0.33	0.03	1.80	0.00	0.07
$\mathbf{v}_{3,0}$	2.77	0.10	0.07	2.33	0.13	0.07
$\mathbf{v}_{1,f}$	0.94	0.31	0.07	1.76	0.10	0.07
$\mathbf{v}_{2,f}$	4.59	0.00	0.00	0.51	0.00	0.00
$\mathbf{v}_{3,f}$	3.17	0.21	0.02	4.11	0.25	0.04

Table 4.49 shows the initial (crisp) and final fuzzy partitions of the 9 HFD data points. Hardening U_f in Table 4.49 with (2.10) leads to the crisp 3-partition (shown by the shaded and bold cells) $X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\} \cup \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\} \cup \{\mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9\}$.

Table 4.49 Initial and final FCM 3-partitions for X in Table 4.47

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$	$\mathbf{x}_7$	$\mathbf{x}_8$	$\mathbf{x}_9$
row 1, U_0	1	0	0	1	0	0	1	0	0
row 2, U_0	0	1	0	0	1	0	0	1	0
row 3, U_0	0	0	1	0	0	1	0	0	1
row 1, U_f	**0.97**	**0.84**	**0.93**	0.01	0.22	0.03	0.00	0.01	0.00
row 2, U_f	0.01	0.06	0.03	0.00	0.11	0.02	**1.00**	**0.99**	**1.00**
row 3, U_f	0.02	0.10	0.04	**0.99**	**0.67**	**0.95**	0.00	0.00	0.00

Since each point in Table 4.47 and Figure 4.87 has two components, it is possible to construct a 3D view of each row in the data that shows the 2D membership function for each object. Figure 4.88 is a 3D plot of the data and final prototypes in which the data have been lightened, and the three prototypes darkened so that you can see them.

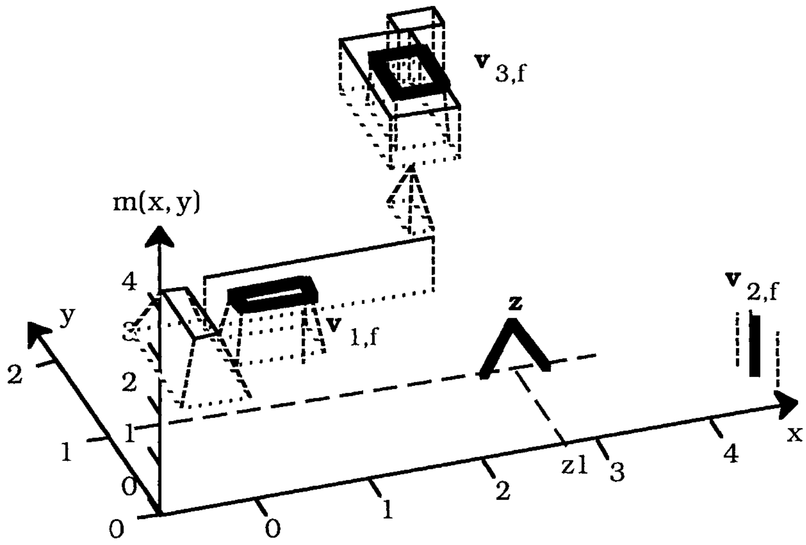


Figure 4.88 The terminal FCM prototypes for c = 3 are used to classify HF data point z

The second prototype (cluster center) corresponds to the point $\mathbf{v}_{2,f}$ = (4.59, 0.51). (Actually, this is true only after the computational results are rounded to two decimal places. To four decimal places, prototype $\mathbf{v}_{2,f}$ = $(4.5868,\ 0.0013,\ 0.0004,\ 0.5137,\ 0.0001,\ 0.0008)^T$, so $\mathbf{v}_{2,f}$ does not, strictly speaking, correspond to a vector in $\Re^2$.) The first and third cluster centers have all non-zero coordinates, and thus define trapezoidal pyramids in $\mathcal{FST}^2(\mathbf{a})$.

Hathaway et al. also showed that the prototypes obtained in this example can be used as a basis for 1-np classifier design. By way of illustration, suppose that the 9 point HFD data are regarded as "training data", and the output of training is the set of HFD prototypes for c = 3 shown in Figure 4.88. Suppose that $\mathbf{z}$ is the triangular membership function in $\mathcal{FST}^2(\mathbf{a})$ specified by the generalized coordinates $\mathbf{z} = (z_1, 0, 0.25, 1.0, 0, 0)^T$ as illustrated in Figure 4.88. As plotted, $\mathbf{z}$ appears visually equidistant from the prototypes for classes 1 and 2. The HFD model enables us to make this a well-defined concept.

An easy calculation of the Euclidean distance from $\mathbf{z}$ to each of the 3 prototypes *in the representation space* $\mathcal{FST}^2(\mathbf{a}) \subset \Re^6$ yields the results in Table 4.50. If the first coordinate of $\mathbf{z}$ is 2.65, the nearest prototype is $\mathbf{v}_{1,f}$, yielding the class 1 label. If the first coordinate for $\mathbf{z}$ is 2.70 (as shown in the third column of Table 4.50), the label assigned by the nearest prototype classifier would be class 2.

Table 4.50 Illustration of a nearest prototype classifier for HF data : tabulated are Euclidean distances $\delta_2(\mathbf{z}, \mathbf{v}_{k,f})$

k	$z_1 = 2.65$	$z_1 = 2.70$
1	1.93	1.98
2	2.00	1.96
3	3.17	3.16

This shows how a classifier can be designed using fused data that will look for the closest match to an unlabeled observation among a library of generalized prototypes. The representation space $\mathcal{FST}^p(\mathbf{a})$ provides the essential ingredient - viz., a unified framework for pattern recognition using models that are already available and well understood. All of the usual problems associated with clustering, cluster validity and nearest prototype classifiers - for example, what distance measure (in $\Re^p$) and how many prototypes per class to use - become sensible questions to ask and try to answer.

B. Feature level fusion

Feature level fusion is much more general and directly takes advantage of the ability of different sensors to measure complementary information. This level of fusion involves combining multi-dimensional, quantitative feature vectors derived from sensor measurements, possibly together with qualitative information. For example, one sensor may give shape information while a different sensor may provide depth. This level of fusion has many similarities to complex pattern classification problems (Wootton et al., 1988, Keller and Hobson, 1989, Keller and Osborn, 1991, Keller et al., 1991).

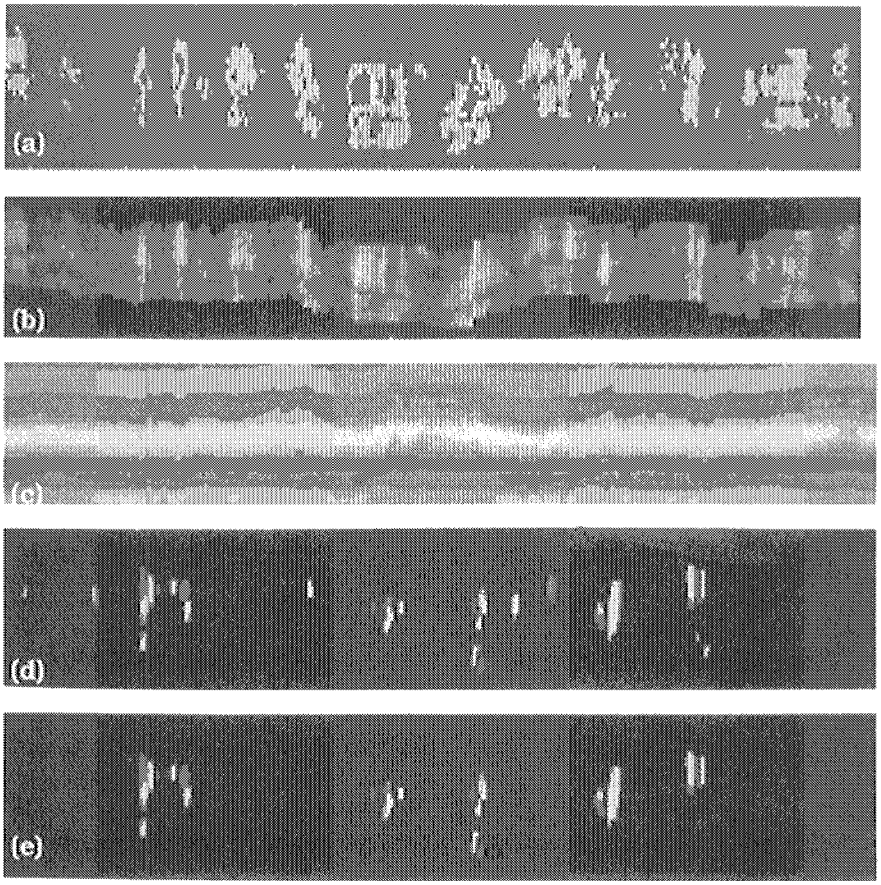


Figure 4.89 Feature level fusion to reduce false alarms

As an example consider the Geo-Centers GPR system, which produces a volume of data as the sensors are moved downtrack. The coordinates are downtrack, cross track and time (which roughly correlates to depth). In Figure 4.89, two views of the Geo-Centers GPR data are displayed: panel (b) is a surface plot where the energy below the surface is "summed up" into a downtrack-crosstrack plane; and panel (c) is a depth plot representing the radar returns in time and downtrack given a fixed crosstrack value. The objects are at locations denoted by very faint "tick marks" in panels (a)-(c) of Figure 4.89. Panel (a) of Figure 4.89 is a thresholded version of the image in panel (b). All of the objects are accounted for, but many false alarms are also evident in panel (a). While the raw data is the same, different features emerge by processing it in different ways. In looking at the depth plots, Frigui et al. (1998a) and Gader et al. (1998b) noticed that the high energy locations in the panel (b) surface plot corresponded (roughly) to rising and falling edges in the panel (c) depth plot at locations related to the size of the objects. Panel (d) of Figure 4.89 represents two thresholded outputs of gradient masks operating on the image in panel (c). Darker values represent strong rising edges, while the lighter color corresponds to falling edges in panel (d). Note that there are still false alarms in panel (d). Hence, by combining these sets of features, we might expect to enhance detection capabilities while eliminating false alarms.

Keller and Gader (1997) proposed one method of combining information on energy (panel (a)) with the information on rising and falling edges (panel (d)) using a fuzzy rule:

IF	there is significant energy in the GPR surface plot.
AND	there are rising and falling edges in the GPR depth plot
AND	the edges are close
THEN	confidence in mine is high

View (e) in Figure 4.89 shows an implementation of the above rule. This particular implementation lost two of the desired detections, but significantly reduced the number of false alarms from the standard approach (panel (a)). Don't read too much into Figure 4.89; it is just an example to illustrate the concept of feature level fusion.

C. Classifier fusion

Decision level fusion generally involves combining information from algorithms that have partially or fully processed individual sensor measurements (or features derived from them), and perhaps, other qualitative information that may reside in rules such as the land mine rule just given. The division between feature level fusion and decision level fusion is not crisp, but decision level fusion is generally considered to be at a higher level, such as combining the outputs of several classifiers. (Tahani and Keller, 1990, Gader et al., 1995a, Gader and Mohamed, 1995, Cho, 1995, Kuncheva et al. 1998). This fusion level is the primary emphasis of this section.

The basic assumption that drives decision level fusion schemes is that classifier algorithms are imperfect. Thus, a good strategy for enhancing the performance of classification systems is to construct multiple independent systems and then combine the results, hopefully achieving higher reliability and robustness through redundancy. The hope is that each individual system makes independent errors which can be overcome using advanced fusion schemes.

Figure 4.90 illustrates a general architecture for *classifier fusion*. In this Figure **D** is any classifier function, $\mathbf{D}: \Re^p \mapsto N_{pc}$. The value $\mathbf{u} = \mathbf{D}(\mathbf{z})$ is the label vector for **z** in $\Re^p$. **D** is a *crisp classifier* if $\mathbf{D}[\Re^p]$ is always a crisp (binary valued) label vector; otherwise, the classifier is fuzzy, possibilistic or probabilistic, which, as we have done earlier, we lump together by the term *soft* classifiers.

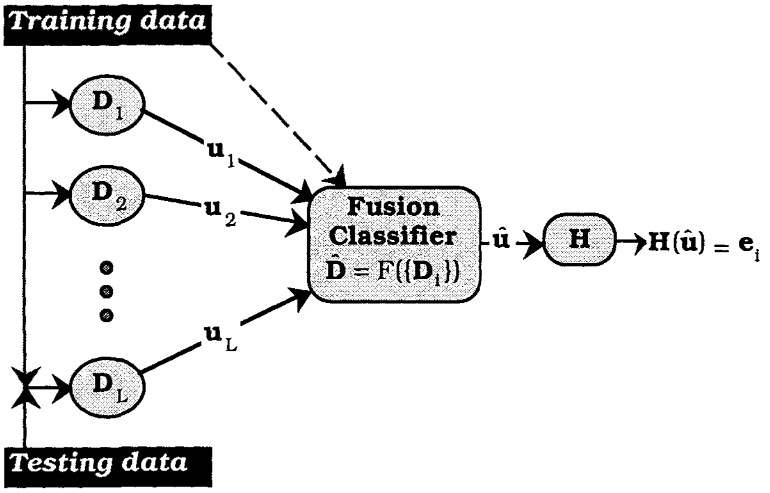


Figure 4.90 Classifier fusion models

On the left in Figure 4.90 we show a bank of L *first level* classifiers - it would not be wrong to conceptualize these as the input layer of a network, and this often helps in visualizing the fusion operation. Each $\mathbf{D}_k$ could be a soft classifier of a different type; for example, $\mathbf{D}_1$ might be a nearest prototype classifier, $\mathbf{D}_2$, a multilayered perceptron, $\mathbf{D}_k$ a fuzzy decision tree, and so on. Most of the time, the L classifiers $\{\mathbf{D}_k\}$ use the same training data to acquire their parameters independently, but for classifier fusion this is not necessary. Exceptions to this include the bagging (Breiman, 1996) and boosting (Drucker et al., 1993) approaches for creating ensembles of neural networks.

The fusion classifier $\hat{\mathbf{D}} = F(\mathbf{D}_1, \ldots, \mathbf{D}_L)$, where F is a specified "fusion operator" that integrates the outputs of the bank of classifiers in the first level, generally maps soft labels to soft labels, $\hat{\mathbf{D}}: N_{pc} \mapsto N_{pc}$. As with other classifiers, we often use the hardening function $\mathbf{H}: N_{pc} \mapsto N_{hc}$ in equation (1.15) to get a final crisp output from $\hat{\mathbf{D}}$ as shown in Figure 4.90. For classifier fusion to improve recognition rates over individual classifier outputs, different classifiers must make different mistakes. A fusion method should emphasize the strengths of individual classifiers and subsets of classifiers, avoid weaknesses, and use dynamically available knowledge about the inputs, the outputs, the classes, and the classifiers.

There are three styles of *classifier fusion.* The simplest type is when $\hat{\mathbf{D}}$ is a fixed operator without any unspecified parameters that cannot be trained and which is simply chosen by the user - e.g., the minimum, maximum, weighted mean, etc.; in this case we call $\hat{\mathbf{D}}$ a *non-trainable fusion operator.* A more aggressive approach to fusion allows $\hat{\mathbf{D}}$ to be trained simultaneously but independently from each of the L $\mathbf{D}_k$'s using the same training data; then we call $\hat{\mathbf{D}}$ a *separately trained fusion operator.* Examples of this type of operator include fuzzy integrals (Tahani and Keller, 1990, Keller et al., 1994a, Gader et al., 1996b, Wang et al., 1998), OWA operators (Cho, 1995), decision templates (Kuncheva et al., 1998, 1999), and many others that we will meet later in this section.

Lastly, if $\hat{\mathbf{D}}$ is trained simultaneously and in conjunction with the L $\mathbf{D}_k$'s using common training data, we call $\hat{\mathbf{D}}$ a *co-trained fusion operator*. There aren't too many examples of this third type of fusion. Jacobs et al. (1991) discuss a mixture of "experts" that use a gating network for $\hat{\mathbf{D}}$ that is trained together with the first level classifiers. However, this model selects (as opposed to combines) classifier outputs, so is not exactly what we call a co-trained fusion model.

As an introduction to *time-based fusion,* Sato et al. (1997) discuss a temporal version of FCM called TFCM (Section 2.6) that integrates T time slices of data sets having fixed spatial sizes in a weighted objective function across time. The output of TFCM is a single fuzzy partition matrix U for the entire set of time slices coupled to T sets of prototype vectors $\{\mathbf{V}_1, ..., \mathbf{V}_T\}$, one set for each slice. When the frame rate of temporal sequences is high, TFCM may be useful for "short bursts" within the overall sequence of temporal data, because changes in the scene will be very slight. In this situation the number of objects (that is, c, the number of clusters in U) should not vary, but the centers of gravity (that is, the prototypes $\mathbf{V}_k$ we use to track the objects) will, if the objects and/or sensor platform have changing positions in time.

There are at least as many methods for classifier fusion as there are for designing classifiers, since the decision fusion mechanism $\hat{\mathbf{D}}$ in Figure 4.90 *is* a classifier. Voting strategies (Mazurov et al., 1987), like majority choice or best "M of N" approaches, and order statistics, like the maximum or median, are obvious simple non trainable methods to fuse multiple classifier outputs.

Kittler (1998) discusses the problem of classifier fusion from a statistical decision theoretic standpoint for two scenarios: fusion of opinions based on identical representations, and opinions based on distinct representations. Standard methods for combining distinct opinions, such as the product rule, sum rule, min rule, max rule majority voting, and weighted averaging are shown to be special cases of compound classification, where all representations are used jointly under suitable Bayesian hypotheses. Kittler (1998) also contains many other statistical based fusion references.

Figure 4.90 looks like (and is, if we regard the L classifier outputs as inputs to a single node) a standard FF neural network, and as such, can be trained on the output of the L classifiers and thus act as a soft fusion technique. We will discuss this method in some detail later in this section, but refer you to Rowley et al. (1998) for a typical example of using a NN as the fusion device.

Additionally, there are methods to choose the best classifier for a particular sample (among say, an ensemble of neural networks, or from different types of techniques) based on some measure of "goodness" or "consistency" of the multiple outputs (Leon et al., 1996, Woods et al., 1997). This is somewhat different in nature from the situation depicted in Figure 4.90. The classifiers aren't combined directly in this method; they are used to determine if classification should be done at all. Refusing to decide (at least until more evidence is forthcoming) can improve overall classification accuracy and is consistent with the principle of least commitment.

Pick your favorite classifier, and you can turn it into a fusion machine. We are being somewhat casual here, because the fact is that fusing the results of classifiers/sensors/information sources is not well understood. There is little theory of sensor fusion. Hence, the "proof is in the pudding" right now, i.e., different approaches can produce very different results on different data (that's a lot of differences - just don't treat them with diffidence). What we will do now is give a few examples of fuzzy set based classifier fusion approaches that have been shown to work in certain domains.

The fuzzy integral (Section 4.5) is a very flexible approach for classifier fusion. Tahani and Keller (1990) were the first to utilize the Sugeno fuzzy integral to combine the results of multiple classifiers. In that paper they established a framework for fuzzy integral fusion in an automatic target recognition application that used three first level classifiers: a Bayes decision theory classifier, a feature based Sugeno fuzzy integral, and a soft prototype classifier based on FCM. The fuzzy integral was able to compensate for two extremely confident (but erroneous) classifications by one of these three classifiers. Subsequently, several authors have used both the Sugeno and Choquet fuzzy integrals to combine multiple information sources (Keller et al., 1994a, Gader et al., 1995a, Gader and Mohamed, 1995, Cho and Kim, 1995, Grabisch et al., 1995).

Example 4.28 In (Hocaoglu et al. 1997, Keller et al., 1998), a system based on fuzzy set theoretic algorithms to perform automatic target recognition from *Laser Radar* (LADAR) imagery was described. Figure 4.91 shows the framework of this approach for an *automatic target recognition* (ATR) system.

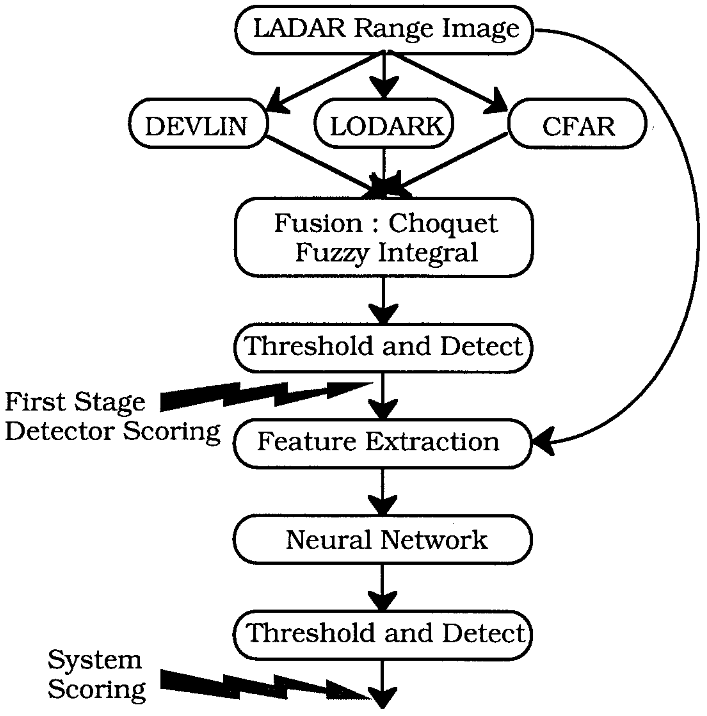


Figure 4.91 A fuzzy logic ATR system

The details of the LADAR pixel target detection filters, called LODARK, DEVLIN and CFAR in Figure 4.91, are in (Hocaoglu et al. 1997, Keller et al. 1998). Briefly: LODARK stands for *LOw and DARK* - LADAR range image targets have more "action" in the low part of the scanning window and correspond to darker pixels than the background. DEVLIN stands for *DEViation from LINear* - the background in range images tends to look like a plane (not airplane of course), hence targets (and other objects) cause a deviation from that flat or linear plane. CFAR stands for *constant false alarm rate* and is usually a size-contrast filter, although this implementation used robust estimators in the size contrast filter (Frigui et al., 1998b).

What is important here is that all three classifiers produced a target confidence at each pixel in the LADAR scene. The Choquet fuzzy integral was used to combine the results of the three classifiers. For one group of experiments, the set of LADAR images was divided into a training set (52 images with 89 targets) and a test set (45 images with 86 targets). See Figures 4.20 and 4.21 for a typical image in this data set. Each pixel level detector was run on the training images and for each threshold value, the probability of detection vs. the number of false alarms was computed (the graph of these points is called a *receiver operating characteristic* (ROC) curve. From the ROC curves on the training data, a threshold was picked for each detector. Each detector was then run with its threshold on the test set and scored. A Sugeno fuzzy measure (see equation (4.47)) was generated from densities calculated as the relative number of detections by each classifier on the training set. In this case, the resultant measure was a probability measure (since the densities summed to one).

The three detector confidences for each pixel in the training images were fused with the Choquet integral, and once again the probability of detection vs. the number of false alarms for all thresholds was computed. An "optimal" threshold was selected (manually) from the training results of the fusion. The results of the three individual and the fused detectors are shown in Table 4.51. On the training data, the Choquet integral combination was able to slightly increase the detection rate while reducing the number of false alarms. Many of the false alarms were generated on just a few "poor" images, and no effort was made to incorporate temporal aspects of the image sequences into the processing.

Table 4.51 Target detection outputs for individual and fused detectors on training data

Detector	# Hits	# False Alarms	Density
CFAR	75 (84.3%)	200	0.32
DEVLIN	80 (89.9%)	227	0.34
LODARK	81 (91.0%)	319	0.34
FUSED	83 (93.3%)	191	-na-

The test images were then submitted to the final configuration. The Choquet fusion scheme for the detectors (using the threshold selected from the training ROC curve) found 81 of the 86 targets in the test images, with 183 false alarms. The second stage detector in Figure 4.91 was added to further reduce the number of false alarms.

In Example 4.28 there were training images, but no "desired outputs" at the pixel level. Hence, the densities for the fuzzy integral were calculated from global statistics. Recently, the Choquet integral has become the fuzzy integral-of-choice for classifier fusion activities where desired outcomes are available. This is because the entire measure can be learned as the optimal solution to a quadratic programming problem (Grabisch and Nicolas, 1994).

Even restricted classes of measures give rise to a wide variety of combination schemes, As noted in Section 4.5, all linear combinations of order statistics (LOS) operators (or OWA fuzzy operators in some circles) are special cases of the Choquet fuzzy integral. Tumer and Ghosh (1998) show that a LOS combination of multiple neural networks provides excellent fusion of classifiers in the presence of outliers, or when there is a high variance of individual classifier performance. They performed an analysis of decision boundaries and ran experiments on 6 standard data sets from the University of California (Irvine) repository to support their views. This study lends support to the Choquet fuzzy integral combination as a very competitive fusion method.

An application where classifier fusion has received considerable attention is handwriting recognition. This is because there are an abundant number of classifier schemes for character and word recognition, along with a huge amount of labeled training and testing data. Classifier fusion methods in this domain include intersection of decision regions, voting, prediction by top choice combinations, Dempster-Shafer theory of evidence, fuzzy integrals, neural networks and rule-based approaches (Ho et al., 1994; Huang & Suen, 1995; Keller et al., 1994a, Suen et al., 1992, Xu et al., 1992, Chi and Yan, 1996, Chi et al., 1996b).

Handwritten word recognition is problematic because of the large variations in the shape of characters, the illegibility and ambiguity present in many handwritten characters, and the overlapping and interconnecting of neighboring characters. In most applications the size of the lexicons (dictionaries) is large and the contents of the lexicons (the classes) are changing. The problem is more complex than traditional pattern recognition problems because the number of classes is relatively large - easily on the order of thousands, and moreover, changes from word image to another. This precludes the use of some decision combination methods that depend on knowing the number of classes and the identity of each class in advance.

One widely used fusion method in handwritten word recognition is the *Borda count*, which is simple to implement and requires no training. All classifiers rank all of the alternatives (classes), and the Borda count is simply the sum of the ranks for each class. In this method, however, all classifiers are treated equally, which may not be preferable when certain classifiers are more likely to be correct than others. Ultimately, more sophisticated techniques are necessary for fusion in this domain because different word recognizers do not contribute equally and do not place equivalent restrictions on the recognition results. For example, a weighted Borda count was shown to achieve better performance than the unweighted Borda count in (Ho et al., 1994). The next example, taken from (Gader et al., 1996b) demonstrates the ability of the fuzzy integral to effectively combine classifier outputs for handwritten word recognition.

Example 4.29 In a test of classifier fusion for word recognition, Gader et al. (1996b) considered three advanced recognition algorithms: a dynamic programming algorithm that is applied to segmented characters, which we call the *segmentation-based method* (SBM, see example 4.12), a *hidden Markov model* (HMM) approach, and a *fuzzy version of the hidden Markov model* (FHMM) method (Mohamed and Gader, 1994, Mohamed and Gader, 1995). The decision fusion strategies all use the ranks of the strings provided by each word recognizer. The HMM and FHMM schemes do not produce output confidence values that can be compared to each other, or to those produced by the SBM. Only the relative ranks of the HMM and FHMM for various words in the lexicon are comparable. The SBM approach used two MLP neural networks (one each for upper and lower case letters) to generate character confidences for unions of primitive segments. The neural networks were trained with handwritten characters that were assigned possibilistic training labels (Gader et al., 1995), which produced better results in word recognition than those from neural networks trained with crisp character labels. The use of ranks provides a measurement which is comparable across recognizers. Gader et al. used only the top n (n= 5, here) strings in the lexicon for each recognizer. For a given word image and lexicon, each classifier produced an ordering of the lexicon. The kth string in each ordering of the lexicon is assigned the rank confidence 1 - (k/n). If k > n, the rank confidence is defined to be 0.

Recall that the Borda count associated with a string in a lexicon is defined as the sum of the ranks, while the weighted Borda count is the weighted sum of the ranks. The weights can be fixed for every classifier, or they can be a function of the match confidence (degree of match) between the image and the lexicon string. Data dependent or data independent approaches can also be used to generate the density values for the fuzzy integrals.

The fuzzy integrals used the rank confidences for the values of the function h(x). The density values were generated using two methods. In the first method, Gader et al. assigned each classifier a fixed density value which was used for every string in every lexicon; this value is considered to reflect the worth of each classifier. An example of the non-data dependent method for combining word classifiers are shown in Tables 4.52(a) and 4.52(b). This example is difficult - can you figure out what the correct word should be from this pair of tables? Did you guess the word "island" - *before* you read the caption of Table 4.52(a)? This is the correct word.

Table 4.52 (a) Three classifier rankings for an image of the word "island"

Rank	HMM	FHMM	SBM
1.0	"grant"	"stpaul"	"island"
0.8	"island"	"grant"	"grant"
0.6	"granada"	"island"	"salem"
0.4	"burwell"	"oneill"	"nehawka"
0.2	"nehawka"	"o'neill"	"roseland"

The three classifiers were run on an image of the word "island" from the SUNY (1989) postal database, and the five rows of Table 4.52(a) correspond to the top five words as ranked by each of the three classifiers. Note that the word "island" appears in the top three choices of each classifier, but is the top choice of only one of them. Actually, the word "grant" seems to be a better guess from the information in Table 4.52(a). Table 4.52(b) shows the results of fixed weight fusion for the top five classes (words) appearing in Table 4.52(a). The Borda count uses no weight factors. For the other three schemes (weighted Borda count, Sugeno integral and Choquet integral) the weights/densities were chosen as 0.65 for SBM, 0.25 for HMM and 0.05 for FHMM. As seen in Table 4.52(b), the weighted Borda count and the Choquet integral pick the correct class for this example.

Table 4.52 (b) Results of classifier fusion on the results in Table 4.52(a)

String	Borda Count	Weighted Borda Count	Sugeno Integral	Choquet Integral
"grant"	2.60	0.81	0.8	0.85
"island"	2.40	0.88	0.8	0.92
"nehawka"	0.60	0.31	0.4	0.32
"salem"	0.60	0.39	0.6	0.26
"granada"	0.60	0.15	0.25	0.15

The second method discussed in Gader et al. (1996b) used data dependent densities. The confidence value produced by the SBM was used to define a density value for it in the fusion scheme. The density values for the HMM and the FHMM were then determined by a heuristic formula involving the SBM confidence and the agreement between the classifiers concerning the rank of each string. More precisely, let:

C_S = confidence value from the segmentation-based classifier
g^S = density of the segmentation-based classifier
r_S = rank of the string by the SBM
g^H = density of the HMM classifier
r_H = rank of the string by the HMM
g^F = density of the FHMM classifier
r_F = rank of the string by the FHMM

If a string is in the top n choices of the segmentation-based system, then define

$$g^S = \max(\varepsilon, \alpha \cdot C_S) \quad ; \qquad (4.131a)$$

$$g^H = \beta \cdot \sqrt{(1 - C_S) \cdot (1 - |r_H - r_S|)} \quad ; \text{ and} \qquad (4.131b)$$

$$g^F = \gamma \cdot \sqrt{(1 - C_S) \cdot (1 - |r_F - r_S|)} \quad . \qquad (4.131c)$$

Otherwise, define

$$g^S = \max(\varepsilon, \alpha \cdot C_S) \qquad (4.132a)$$

$$g^H = \beta \cdot \sqrt{(1 - C_S) \cdot (1 - |r_H - r_F|)} \qquad (4.132b)$$

$$g^F = \gamma \cdot \sqrt{(1 - C_S) \cdot (1 - |r_F - r_H|)} \qquad (4.132c)$$

Here α, β, and γ are parameters that can be optimized and $\varepsilon > 0$ is very small. The expression $\max(\varepsilon, \alpha \cdot C_S)$ was used to keep the densities for the SBM non-negative in the case that an appropriate segmentation cannot be found. The same method can be used to define weights for the weighted Borda count.

For example, the data-dependent Choquet combination method would assign confidence values shown in Table 4.53 for the strings "island" and "grant" from Table 4.52(a). The values of the parameters used for the computations shown in Table 4.53 were $\alpha = 0.9$, $\beta = 0.1$, and $\gamma = 0.4$.

Table 4.53 Example of data-dependent Choquet integral fusion

String	C_S	g^S	g^H	g^F	Choquet Integral
"island"	0.75	0.68	0.02	0.04	0.92
"grant"	0.46	0.42	0.27	0.08	0.86

The experiments were performed on handwritten words from the SUNY CDROM database. The "BD city" words were used (Hull, 1994). The FHMM and HMM were trained using the standard training set from that database. The SBM method was trained using a different set of data. In the SBM experiment, 126 words from the training set were used to "train" densities and weights. All of the 317 BD city names from the test set were used for testing. Sets of lexicons that had an average length 100 were used for both training and testing. The results of the three individual classifiers are shown in Table 4.54.

Table 4.54 Recognition results for individual classifiers

Classifier	Training	Testing
HMM	74.6%	71.6%
FHMM	74.6%	73.2%
SBM	82.5%	83.9%

The "training" method for the weighted Borda count and the fixed density fuzzy integral approaches was a "modified" exhaustive search. Weights/densities were varied from 0.05 to 0.95 by increments of 0.05. The training method that was used for the data-dependent densities was a similarly modified exhaustive search on α, β, and γ. In each case, "optimal" values for the parameters were found on the training set and then used on the test set. The top choice results for the Gader et al. (1996b) experiments are shown in Table 4.55.

Table 4.55 Training and test results for fused classifiers

Combination Approach	Optimal Training	Testing
Data-Dependent Choquet	89.7%	88.0%
Data-Dependent Sugeno	89.7%	86.4%
Data-Dependent Weighted Borda	88.9%	85.5%
Fixed Choquet	88.1%	82.0%
Fixed Sugeno	88.1%	85.2%
Fixed Weighted Borda	88.1%	86.4%
Borda	84.1%	83.3%

Gader et al. (1996b) attempted to train several standard MLP neural networks to fuse classifiers from the same data that was used by the fuzzy integral. Each feature vector contained ten inputs: the segmentation confidence (C_S), the word ranks for the three classifiers (r_S, r_H, r_F), the data dependent densities for the three classifiers (g^S, g^H, g^F), and the fuzzy measures of the three 2-element subsets of classifiers. Recall that for a fuzzy measure defined over the set of three information sources (in this case, the three classifiers), there are eight subsets to consider. Leaving off the measure of the empty set, which is 0, and that of the whole set, which is 1, the fuzzy measure is completely specified by the measures of the three singleton sets (these are the densities above), and the measures of the three subsets containing two of the three sources. Hence, the neural networks had as input the segmentation confidence, the classifier outputs, and the fuzzy measure. The target was set to 0.9 if the string represented the correct choice for the current word image, and 0.1 if it was incorrect. Many architectures were investigated. Table 4.56 shows the best results obtained.

Table 4.56 Training and test results : neural nets with crisp outputs

Architecture	# Iterations	Training Results	Testing Results
10:5:1	1000	84.1%	80.4%
10:5:1	3000	84.9%	82.3%
10:5:1	6000	84.9%	81.4%
10:5:1	21000	86.5%	79.5%
10:10:1	2000	83.3%	81.4%
10:10:1	4000	83.3%	81.4%
10:10:1	10000	86.5%	81.7%
10:10:1	15000	88.1%	80.8%
10:10:5:1	5000	84.9%	82.0%
10:10:5:1	9000	86.5%	81.4%

It is clear from Table 4.56 that the neural network architectures did not match the performance of the fuzzy integral for fusing the three classifiers on this data set. Gader et al. conjecture that this may be true in handwritten word recognition because we are not learning a nonlinear function in the same sense that standard pattern recognizers do - i.e., we are not hacking through Dubois and Prade's "jungle of function approximation". Since strings need to be ranked, there are a very large number of possible classes and hence, we cannot use the standard class coding approach. This makes the task for a neural network extremely difficult.

Siy and Chen (1974) wrote one of the first papers about the application of fuzzy models to handwritten character recognition. Like Chang and Pavlidis (1977), this paper contained precursors of some elements of many papers to follow, including those of Chi et al. (1996b) and Chi and Yan (1996). Although the language of syntactic pattern recognition is not used in Siy and Chen, some of the material on this topic that we will present in Section 4.10 is closely related to ideas in this paper, so we take a little stroll down Siy and Chen lane, as it were, to check out the scenery.

Siy and Chen argued that all handwritten characters were distorted versions of printed characters, and that all alphanumeric characters comprised essentially three basic "strokes": the line, a portion of a circle, or a whole circle. They suggested the set of 15 "features" shown in Figure 4.92, made from one of the three basic strokes, as a basis for decomposition and subsequently, recognition, of the various characters in any alphabet. In Section 4.10 we will call these 15 arcs the *primitives* of a *grammar* for syntactic approaches to handwriting analysis. Shown directly beneath the symbolic name of each primitive is a 2-digit number that will be used to encode the prototypical description of each character.

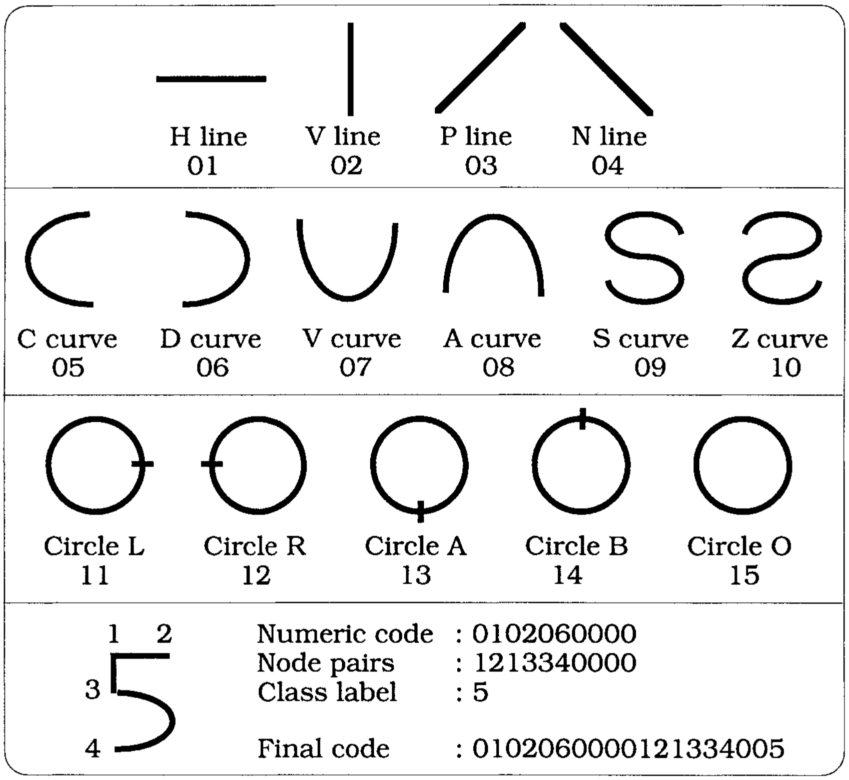


Figure 4.92 The 15 branch features (primitives) of Siy and Chen

A character is represented by three strings of numbers; the first string is made by concatenating digit pairs (e.g., 01=H line, 02 = V line, etc.) in ascending order; the second string encodes the node pairs needed to specify the stroke sequence, ordered to connect the digit strings in the first pair; and the third string is a class label (this is how training and test data are labeled). A functional representation of the digit "5", using $(\cdot)$ to indicate concatenation, is $5 = H(1,2) \cdot V(1,3) \cdot D(3,4)$, which indicates three things: the sequence of strokes and type of strokes (H and then V and then D), and the sets of node pairs ((1,2) and then (1,3) and then (3,4)). Using this scheme, for example, the numeral "5" will be encoded as shown in the lowest panel of Figure 4.92.

Siy and Chen skeletonize the binary character images by a thinning algorithm (see examples 5.6 and 5.14, and also, e.g., Gonzalez and Woods, 1992). Next, a set of nodes in the skeleton is found. Nodes can be tips, corners, and junctions (strokes with 1, 2 or more than 2 edges incident to them, respectively). See Figure 5.17 for an illustration of a corner and a junction (called a triple point in Figure 5.17 because there are 3 edges incident to the node).

A *branch* b is an arc (element of the skeleton) connecting a pair of adjacent nodes. Branches are classified by two attributes; their straightness and orientation. To illustrate, suppose a branch b is extracted. At this point b might be a line, or it might be a curve. Consequently, Siy and Chen determine the best fit (minimum least squared error) line to the points along the skeleton b. Once this is done, b is classified by computing its "straightness", which is defined as its membership in the fuzzy set "nearly lines", defined as

$$m_L(S) = \begin{cases} 1 - (S / S_T) & ; S < S_T \\ 0 & ; S \geq S_T \end{cases} \quad , \tag{4.133}$$

where S is the fitting error of the best fit line and S_T is a threshold on the fitting error. If $S = 0$, $m_L(S) = 1$. Thus, when the fitting error is zero, b is a line, and otherwise, b departs from linearity to some extent. Branch b is classified as a curve if $0 \leq m_{SL}(b) < 0.5$; and otherwise, b is declared a line.

To handle the orientation of b, Siy and Chen define membership functions for each of the four line segments (H, V, P and N) in Figure 4.92. For example, the membership function for the horizontal H line in Figure 4.92 is $m_H(\theta) = 1 - \min\{\min\{|\theta|, |180 - \theta|, |360 - \theta|\}/45, 1\}$, where $\theta = \tan^{-1}(m)$ is the angle of inclination in degrees of the best fit line (whose slope is m) to the branch b under consideration. If the branch b passes the straightness test in (4.133) so it is declared a line, b is then assigned a crisp membership in the set whose branch membership function maximizes this subgroup of 4 membership functions. In our notation, each branch that is declared linear is associated with a possibilistic label vector $\mathbf{u}(b) \in N_{p4}$ whose entries are computed with the four "line type" membership functions, and then branch b is assigned to a crisp line type by hardening the possibilistic label, $b \in$ line type i $\Leftrightarrow \mathbf{H}(\mathbf{u}(b)) = \mathbf{e}_i$, where i takes values 1 to 4, as say, the line type runs through H, V. P and N.

Siy and Chen define 6 membership functions that are similar to the ones used for orientation of lines for non-linear cases. If a branch b is declared a curve by equation (4.133), these membership functions are used to label b as one of the remaining 11 non-linear feature types. Eventually, every branch in a character skeleton is crisply labeled as one of the 15 primitives in Figure 4.92.

Aiming towards a 1-np classifier that has the flavor of equation (4.2), Siy and Chen assign a crisp class label to each character in the training data, and then run it through the above decomposition, finally obtaining a 3 string prototype for each training character. Since each training data produces a prototype, the prototypes will be correctly labeled automatically. Moreover, as there will be many variations of the same symbol, there may be several distinct prototypes for a single character. The measure of "nearest" that was chosen instead of the metric δ in (4.2) was exact string matching, bit by bit, against the strings derived to represent an input datum. Siy and Chen use the relative frequencies of occurrence of each prototype to make the search for a matching prototype during testing and operation of the character recognizer a little more efficient. Remember, this was 1974, and matching n_{tr} prototypes to a long string for a lot of test samples could be computationally arduous.

As described, the 1-np classifier implemented by Siy and Chen has a "reject" option - that is, the system has three output categories during testing: correct if one prototype exactly matches the input and the labels agree; incorrect, if one prototype exactly matches the input and the labels disagree; and no decision when no prototype matches the input. The training data discussed by Siy and Chen consisted of 50 samples for each of the 10 integers 0, 1, ..., 9, so their system produced n_{tr} = 500 prototypes for the 10 characters. Then the system was tested on a set of 500 unseen samples called the "Honeywell 500" data by Siy and Chen. On this test data, their simple 1-np classifier obtained a success rate of 98.4% correct - that is, 8 of the 500 test characters were labeled incorrectly - three 9's, two 0's and one each of the numbers 3, 4 and 5. As we pointed out at the beginning of Section 4.3, nearest prototype classifiers are simple, effective and cool. Granted that the data set used by Siy and Chen is small, this example still seems to bear out our assertion. Siy and Chen (1974) does not exemplify a fusion technique: we discussed this paper here to prepare for the next group of papers, which consider the same topic, and that do use classifier fusion. Now we spin forward to 1996, and see how much better we can do with all the latest neural gadgets and fusion techniques at our disposal.

Earlier in this section we mentioned that standard FF neural networks can be used for classifier/sensor fusion. Next we discuss a set of four papers by Chi and Yan (1995, 1996) and Chi et al. (1995, 1996b) that all use multilayered perceptrons (MLPs) as a principle component of classifier design. The two 1995 papers discuss single classifiers, while the two 1996 papers have two classifiers in the first level, and an MLP is used as the fusion classifier at the second level. One of the primary applications that we have been using to illustrate fusion so far - handwritten digit recognition - is the focus of all four papers, and all four use the same data set. After we discuss the four papers, we will combine their results in a single example - Example 4.30.

All four papers base their examples and discussion on the same database, identified as the United States *National Institute of Standards and Technology* (NIST) special database number 3, which consists of handwritten segmented characters. And all four papers use the same data sets X_{tr} and X_{te} for training and testing of the classifiers developed in them. The cardinalities of X_{tr} and X_{te} are equal, both being 10,426 crisply labeled samples of the 10 digits 0, 1, ..., 9. The features that are derived from the NIST database differ in the four papers: the 1995 papers are based on feature vectors in $\Re^{64}$, while the two 1996 papers use feature vectors in $\Re^{36}$. We will not report many details of the feature extraction and classifier design methods for each of these papers, but we do want to convey the basic flavor in each of them.

Chi et al. (1995) and Chi and Yan (1995) use functions of the pixel counts from 8×8 subwindows in the 64×64 image of each digit in the database to obtain 64 input features as the basis for the design of a fuzzy rule based classifier. The fuzzy rules in both 1995 papers are found by first running a self-organizing feature map (SOFM, see Section 4.3.D) on X_{tr} to generate prototypes from the training data; in both papers, the SOFM display space is a square 2D grid of various sizes. Then, the SOFM prototypes are used to generate triangular premise membership functions (PMFs) using a variant of a membership function generation method due to Dickerson and Kosko (1993). Finally, fuzzy rules of the Takagi-Sugeno (TS) type are extracted from the training data using a well known method due to Wang and Mendel (1992). Both 1995 papers use product aggregation (T_2= product) as in (4.72c) for aggregation of the LHS of each rule to get its firing strength.

The major difference between the two 1995 papers is the inferencing method used during defuzzification when an input is submitted to the TS rule base. It is easier to describe the inferencing procedure used in both papers by abandoning the formalism of label vectors, so we will cast these classifiers in the notation of 4.6.D, i.e., using

S_{TS} instead our standard notation $\mathbf{D}$, to denote classifier outputs, and instead of crisp label vectors for the output functions, we use the digits 0, 1, ..., 9, which correspond to crisp labels for each sample in the training and test data.

Unlike Chiu (1994), Chi et al. (1995) do use the standard TS defuzzification formula shown in (4.73). Since there are M rules, with M >> c = 10 classes, many rules will have the same crisp label - one of the 10 digits from 0 to 9 - as their right hand sides. Since (4.73) always makes a convex combination of the output functions by combining them with the associated firing strengths, the result of using this formula in the present instance is to produce a number in the closed interval [0, 9] for each input datum. With the notation just established, (4.73) takes the form

$$\hat{S}_{TS}^{CWY}(\mathbf{z}) = \frac{\sum_{i=1}^{M} \alpha_i(\mathbf{z}) j_i}{\sum_{j=1}^{M} \alpha_j(\mathbf{z})} = u \in [0,9],\ \ j_i \in \{0,1,\ldots 9\}\ \forall\, i \quad . \tag{4.134}$$

The real number in [0,9] is now converted into one of the 10 crisp labels, and the TS system becomes a crisp classifier by computing

$$S_{TS}^{CWY}(\mathbf{z}) = \left\lfloor \hat{S}_{TS}^{CWY}(\mathbf{z}) + 0.5 \right\rfloor \quad , \tag{4.135}$$

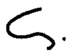
where $\lfloor * \rfloor$ again denotes the floor of its argument.

In Chi and Yan (1995), the same basic classifiers use sets of 64 input features as given in Chi et al. (1995), but the method of inference in the fuzzy rule base is changed. Chi and Yan (1995) use a 3 layer feed forward network. The first layer generates the fuzzy membership function values. The number of nodes in the hidden layer is equal to the number of rules, and the output of the kth hidden node is equal to the firing strength of the kth rule. The output layer has 10 nodes, one for each of the 10 digits 0, 1, ..., 9. The ith output node combines the output of the hidden layer nodes as follows:

$$u_i(\mathbf{z}) = F_L\left(\frac{\langle \mathbf{w}_i, \alpha_i(\mathbf{z}) \rangle}{\sum_{j=1}^{M} \alpha_j(\mathbf{z})} \right), i = 0, 1, \ldots, 9 \quad , \tag{4.136}$$

where F_L is the logistic function at (4.97) with $\lambda = 1$ and $\beta = 0$. Unlike (4.135), this network produces $\mathbf{S}_{TS}^{CY}(\mathbf{z}) = (u_0(\mathbf{z}), \ldots, u_9(\mathbf{z}))^T$, a vector output. Chi and Yan learn the weights $\{w_{ij}$: i = 0, 1, ..., 9; j = 1, 2, ..., M$\}$

by the usual back propagation method, so this is essentially a single layer perceptron whose inputs are normalized firing strengths. Once the weight vectors are found, (4.136) is used to classify test inputs. Chi and Yan call this an *optimized fuzzy rules* (OFR) classifier; they do not specify how $\mathbf{S}_{\mathrm{TS}}^{\mathrm{CY}}(\mathbf{z})$ is hardened to produce crisp labels, so we presume they use the same strategy as in (4.135).

In the 1996 papers the fuzzy rule base is obtained by a completely new method, and, as we have mentioned, the features also change. Both 1996 papers base their features on Siy and Chen's (1974) shape features that are shown in Figure 4.92. Chi et al. (1995) modify this set of features just a bit in that they use the 4 lines and 6 arcs that are shown in the upper and middle panels of Figure 4.92, but the 5 circles in Figure 4.92 are replaced by 2 shapes. The eleventh shape used in these two 1996 papers is circle O as shown in Figure 4.92, but the four circles L, R, A and B in Figure 4.92 are replaced in these two papers by a twelfth primitive that is simply called "curve", which is shaped like this: . The same membership function, equation (4.133), that Siy and Chen proposed in 1974 is used in both of these papers to assess the extent to which a given segment is linear.

Working on the presumption that a given numeral can be well characterized by its 6 longest segments, Chi et al. (1996b) extract a total of 36 numerical features for each datum from its 6 longest segments, and convert the training and test data in the NIST database into these feature vectors; Chi and Yan (1996) use these same 36 features, which are obtained as follows. The four basic features are computed: *type* of segment (this is a symbolic feature which is one of the 12 shape descriptors), normalized segment length, and normalized coordinates of the center of gravity of the segment relative to the center of gravity of the thinned skeleton of the digit. For up to 6 segments per skeleton, this makes 24 features. Added to this are 12 more features: number of segments in this digit, numbers of end points in each of four quadrants whose axes lie at the center of gravity of the thinned skeleton of the digit, normalized total length, the center of gravity of the thinned skeleton of the digit, numbers of lines, circles and curves, and aspect ratio of the image. Notice that this list of 36 features has one symbolic feature, 8 integer-valued features, and 27 continuously valued features. A typical crisp ID3 decision tree rule extracted from the training data using these features looks like this:

IF	the type for longest segment is circle
AND	the type for second longest segment is C curve
AND	normalized y coord. of skeleton centroid is > 0.586
THEN	digit = 6

Chi and Yan (1996) discuss a method for fuzzifying the crisp ID3-derived rules that is applicable to symbolic, discretely-valued, and continuously-valued real data. They effectively quantize the n distinct values of each of the p features in the training data using trapezoidal membership functions, so that when ID3 is applied to the training data, the crisp ID3 rules are well defined over continuos ranges of values that capture the training data. Figure 4.93 illustrates their fuzzification of the kth internal node v_k after the edges have been established with ID3 using the n values of the i-th feature in the training data.

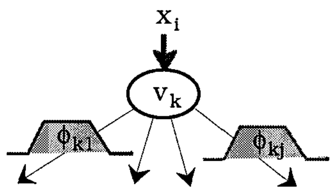


Figure 4.93 Internal node k in Chi and Yan's fuzzy decision tree

Each edge leaving v_k is associated with one or more values of the i-th feature. Chi and Yan span the values with a trapezoidal membership function that is either single sided (one value moves along the exit edge) or double sided (some range of training values flow through the exit edge). This construction lies conceptually somewhere between the approach of Umano et al. (1994), who defined discrete premise membership functions, and that of Zeidler et al. (1996), who use trapezoidal membership functions on the exit edges (i.e., as premise membership functions for the fuzzy rules).

While Zeidler et al. (1996) make estimation of the trapezoidal premise membership functions part of the training problem, Chi and Yan (1996) simply define trapezoidal PMFs with one fixed, user defined constant that adjusts the slope of the sides of each trapezoid as a function of the examples passing through that node during tree building with ID3. Chi and Yan state that the choice of the slope (there is only one parameter for the whole tree) is problem dependent. Since the training features flowing through each edge of the tree have different values, the node functions $\{\phi_{kj}\}$ will not be identical, but they all have the same functional form. Letting $[x_{kj,\min}, x_{kj,\max}]$ denote the interval spanned by the training data along edge kj and s be the user-defined "slope" of the trapezoid, the PMFs all have the form

$$\phi_{kj}(z) = \left\{ \begin{array}{ll} 1 & ; x_{kj,\min} < z < x_{kj,\max} \\ \dfrac{z-(1-s)\cdot x_{kj,\min}}{s \cdot x_{kj,\min}} & ; (1-s)\cdot x_{kj,\min} \le z < x_{kj,\min} \\ \dfrac{(1+s)\cdot x_{kj,\max} - z}{s \cdot x_{kj,\max}} & ; x_{kj,\max} < z < (1+s)\cdot x_{kj,\max} \\ 0 & ; \text{otherwise} \end{array} \right\}. \tag{4.137}$$

Once the ID3 tree has been fuzzified, Chi and Yan again compute firing strengths along its paths using the T2 or product norm shown in equation (4.72c). Now Chi and Yan's tree is in the structural form of the Chang-Pavlidis tree shown in Figure 4.39, each leaf v_{L_i} possessing two pieces of information, $\alpha_i(\mathbf{z})$, the firing strength or decision value along the path from v_1 to v_{L_i}, and $\mathbf{e}_{j_i}$, one of the c crisp label vectors for the classes in the training data.

The way this tree operates on an input $\mathbf{z}$ is interesting and novel. Recalling that M is our notation for the number of leaves (or number of rules) in such a tree, we let $\boldsymbol{\alpha}(\mathbf{z}) = (\alpha_1(\mathbf{z}),\ldots,\alpha_M(\mathbf{z}))^T$ denote the M-vector of firing strengths obtained by traversing the tree from its node to the M leaves. Chi and Yan (1996) define the output vector of their decision tree as

$$\mathbf{S}_{DT}^{CY}(\mathbf{z}) = \left(\left\langle \mathbf{w}_1, \boldsymbol{\alpha}(\mathbf{z})\right\rangle,\ldots,\left\langle \mathbf{w}_c, \boldsymbol{\alpha}(\mathbf{z})\right\rangle\right)^T \quad , \tag{4.138}$$

where the c weight vectors $\{\mathbf{w}_i\} \subset \Re^M$ are weight vectors of the output nodes of a 2 layer feed forward neural network that has 10 output nodes (for the 10 digits 0, 1, ...,9). Estimation of the $\{\mathbf{w}_i\}$ by backpropagation using the same training data as was used to generate the crisp ID3 tree also involves a set of user defined fuzzy relational matrices that are given in Chi and Yan (1996). The node functions in the output layer are hyperplanes whose weight vectors are the c vectors needed in (4.138) to make it operational.

The elements of the right hand side in (4.138) are *not* guaranteed to lie in [0, 1]. To make $\mathbf{S}_{DT}^{CY}$ classify inputs, Chi and Yan interpret the jth element of the output vector, $\left\langle \mathbf{w}_j, \alpha(\mathbf{z})\right\rangle$, as the degree to which input $\mathbf{z}$ belongs to class j, j = 0,...,9. Labeling decisions are then made by simply hardening this vector, i.e., by selecting the class (index of the vector $\mathbf{S}_{DT}^{CY}$) corresponding to the maximum value of $\left\langle \mathbf{w}_j, \alpha(\mathbf{z})\right\rangle$, thereby obtaining a crisp decision. This fuzzy rule-based crisp classifier is one of the two classifiers used in both Chi and Yan (1996) and Chi et al. (1996b); we will call this classifier $\mathbf{D}_{DT}^{CY}$.

The second classifier used in Chi and Yan (1996) is a standard 1-nmp design as in equation (4.7), based on prototypes obtained from the training data using Yan's (1993) method called *optimized prototypes* (OP) (see our discussion of this in Section 4.3.D). We will denote these prototypes by $\mathbf{V}_{OP}$, and denote this classifier by our standard notation, $\mathbf{D}^{CY}_{\mathbf{V}_{OP},\mathbf{E},\delta}$. The second classifier in Chi et al. (1996b) is based on *hidden Markov models* (HMM). We don't want to stray too far from classifier fusion, so we simply denote this classifier as $\mathbf{D}^{CSY}_{HMM}$, the CSY standing for the authors, Chi, Suters and Yan, and refer you to their paper for details about their HMM models.

Both 1996 papers then fuse the two classifiers in them by using a standard MLP as the fusion classifier, say $\hat{\mathbf{D}}_{NN}$. The input layer to $\hat{\mathbf{D}}_{NN}$ has 21 nodes (10 for the outputs from each of the first level classifiers, plus one node to introduce the bias constant in all the hidden layer nodes). Inputs from the fuzzy rule base classifier $\mathbf{D}^{CY}_{DT}$ lie in the range [0,1], while inputs from the other two classifiers ($\mathbf{D}^{CY}_{\mathbf{V}_{OP},\mathbf{E},\delta}$ and $\mathbf{D}^{CSY}_{HMM}$) are normalized to lie in the same range. The hidden layer in $\hat{\mathbf{D}}_{NN}$ had standard node functions $\Phi_{LH} = F_L \circ f_H$ as in Section 4.7. Each unipolar sigmoid is fixed, with $\lambda = 1$ and $\beta = 0$. The output node functions were not parametrized, and had unipolar sigmoids with $\lambda = 1$ and $\beta = 0$. The final output of $\hat{\mathbf{D}}_{NN}$ was, we presume, hardened in the same way as the "optimized fuzzy rules neural network". $\hat{\mathbf{D}}_{NN}$ was trained to acquire weight vectors for the hyperplanes in the hidden layer nodes with the same training data as used for all other classifiers, using the outputs from the first level and their target labels as IO data, much in the manner of the hybrid method of training RBF networks that we discussed in Section 4.8. $\hat{\mathbf{D}}_{NN}$ follows the standard scheme we called separately trained fusion at the beginning of this section. Now we are ready to discuss the results reported in the four papers.

Example 4.30 This example combines recognition rates from tables in the four papers (Chi and Yan (1995, 1996) and Chi et al. (1995, 1996b). The Euclidean norm is used for both the 1-nn and 1-nmp rules. Table 4.57 shows the % correct classifications in both training (resubstitution) and testing (generalization) on the NIST data described above for all the classifiers discussed in the four papers. FR in this table stands for fuzzy *rules*. We repeat the description of the data here for convenience. The database was the *National Institute of Standards and Technology* (NIST) special database number 3, which consists of handwritten segmented characters. And all four papers use the same data sets X_{tr} and X_{te} for training and testing of the classifiers developed in them. The cardinalities of X_{tr} and X_{te} are equal, both being 10,426 crisply labeled samples of the 10 digits 0, 1, ..., 9.

Table 4.57 is divided into 4 sections, corresponding to the classifiers discussed in each of the four papers, and when a result was repeated in one of the source papers, we do not repeat it here. The first three rows of Table 4.57 concern results reported in Chi et al. (1995), which used three one stage classifiers, viz., the crisp 1-nn rule at (4.2), the crisp 1-nmp rule at (4.7) using 809 SOFM prototypes from a 30×30 2D display grid which had 91 inactive cells, and S_{TS}^{CWY}, the fuzzy rule based classifier defined in (4.135). According to Chi et al. (1995), the *10,401* rules generated by the method used to design S_{TS}^{CWY} with 225 SOFM prototypes did not cover all of the inputs (there was not a rule for every possible LHS of the TS system). Because of this, it was possible for a test input to not match any rule, thereby making all M firing strengths zero. This causes the denominator in (4.134) to be zero. When this occurred, the test input was declared "unsure" by S_{TS}^{CWY}, and was forwarded to a backup 1-nmp classifier that used SOFM prototypes. The backup classifier in the third row of Table 4.57 had a 15×15 display grid, and apparently all 225 prototypes were active.

We believe the reason that S_{TS}^{CWY} produced so many rules is that there is a problem with the use of (4.134) and (4.135). Suppose only two rules fire : rule 1 has firing strength 0.8 for the output "7", while rule 2 fires with strength 0.8 and output label "4". Using (4.134) in this case results in $\hat{S}_{TS}^{CWY}(\mathbf{z}) = (0.8 \cdot 7 + 0.8 \cdot 4)/(0.8 + 0.8) = 5.5$, so the output of (4.135) will be the integer "6 = $\lfloor 5.5 + 0.5 \rfloor$". This problem is the result of using numerical proximity instead of structural proximity as the underlying rationale for defuzzification, and can result in needing a rule for almost every training datum.

The crisp 1-nn rule is perfect in resubstitution, as it must be, and can be viewed as a benchmark of sorts for all the other classifiers shown in Table 4.57. The TS based classifier S_{TS}^{CWY} nearly matches this performance in resubstitution, but is 1.2% less in testing. The second block of Table 4.57 shows the outputs of four single stage classifiers. The first row in this set of four is a standard 1-nmp rule classifier based on 395 active prototypes in a grid of 400. This is the same design as in the second row of the first sub block, but it uses many fewer prototypes, and the decrease in recognition rates is ascribed to this. The second classifier is a TS system with 395 rules derived by the three stage procedure reported above (SOFM, PMFs, Wang-Mendel training) that uses the standard TS defuzzification in equation (4.73). This row catches the eye, being some 30% less accurate than any other classifier in this table. Following this is the result of using a standard 65:30:10 multilayered perceptron, which does pretty well on the training data, but falters a little on the test set. The last classifier in this group is S_{TS}^{CY}, based on 395 optimized fuzzy rules that uses the defuzzification in (4.136). To their credit, Chi and Yan graciously point out the standard MLP perceptron does a little better than this design, and offer some possible explanations for this in the 1995 paper.

Table 4.57 Training and test results for classifier designs from four papers : Chi et al. (1995, 1996b) and Chi and Yan (1995, 1996)

Reference	Classifier	Train Results	Test Results
Chi et al. (1995) (64 features)	1-nn	100	98.0
	1-nmp/SOFM (809 active in 900 prototypes)	98.7	96.3
	S_{TS}^{CWY} (10,401 FRs) with backup: "unsure" with SOFM 1-nmp using 225 prototypes	99.99	96.8
Chi &Yan (1995) (64 features)	1-nmp/SOFM (395 prototypes)	97.4	95.0
	FR (395 fuzzy rules with (4.73))	67.9	64.7
	MLP (65:30:10) neural net	99.98	97.1
	S_{TS}^{CY} (395 OFRs)	99.0	96.3
Chi et al. (1996b) (36 mixed features)	ID3 (2163 leaves)	98.8	91.4
	ID3 (pruned, 828 leaves)	97.1	91.9
	ID3 (simplified, 151 leaves)	94.6	90.7
	$\mathbf{D}_{DT}^{CY}$ = fuzzy ID3 (151 fuzzy rules)	97.7	95.0
	$\mathbf{D}_{HMM}^{CSY}$ = hidden Markov models	93.6	92.8
	$\mathbf{D}_{\mathbf{V}_{OP},\mathbf{E},\delta}^{CY}$ (1000 "optimal" prototypes)	98.4	97.8
FUSION A	$\hat{\mathbf{D}}_{NN} = F(\mathbf{D}_{DT}^{CY}, \mathbf{D}_{hmm}^{CSY})$ F = MLP (21:20:10)	99.97	97.8
FUSION B	$\hat{\mathbf{D}}_{NN} = F(\mathbf{D}_{DT}^{CY}, \mathbf{D}_{\mathbf{V}_{OP},\mathbf{E},\delta}^{CY})$ F = MLP (21:20:10) with backup: "unsure" using 1-nmp $\mathbf{D}_{\mathbf{V}_{OP},\mathbf{E},\delta}^{CY}$ with 1000 optimized prototypes	98.8	**98.6**
Chi & Yan (1996) (36 mixed features)	ID3 (fuzzy, cont. feat. only)	96.0	93.0
	ID3 (fuzzy, symbolic and discrete. feat. only)	97.3	93.8
	$\mathbf{D}_{DT}^{CY}$ = fuzzy ID3 (all features)	97.7	95.0
FUSION C	$\hat{\mathbf{D}} = F(\mathbf{D}_{DT}^{CY}, \mathbf{D}_{\mathbf{V}_{OP},\mathbf{E},\delta}^{CY})$ F = MLP (21:20:10)	99.6	**98.6**

The third group of classifiers that appear in Table 4.57 are the ones discussed in Chi et al. (1996b). The first two rows in this group correspond to the crisp ID3 decision tree of Section 4.6.C, with and without pruning. Notice that the error rates of these two classifiers bears out the general supposition that pruning crisp decision trees increases generalization at the expense of training accuracy. The third row reports the success of a two stage simplification of the crisp ID3 tree using a method due to Quinlan (1987). Rows 4-6 in this block show the recognition rates of the fuzzy ID3 tree described above, the hidden Markov models classifier, and a 1-nmp rule classifier that uses 1000 of Yan's (1993) optimized prototypes. Row 7 shows the results of using fusion method A, the 21:20:10 neural network that we described earlier, which is used to fuse the hidden Markov model and fuzzy ID3 designs. Row 8 of this third group shows fusion method B, which is fusion method A augmented by a backup option for "unsure" results that is forwarded to a 1-nmp rule for resolution with $\mathbf{D}^{CY}_{\mathbf{V}_{OP},\mathbf{E},\delta}$, the classifier of row 6. We see that this method, fusion with backup, produces a test rate of 98.6% correct, the best so far.

The final subset of four rows in Table 4.57 reports the results discussed by Chi and Yan (1996). The first row in this group reports the error of the fuzzified ID3 tree developed in Chi et al. (1996b) on just the 27 continuously valued features of each training sample. Performance on just these features is not so good, and the second row shows that the same type of fuzzy tree built with only the 8 integer and 1 symbolic features is better - an interesting result, that the classifier is more successful using these 9 non-continuos features than 27 continuously valued numbers. The fuzzified ID3 tree using all 36 features displayed in the third row of this subset shows a slight increase in performance from the use of either of the feature subsets shown in the first two rows of this group. And finally, the last row in Table 4.57 shows the performance of fusion method C, which is a model that combines the classifier $\mathbf{D}^{CY}_{\mathbf{V}_{OP},\mathbf{E},\delta}$ shown in row 6 of the third subset of rows in Table 4.57 with $\mathbf{D}^{CY}_{DT}$, the classifier in row 3 of the fourth subset of rows. The fusion model is again a 21:20:10 built and trained as described above. Chi and Yan's (1996) fusion model $\hat{\mathbf{D}} = F(\mathbf{D}^{CY}_{DT}, \mathbf{D}^{CY}_{\mathbf{V}_{OP},\mathbf{E},\delta})$ ties Chi et al.'s (1996b) fusion with backup model, also achieving a recognition rate of 98.6 % correct in testing.

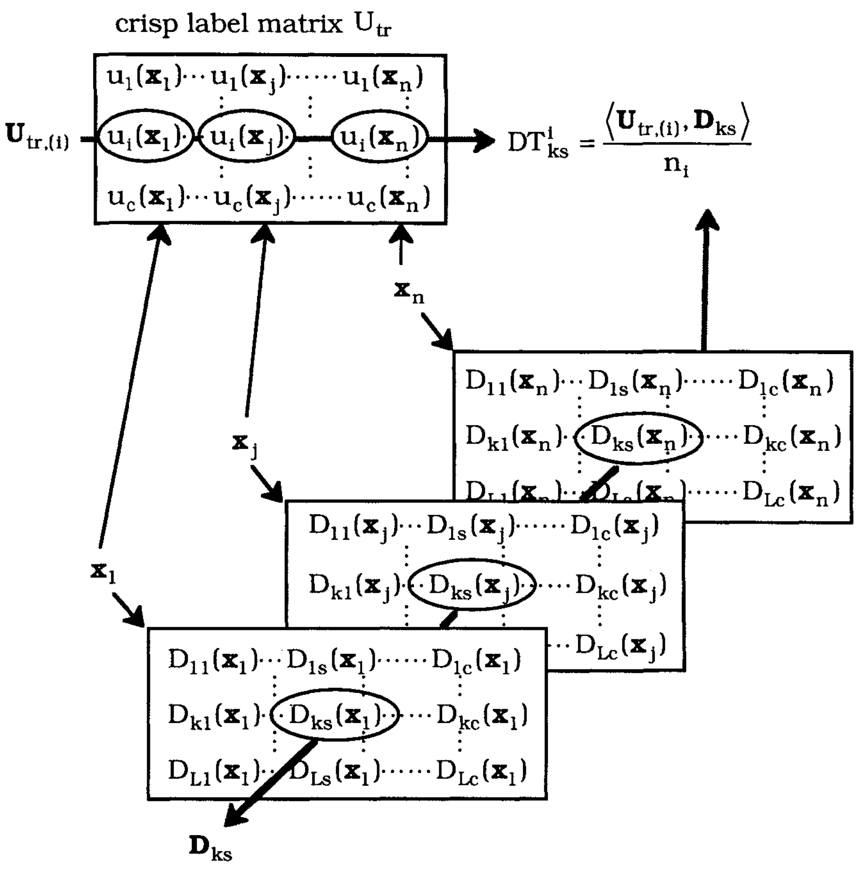
To summarize, two fusion models do improve the performance of all of the systems that depend on single stage classifiers reported in the four papers on which this example is based. At 98.6% correct, they both do a little better on test than the 1-nn rule at 98.0% correct. About this we make two observations: first, a k-nn rule with $k > 1$ might do better; second, these results involve only one data set, and one set of samples for training and testing, so take our usual caution seriously - the data really control your results.

Lest you think we mean to degrade the results in Table 4.57, we point out that an improvement of just 0.6 % in this test set corresponds to an improvement of 0.006*10,426 = 63 digits. If a zip code reader was being used by the United States Social Security office, and your social security check was being processed for automatic mailing by an optical scanner based on Chi and Yan's model instead of the 1-nn rule after the letter had been addressed by hand, this slight improvement would mean that, on average, about 63 more checks in each set of 10,000 would reach the correct destination. Everything is relative - in the handwritten digit recognition business, every digit counts.

As a final observation, return to 1974, and look at the test error reported in Siy and Chen's paper for single character recognition - 98.4% correct, using parameters from their 15 shape features and a standard 1-np classifier. Admittedly, the 500 test data of Siy and Chen are a far cry from the 10,426 test data of Chi and Yan (1996), and again, both are the results of just one training and testing cycle. How far have the "intelligent" architectures we have developed in the 22 years between these two papers really taken us? You be the judge.

The last fusion method we discuss was introduced by Kuncheva et al. (1995) under the original name of *fuzzy templates*. Various improvements and changes during the evolution of this fusion model can be traced through Kuncheva (1998) and Kuncheva et al. (1998), and in the latest iteration of the model (Kuncheva et al., 1999), it is called *decision templates*, so that is the name we will use here.

As usual, we begin with the assumption that we have a set of crisply labeled training data, $X_{tr} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \Re^p$, and as in earlier sections, we denote the crisp $c \times n$ partition matrix whose columns are the label vectors of the points in X_{tr} by $U_{tr} \in M_{hcn}$. This method begins with the construction of a set of c $(L \times c)$ prototype or template matrices, say $\{DT^i: i = 1, \ldots, c\}$. DT^i is called the *decision template for class i.* The training data and the L first level classifiers $\{\mathbf{D}_i\}$ are used non-iteratively in the construction of the decision templates, and when we have these matrices, the second level fusion classifier *is* trained. Let $D_{ks}(\mathbf{x}_j)$ denote the output of classifier k for class s on training input $\mathbf{x}_j$, so k = 1 to L, s = 1 to c and j = 1 to n. If we submit all n training data to the L classifiers, we have a total of $Lc \cdot n$ values. Imagine a 3D array of these classifier output values that shows (Lc) of them as a function of the index on the training data, as in Figure 4.94, which illustrates the fundamental construction pictorially.



crisp label matrix U_{tr}
$u_1(\mathbf{x}_1)\cdots u_1(\mathbf{x}_j)\cdots u_1(\mathbf{x}_n)$
$\mathbf{U}_{tr,(i)}$
$u_i(\mathbf{x}_1)\cdots u_i(\mathbf{x}_j)\cdots u_i(\mathbf{x}_n)$
$u_c(\mathbf{x}_1)\cdots u_c(\mathbf{x}_j)\cdots u_c(\mathbf{x}_n)$
$DT^i_{ks} = \frac{\langle \mathbf{U}_{tr,(i)}, \mathbf{D}_{ks} \rangle}{n_i}$
$\mathbf{x}_n$
$\mathbf{x}_j$
$\mathbf{x}_1$
$D_{11}(\mathbf{x}_n)\cdots D_{1s}(\mathbf{x}_n)\cdots D_{1c}(\mathbf{x}_n)$
$D_{k1}(\mathbf{x}_n)\cdots D_{ks}(\mathbf{x}_n)\cdots D_{kc}(\mathbf{x}_n)$
$D_{Lc}(\mathbf{x}_n)$
$D_{11}(\mathbf{x}_j)\cdots D_{1s}(\mathbf{x}_j)\cdots D_{1c}(\mathbf{x}_j)$
$D_{k1}(\mathbf{x}_j)\cdots D_{ks}(\mathbf{x}_j)\cdots D_{kc}(\mathbf{x}_j)$
$D_{Lc}(\mathbf{x}_j)$
$D_{11}(\mathbf{x}_1)\cdots D_{1s}(\mathbf{x}_1)\cdots D_{1c}(\mathbf{x}_1)$
$D_{k1}(\mathbf{x}_1)\cdots D_{ks}(\mathbf{x}_1)\cdots D_{kc}(\mathbf{x}_1)$
$D_{L1}(\mathbf{x}_1)\cdots D_{Ls}(\mathbf{x}_1)\cdots D_{Lc}(\mathbf{x}_1)$
$\mathbf{D}_{ks}$

Figure 4.94 Construction of the decision templates

Now we construct the (k,s)th element of the ith decision template by taking the n-vector $\mathbf{U}_{tr,(i)}$, the i-th row of U_{tr} (recall our notation for rows of matrices as vectors, and that $|X_{tr,i}| = n_i \ \forall i$), dividing it by n_i, the number of elements in class i in the training data, and then computing its Euclidean inner product with the n-vector $\mathbf{D}_{ks} = (D_{ks}(\mathbf{x}_1), \ldots, D_{ks}(\mathbf{x}_n))^T$. Do this Lc times - i.e., run k and s over the n L×c frames shown in Figure 4.94. This generates the L×c matrix DT^i, which is the decision template for class i. The vectors $\{\mathbf{D}_{ks}\}$ and the matrix DT^i depend only on the training data X_{tr}, indicated formally by

$$DT^i_{ks}(X_{tr}) = \frac{\langle \mathbf{U}_{tr,(i)}, \mathbf{D}_{ks}(X_{tr}) \rangle}{n_i}, \ k = 1,\ldots,L; \ s = 1,\ldots,c \quad . \tag{4.139}$$

Suppose that $\mathbf{z}$ in $\Re^p$ is an input to the system and that $\mathbf{D}_i(\mathbf{z}) \in N_{pc}$ is the label vector produced by classifier $\mathbf{D}_i$, i = 1,...,L. It is immaterial what kind of label vector each classifier produces - some might be crisp, some fuzzy, or probabilistic, or possibilistic. Thus, the decision templates scheme accommodates "mixed" types of classifiers automatically. To use the decision templates to fuse the outputs of the first level classifiers, we construct one more L×c matrix, DP($\mathbf{z}$), the *decision profile* of the input $\mathbf{z}$, by arraying the L label vectors $\{\mathbf{D}_i(\mathbf{z})\}$ as the rows of the desired matrix,

$$DP(\mathbf{z}) = \begin{bmatrix} \leftarrow (\mathbf{D}_1(\mathbf{z}))^T \rightarrow \\ \vdots \\ \leftarrow (\mathbf{D}_k(\mathbf{z}))^T \rightarrow \\ \vdots \\ \leftarrow (\mathbf{D}_L(\mathbf{z}))^T \rightarrow \end{bmatrix}_{L\times c} . \tag{4.140}$$

Equation (4.140) enables us to describe the construction in Figure 4.94 succinctly: the decision template DT^i for class i is just the average of the decision profiles $\left\{DP(\mathbf{x} \in X_{tr,i}\right\}$ of the elements of the training data that have the i-th crisp label. While the construction in (4.140) is clear, it does not show you the crucial idea that makes the matrix DP($\mathbf{z}$) useful for fusion, which is that its c *columns* correspond to the opinions of the L classifiers about each of the classes. That is, the ij-th entry in DP($\mathbf{z}$) is the support of classifier i for class j. To emphasize this we show (4.140) in a pictorial way in Figure 4.95:

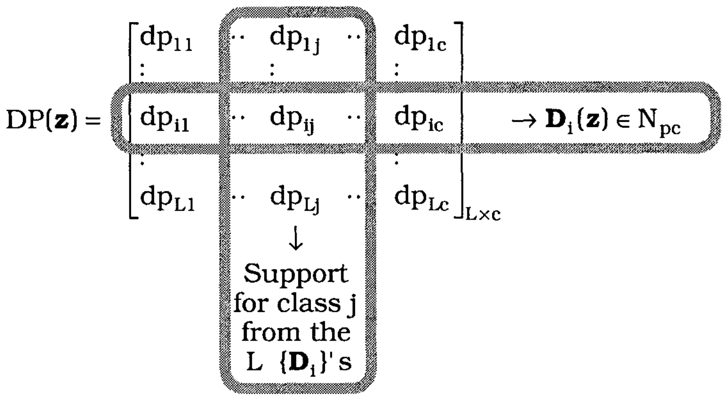


Figure 4.95 Construction of the decision profile

The important point made by Figure 4.95 is that it is the *columns* of the decision profile matrix that carry information about the c classes. *Many* fusion schemes (fuzzy integrals, Dempster-Shafer, etc.) use the columns of DP($\mathbf{z}$), but most authors that use these approaches do not frame their models in the language of matrices because there is no advantage in doing so. The idea of bundling the information used by decision templates and the decision profile into matrix form so an entire $L \times c$ matrix DT^i can carry information about support for class i began with Kuncheva et al. (1995); as we shall see, this leads to some new fusion models.

We have not discussed a fusion operator F that specifies $\hat{\mathbf{D}}$ yet, and interpretation of the decision profile as in Figure 4.95 allows us to create many column-wise or *class-conscious* schemes that accomplish fusion of the L first level classifiers. To get a feel for the idea, we illustrate the use of DP($\mathbf{z}$) with five aggregation operators which use the rows or columns of the decision profile directly to fuse the outputs of the L first level classifiers. We illustrate the five aggregation operators that appear in Example 4.31 below: *majority* (MAJ), *maximum* (MAX), *minimum* (MIN), *average* (AVR) and *product* (PRO). Here is an example of how these five operators work. Suppose L = 6, c = 3 and the decision profile for a particular input is

$$DP(\mathbf{z}) = \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.9 & 0.1 & 0.0 \\ 0.3 & 0.4 & 0.3 \\ 0.7 & 0.1 & 0.8 \\ 1.0 & 0.0 & 0.0 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} . \tag{4.141}$$

From the rows of this matrix we see that $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}_3$ and $\mathbf{D}_6$ are fuzzy classifiers, $\mathbf{D}_4$ is a possibilistic classifier, and $\mathbf{D}_5$ is a crisp classifier. To compute the majority vote, we harden the 6 *rows* of DP($\mathbf{z}$) in (4.141) with $\mathbf{H}$ at (1.15), obtaining 1 vote for class 2 (from row 3), 1 vote for class 3 (from row 6), and the remaining 4 votes for class 1, yielding class 1 as the majority. This will be the fused output using MAJ, and, as we shall see, the evidence supporting class 1 in this DP is pretty strong.

The other four aggregation operators illustrated in Example 4.31 first operate on each *column* of the decision profile separately, producing new label vectors in N_{pc} across the three classes, which are then hardened with $\mathbf{H}$ to get a crisp decision. The outputs of these four operators on DP($\mathbf{z}$) in (4.141) are illustrated in Table 4.58, where the final label for z is class 1 in all four cases.

Table 4.58 Using aggregation with the decision profile

$$\mathrm{DP}(\mathbf{z}) = \left|\begin{array}{ccc} 0.5 & 0.4 & 0.1 \\ 0.9 & 0.1 & 0.0 \\ 0.3 & 0.4 & 0.3 \\ 0.7 & 0.1 & 0.8 \\ 1.0 & 0.0 & 0.0 \\ 0.2 & 0.3 & 0.5 \end{array}\right|$$

$$\begin{array}{lccccc} & \downarrow & \downarrow & \downarrow & & \\ \mathrm{MAX} & (1.0, & 0.4, & 0.8) & \mathbf{H} \rightarrow & 1 \\ \mathrm{MIN} & (0.2, & 0.0, & 0.0) & \mathbf{H} \rightarrow & 1 \\ \mathrm{AVR} & (0.6, & 0.22, & 0.28) & \mathbf{H} \rightarrow & 1 \\ \mathrm{PRO} & (.02, & 0.00, & 0.00) & \mathbf{H} \rightarrow & 1 \end{array}$$

Kuncheva et al. (1999) give a three-way classification of fusion operators based on: (i) the way the rows and/or columns of the decision profile are used; (ii) the type of labels produced by the first level classifiers; and (iii) the type of training employed by the second level classifier. *Class conscious* (CC) operators use the information in the decision profile DP(**z**) "class by class" - that is, column by column - when making a decision about how to label **z**. *Class indifferent* (CI) fusion operators use information in the decision profile "indiscriminately" in the sense that the class by class information is either ignored or integrated across all classes - that is, all of the information in DP(**z**) is used by each decision template. Thus, in this classification scheme the primary discriminant between fusion methods is whether a fusion model does or does not use all Lc entries of DP(**z**) when considering the evidence for a single class label.

For example, the four aggregation operators in Table 4.58 use only column i of DP(**z**) to construct an overall assessment (the corresponding entry of the output label vector) of the support for class i, so these four operators are class conscious, but the MAJ operator, which uses only the rows of DP(**z**), is class indifferent. Another distinction between some of these operators is that AVR and PRO use all the values in each column of DP(**z**), while MIN and MAX use only one, so these latter two aggregations are more sensitive to outliers and noise than the AVR and PRO operators (cf. the discussion following equation (2.93), where we pointed out that Dunn's (1974a) separation index is too sensitive to outliers because of its reliance on MIN and MAX). AVR is not much better, since it has a zero breakdown point (one outlier can destroy the estimate, as illustrated by Sketch B in Section 2.3.H).

When at least one of the L first level classifiers produces soft label vectors, Kuncheva et al. (1999) classify $\hat{\mathbf{D}}$ as follows:

CC1 Class conscious (without second level training) for $\hat{\mathbf{D}}$. Examples include MAX, MIN, AVR, PRO and OWA (*ordered, weighted aggregation* operators, Yager and Kacprzyk, 1997 and Kuncheva, 1997).

CC2 Class conscious (with second level training) for $\hat{\mathbf{D}}$. Examples include the: *probabilistic product* PPR (Tax et al., 1997; Kuncheva, 1998); *fuzzy integral* (FI) as discussed above; and trained linear combinations (Hashem, 1997).

CI2 Class indifferent (with second level training) for $\hat{\mathbf{D}}$. Examples include the: *linear discriminant classifier* (LDC), *quadratic discriminant classifier* (QDC), and *Fisher's linear discriminant* (FLD), all of which are discussed by Duda and Hart (1973), the *logistic classifier* (LOG, Anderson, 1982), Dempster-Shafer (DS) aggregation (Rogova, 1994), neural networks (Jordan and Xu, 1995) and the *decision templates* (dt) approach being discussed here.

If all L classifier outputs are crisp *before* aggregation (either because the L classifiers are themselves crisp, or because soft outputs are hardened before aggregation, the distinction between class consciousness and class indifference disappears. Fusion methods for exclusively *crisp* (denoted here as "C") first level classifiers are classified by Kuncheva et al. (1999) according to the type of training at the second level:

C1 No second level training for $\hat{\mathbf{D}}$. An example is MAJ.

C2 Second level training for $\hat{\mathbf{D}}$. Examples include the : *behavior knowledge space* (BKS) method (Huang and Suen, 1995) and *naive Bayes* (NB) combination (Xu et al., 1992).

Example 4.31 will contain almost all of the fusion operators given as examples in these categories. In the decision templates approach, the action of the classifier $\hat{\mathbf{D}}: N_{pc} \mapsto N_{pc}$ in Figure 4.90 is defined as

$$\hat{\mathbf{D}}(\mathbf{z}) = (\hat{D}_1(\mathbf{z}), \ldots, \hat{D}_c(\mathbf{z}))^T \in N_{pc} \quad , \qquad (4.142)$$

where each element $\hat{D}_i(\mathbf{z})$ in the fused soft label vector $\hat{\mathbf{D}}(\mathbf{z})$ is computed as some function s of the pair of matrices DP($\mathbf{z}$) and $DT^i(X_{tr})$,

$$\hat{D}_i(\mathbf{z}) = s(DT^i(X_{tr}), DP(\mathbf{z})), \ i = 1, \ldots, c \quad . \qquad (4.143)$$

Kuncheva et al. (1998) interpret the function $s: \Re^{Lc} \times \Re^{Lc} \mapsto \Re^{+}$ as a general "similarity measure". Equation (4.143) makes it clear why the decision templates model is class indifferent: *all Lc entries* of both the decision profile of $\mathbf{z}$ and the i-th decision template derived from X_{tr} are used to produce an estimate of the single value in this equation. This stands in contrast to class conscious fusion models such as the Dempster-Shafer and (some) fuzzy integrals, which use only the L entries in the i-th column of DP($\mathbf{z}$) to build an estimate of a value for class i that is conceptually equivalent to $\hat{D}_i(\mathbf{z})$ - i.e., that is the fused estimate of support for the i-th label. Keller and Osborn (1996) discuss a method for training a classifier based on the Sugeno fuzzy integral wherein all class information is used, and this is in some sense a bridge between the categories listed above. Some genetic algorithm approaches for training fuzzy integral measures also utilize the classifier information across classes in a similar way (Wang et al., 1998).

Included in Group CI2 are 11 decision template models that are realized by making specific choices for s in (4.143). These measures are discussed in Kuncheva et al. (1999) and appear in Example 4.31, where they are all abbreviated by dt:s in Table 4.59. These 11 decision template fusion models can be divided into four subcategories, depending on the specific function used for s: s1 to s4 are proper similarity measures; I1 to I5 are inclusion indices; dt:c uses a consistency index; and dt:np is the 1-np rule. We will not discuss all 11 measures here, but will illustrate each of the four subgroups with one example.

The most obvious way to use the $\{DT^i\}$ is to regard them as prototypes in $\Re^{Lc}$, the vector space of all decision profile matrices. We might indicate this explicitly in the notation adhered to throughout this book by renaming the matrices $\{DT^i\}$ as $\{\mathbf{v}_i^{DT}\}$, and (conceptually) regarding them as vectors in $\Re^{Lc}$. Then if we regard any decision profile computed by (4.140) as a vector in this space, the 1-np classifier, equation (4.2) can be used to choose the best match (nearest prototype) from among the c prototypes. In the context of equation (4.2) "best" is defined by choosing any metric δ, here on $\Re^{Lc}$, and when that is done, by defining $\mathbf{V}^{DT} = \{\mathbf{v}_i^{DT} : i = 1, \ldots, c\}$, we have all the elements of the standard 1-np classifier, but now operating as a trained, second level fusion classifier, $\hat{\mathbf{D}} = \mathbf{D}_{\mathbf{V}^{DT}, \mathbf{E}, \delta}$. Since there are two infinite families of metrics on any vector space (induced by the inner product and Minkowski norms on $\Re^{Lc}$), this produces two infinite families of fusion models, one for each such δ. The structure of the decision templates fusion model for this choice of matching is exactly that of the single layer competitive learning network shown in Figure 4.11, and when the metric is an inner product, the second level classifier $\hat{\mathbf{D}} = \mathbf{D}_{\mathbf{V}^{DT},\mathbf{E},\delta}$ is piecewise linear as in Figure 4.5. This choice for s in (4.143) using the Euclidean norm for δ in equation (4.2) is dt:np in Table 4.59.

The four similarity measures used by Kuncheva et al. (1999) that are seen in Table 4.59 are based on regarding the Lc entries of the pairs of matrices used in (4.143) as membership values of two fuzzy sets (formally, this can be done, since all Lc entries in DP(**z**) and each $DT^i(X_{tr})$ lie in [0,1]). Thus, any measure of similarity between pairs of discrete membership functions can be used in (4.143) - for example, most of the measures discussed in Pal and Bezdek (1994). As an example, s1 in Table 4.59 is computed as

$$s1(DT^i(X_{tr}), DP(\mathbf{z})) = \frac{\sum_{k=1}^{c}\sum_{j=1}^{L}\min\{dt^i_{jk}, dp_{jk}\}}{\sum_{k=1}^{c}\sum_{j=1}^{L}\max\{dt^i_{jk}, dp_{jk}\}} \quad , \tag{4.144}$$

which is the ratio of the relative cardinalities of the intersection to the union of the two "fuzzy sets". The five measures called I1 to I5 in Table 4.59 are called *indices of inclusion* (Dubois and Prade, 1980). For example, I1 is defined as

$$I1(DT^i(X_{tr}), DP(\mathbf{z})) = \frac{\sum_{k=1}^{c}\sum_{j=1}^{L}\min\{dt^i_{jk}, dp_{jk}\}}{\sum_{k=1}^{c}\sum_{j=1}^{L} dt^i_{jk}} \quad , \tag{4.145}$$

which presumably measures the extent to which the i-th decision profile DT^i is included in the intersection of DT^i with DP(**z**) when the entries of these matrices are regarded as fuzzy sets. And finally, there is one *consistency index* in Table 4.59,

$$c(DT^i(X_{tr}), DP(\mathbf{z})) = \underbrace{\max}_{j,k}\left\{\min\{dt^i_{jk}, dp_{jk}\}\right\} \quad . \tag{4.146}$$

With a little thought, you will be able to supply many other plausible choices for the fusion operator F = s. For example, the correlations between DP(**z**) and the L DT's are easily computed as the cosine of the angle between (each of) them (essentially the dot product of matrices).

There are two more models in Table 4.59 that we have not mentioned, the *oracle* (OR) model (Woods et al., 1997) and the *single best* (SB) models. The error rate of the oracle is defined as follows: during testing, if any of the hardened outputs of the L first level classifiers is correct for a given input, count the oracle output as correct for that input. Thus, oracle is (almost always) an empirical upper bound on the possible improvement that can be realized by fusing the outputs of more than one classifier. We say *almost always* because it is not impossible to arrange the outputs so that the fused error rate is less than oracle.

To see this, suppose that the decision profile from a set of two possibilistic classifiers for some input $\mathbf{z}$ which is known to have a crisp class 2 label is $DP(\mathbf{z}) = \begin{bmatrix} 0.4 & 0.3 \\ 0.6 & 0.5 \end{bmatrix}$. If both rows of DP($\mathbf{z}$) are hardened (that is, the outputs of both first level classifiers are hardened), each produces the crisp label for class 1, so the oracle is wrong. Suppose the decision templates for this case are the matrices $DT^1 = \begin{bmatrix} 0.7 & 0.1 \\ 0.5 & 0.0 \end{bmatrix}$ and $DT^2 = \begin{bmatrix} 0.3 & 0.4 \\ 0.4 & 0.5 \end{bmatrix}$. Take for the similarity measure (s in (4.143)) the function

$$s(DP(\mathbf{z}), DP^1) = 1 - \left(\frac{\left\| DP(\mathbf{z}) - DT^1 \right\|_1}{Lc} \right) ,$$

where the arguments of s are regarded as vectors in $\Re^{Lc}$. Then calculate

$$s(DP(\mathbf{z}), DT^1) = 1 - \left(\frac{0.3 + 0.2 + 0.1 + 0.5}{4} \right) = 0.725 \text{ ; and}$$

$$s(DP(\mathbf{z}), DT^2) = 1 - \left(\frac{0.1 + 0.1 + 0.2 + 0.0}{4} \right) = 0.9 \quad .$$

Both decision templates produce the correct label when the output of (4.143) is hardened, so it is possible to do better than the oracle classifier - possible, but not often likely.

At the other extreme is the *single best* SB) result, which means that the L classifiers are run independently (without fusion), and the best performance of any one of them is recorded as SB. Thus, SB is *not* a fused classifier, and is (almost always) an empirical lower bound on the performance of fusion schemes, which presumably - but again, this is not guaranteed by any theory - improves the performance of the single best one (after all, this is really the only reason to consider second level classifier fusion at all). The gap between the single best and oracle classifiers might be taken as the "potential for improvement" that can be realized by any fusion scheme.

Now we are ready to present Example 4.31, which compares 25 fusion models, including 11 decision template models, and one non-fused classifier, to each other. Some of the results in Example 4.31

come from Kuncheva et al. (1998), and some are published here for the first time.

Example 4.31 We briefly describe the two data sets used in Kuncheva et al. (1998). Satimage is a set of $n = 6{,}534$ vectors in $p = 36$ dimensions with $c = 6$ crisp labels derived from multispectral LANDSAT images. The labels and their percentage of samples in the data are: red soil (23.82%), cotton (10.92%), gray soil (23.43%), damp gray soil (9.73%), soil with vegetation (10.99%) and very damp gray soil (23.43%). Kuncheva et al. (1998, 1999) used four features from this set, #s 17-20, as recommended by the database designers. Phoneme is a set of $n = 5{,}404$ vectors in $p = 5$ dimensions with $c = 2$ crisp labels. The two classes and their percentage of the total sample were: nasals (70.65%) and orals (29.35%). In what follows we call these two source data sets *Satimage* and *Phoneme*.

All 2D subsets that can be made by selecting *pairs of features* from the 4D and 5D source data were extracted from Satimage and Phoneme. The Satimage data with $p = 4$ features yields 6 pairs of 2D features, so L for the Satimage experiments using 2D subsets is 6. Similarly, using all 2D subsets of the 5D phoneme data yields $L = 10$ for the Phoneme experiments. Each of these *sets of* 2D data were the basis for the design of a corresponding first level quadratic discriminant classifier (6 first level classifiers for Satimage, 10 first level classifiers for Phoneme). We can assert with some assurance that the (two sets of) classifiers for these experiments were *not* independent because each feature from the original data sets participated in the training of p-1 of them.

We discuss training and testing for the Satimage data, where $L = 6$ yields 6 2D training and test sets, say $\{X_{tr,1}, X_{te,1}\},\ldots,\{X_{tr,6}, X_{te,6}\}$. Then, 6 first level QDC classifiers $\mathbf{D}_1,\ldots,\mathbf{D}_6$ are trained with only the corresponding 2D data sets. That is, $\mathbf{D}_k$ is trained with $X_{tr,k}$, and during testing, only samples from $X_{te,k}$ are submitted to $\mathbf{D}_k$, $k = 1,\ldots,6$. Thus, row 1 of DP($\mathbf{z}$) is $\mathbf{D}_1(\mathbf{z})$ with $\mathbf{z} \in \Re^2$ having two of the four features, row 2 of DP($\mathbf{z}$) is $\mathbf{D}_2(\mathbf{z})$ with $\mathbf{z} \in \Re^2$ having a different pair of the four features, and so on. The decision template for the kth class, $DT^k(X_{tr,k})$, is built according to (4.139) with the 2D data set $X_{tr,k}$, $k = 1,\ldots,6$. Table 4.59 lists the results of the experiments. Training and testing for the Phoneme data was done similarly, but with $L = 10$ first level classifiers. Each of the two data sets was used to make 40 training and testing runs in 4 groups of 10 run averages as follows. First, 100 training data are extracted randomly from either Satimage or Phoneme.

Table 4.59 Cumulative rankings for 25 fusion schemes and the single best one stage classifier

Satimage Data : L = 6		Phoneme Data : L = 10		Sum of ranks: both data sets		Relative efficiency
$\hat{\mathbf{D}}$	$\Sigma\rho(\hat{\mathbf{D}})$	$\hat{\mathbf{D}}$	$\Sigma\rho(\hat{\mathbf{D}})$	$\hat{\mathbf{D}}$	$\Sigma\rho(\hat{\mathbf{D}})$	rel. eff. $(\hat{\mathbf{D}})$
OR	104	OR	104	OR	208	1.00
DS	82	dt:I1	78	dt:I1	159	0.76
MIN	86	dt:s1	78	dt:I2	159	0.76
dt:np	90	dt:s3	85	dt:s1	159	0.76
PRO	91	LOG	87	dt:s2	159	0.76
dt:I1	81	dt:s2	78	PRO	158	0.75
dt:s1	81	dt:I2	78	dt:np	158	0.75
dt:s2	81	DS	72	DS	154	0.73
dt:I2	81	MAJ	71	dt:s3	147	0.70
dt:I4	77	AVR	71	MIN	138	0.65
PPR	64	dt:np	68	AVR	131	0.62
dt:s3	62	PRO	67	dt:I4	126	0.59
AVR	60	BKS	63	MAJ	109	0.51
MAX	48	FI	61	FI	108	0.50
FI	47	MIN	52	LOG	102	0.47
NB	40	MAX	52	MAX	100	0.46
MAJ	38	SB	42	BKS	88	0.40
dt:I5	38	dt:I4	49	PPR	88	0.40
dt:I3	37	NB	31	NB	71	0.32
SB	26	dt:I3	30	SB	68	0.30
BKS	25	PPR	24	dt:I3	67	0.30
dt:c	25	dt:c	23	dt:I5	51	0.22
LOG	15	dt:s4	15	dt:c	48	0.20
LCD	11	dt:I5	13	dt:s4	23	0.08
dt:s4	8	QDC	7	LCD	16	0.04
QDC	6	LCD	5	QDC	13	0.03

After training the first level classifiers (6 QDC classifiers for Satimage, 10 for Phoneme) and, for fusion schemes that are trainable, the second level fusion models, testing is done with the remaining data. The 100 points are replaced, a new draw is made, and training and testing are repeated. This cycle is repeated ten times, and the error rates of the 25 fusion and SB one-stage schemes are averaged, independently - that is, 10 runs of the oracle are averaged, 10 runs of DS are averaged, etc. Then the performance of the 25 fusion and single best 1- stage schemes are ranked 1-26, with 26 denoting the best (highest score), and 1 the worst (lowest score). In this scheme the oracle is expected to rank first (score 26 each time), and the single best (which is not a fusion classifier) last (score 1 each time). Once the 10 runs with $|X_{tr}| = 100$ were completed, another 10 runs were made with $|X_{tr}| = 200$, then another 10 with $|X_{tr}| = 1000$, and finally 10 with $|X_{tr}| = 2000$. Thus, at the end of the experiment, we have 4 ranks for each of the 25 fusion and 1 non-fusion models for each of two data sets. Consequently, the minimum possible *cumulative rank* ($\Sigma\rho(\hat{\mathbf{D}})$ in Table 4.59) over 8 averages of 10 runs is 8=4(1)+4(1), and the maximum is 208 = 4(26)+4(26). Notice that QDC appears in Table 4.59 as a fusion model. These three entries in Table 4.59 correspond to using L QDC's in the first level, and one additional QDC as the second level (fusion) classifier.

The last column in Table 4.59 shows the "efficiency" of each of the fusion schemes relative to the efficiency of the oracle, which is taken to be 1. The relative efficiency is computed by linearly transforming the ranks with the formula eff = (0.005*rank)-0.04, which lie in the interval [8, 208] into the unit interval [0, 1]. The maximum is attained as expected, but the minimum is not attained because different fusion models attained the minimum rank on various runs during the experiment. From the last column of Table 4.59 we see that just below the oracle, 4 of the 11 decision template fusion schemes are tied at a relative efficiency of 0.76, followed closely by PRO (aggregation by the simple product). The Dempster-Shafer (DS) model is very close to this, scoring a relative efficiency of 0.73. The fuzzy integral (FI) used in this particular example has a relative efficiency of 0.50. Discounting the oracle, *this* FI is exactly at the median relative efficiency of the 25 classifier fusion schemes, with 12 schemes above it, and 12 below.

We hasten to point out that there is exactly one way to train decision templates for a given choice of s (in some sense this corresponds to the flexibility afforded other methods by various training schemes). There are dozens of papers that address the question of how best to train fuzzy integrals (i.e., how to generate fuzzy measures, see Section 4.5) ; and, historically, perhaps even more that consider methods for training DS models (i.e., how to get the bpa's, see Section 5.7.A). Just as our summary of the comparison between ssfcm and ssFCM in Section 2.2.B pointed out that *some* choice of supervising points would probably enable ssfcm-AO to produce the same partition that ssFCM did in Example 2.3, we think that using different training schemes for, say, several FI or DS models would produce ranks in Table 4.59 that would be spread out across the table just like the decision templates do in this example.

In this example the fuzzy integral was trained using the method given by Tahani and Keller (1990), while the Dempster-Shafer model was trained with the approach outlined in Rogova (1994). This emphasizes again an important point that we have made before: most methods (here, fusion methods) will probably produce fairly similar results if you have enough time to look for the solution you want. In fact, and very analogous to fuzzy rule-based systems "hiding" as fuzzy decision trees, some of the decision templates in Table 4.59 can be realized as Choquet integrals : MAX, MIN, AVR, in addition to all other linear combinations of order statistics (Grabisch et al., 1999).

There are also 4 relatively inefficient decision template schemes very close to the bottom of the last column in Table 4.59 - even lower than the single best classifier, and only the LDC and QDC are below these four. This probably indicates that the choice of the similarity measure used to match the decision profile to the decision templates is very important. We cannot say from these results which of the four subtypes mentioned above is better or worse on the basis of this limited set of experiments, especially since there are similarity and inclusion indices in both the top 5 decision templates and the bottom 4 decision templates. However, Kuncheva et al. (1998, 1999) distinguish between two types of similarity measures: *integral measures* that are based on cardinality (s1, s2, s3, I1, I2, I3); and *pointwise measures*, which allow a single degree of membership to determine their value (s4, I4, I5, C). The fact that 5 of the 6 integral measures in Table 4.59 place well above I4, the most highly ranked pointwise measure, strongly suggests that integral measures are more reliable than pointwise ones. The dt:np scheme ranks well in Table 4.59, being tied with the PRO scheme for third place in terms of rank or relative efficiency, and once again illustrates that old adage - nearest prototype classifiers are simple, effective and cool. Further, we find it interesting that the simple PRO scheme, which is class conscious, but which has no second level training, does so well.

To conclude, here are some differences between the decision templates approach to classifier fusion and most of the other separately trained fusion schemes we know of. First, when the i-th decision template DT^i is matched against the decision profile of an input, the comparison is class indifferent (uses all of the columns in the decision profile matrix), in contrast to many other popular fusion schemes which are class conscious (use only the appropriate column of the decision profile matrix for each class).

Second, decision templates are non-parametric in the sense that there are no parameters to learn or thresholds to choose. The only choices that must be made to implement them are the choice of F = s, the fusion operator that matches the decision profile matrix DP($\mathbf{z}$) to the c matrices $\{DT^i\}$, and possibly, the method of hardening soft label vectors whenever the need for crisp outputs arises. The choice of a good F is certainly not trivial. As the results in Table 4.59 show, decision templates with some F's worked fine, and with other F's, the same fusion model applied to the same first level classifier outputs were relatively terrible. Since there are infinitely many F's, as always in pattern recognition, finding the right one is what makes the method successful for a particular data set - and you know that don't come easy! On the other hand, once the function s is chosen in equation (4.143), training the decision templates amounts to performing the non-iterative calculations in (4.139), so from the viewpoint of ease of training, the decision template approach is much simpler than most of the other trainable second level fusion models we know about.

Example 4.31 is abstracted from the most ambitious comparison of different fusion models that we know of, accounting for 25 different fusion $\hat{\mathbf{D}}$'s that combine L = 6 (Satimage data) and L =10 (Phoneme data) first level classifiers. Perhaps the most important thing to be gained from this example is the insight it gives into some subtle differences between the various strategies that can be used for separately trained fusion classifiers. There are many, many other approaches to classifier fusion besides the few that we have discussed in this section. We will briefly discuss a few others in Section 4.11.

4.10 Syntactic pattern recognition

Syntactic (or structural) pattern recognition has far fewer advocates in the engineering community than numerical pattern recognition. We aren't sure we can supply a reason for this, but offer several possibilities. First, many syntactic approaches are couched in the highly specialized jargon of formal language theory, which is less accessible to many engineers than it (perhaps) should be. A much more probable reason for the paucity of fuzzy models in this field is that, quite simply, it is usually much harder to build a working, fielded system based on syntactic models than numerical ones. Good basic texts on syntactic pattern recognition include Gonzalez and Thomason (1978), Pavlidis (1980) and Fu (1982). Bunke (1992) is a collection of recent papers on advances in structural pattern recognition.

The *primitive* is the basic element or building block in the syntactic approach; it plays roughly the same role here that numerical features play in the methods we have discussed prior to this section. Another key concept in structural pattern recognition is the *grammar* that can be built from primitives. The grammar can be thought of as a "basis" for a set of objects. Just as $N_{hc} = \{\mathbf{e}_1, \ldots, \mathbf{e}_c\}$ spans or generates every vector in $\Re^c$ as a unique linear combination of its c elements, we can imagine the "span" of a grammar as all of the objects that can be generated from the primitives, using specified rules of combination called *production rules*. The span of N_{hc} is the vector space $\Re^c$; the analogous concept in the syntactic approach is called the *language* generated by the grammar. We will formalize each of these concepts in Section 4.10.A.

The basic idea is to decompose objects, usually hierarchically, into simpler structures, until the source object can be described in terms of its primitives ("basis elements") and the structure that relates them to each other. The paragraph you are reading right now is a good example. If we regard the paragraph as the object, we can decompose it into successively finer and finer elements- viz., sentences, then words, then characters, and finally, "strokes", which are the primitives in the language being illustrated (written English). Notice that the language provides us with two capabilities: first, if we have its building blocks, we can generate objects from it; and conversely, once we have the language, we can test and classify given objects as belonging to it or not. In other words, we have the same problems - *representation* and *recognition* of objects that underlie almost all pattern recognition problems.

Usually primitives are physically recognizable elements of the object. Because sensors don't provide data that is readily converted to the form needed by syntactic methods, there is at least one extra, very hard step - the extraction of sets of primitives from numerical data - if the syntactic approach is to be based on sensor data. Syntactic representation of objects by a set of concatenated or otherwise connected primitives requires a level of sophistication that implies more effort than the results often justify. Coffee cups are simple enough to describe structurally, but a power plant, for example, is not so easy to decompose into a manageable set of primitives. Notable exceptions to this are applications in the fields of shape analysis and character recognition, because data from these two applications lend themselves well to simple sets of primitives which effectively capture structural properties of the objects of interest.

Numerical and structural techniques are often discussed as if they are separate disciplines, but numerical pattern recognition techniques are often very much in evidence in syntactic models, where they can be used to extract structural primitives from sensor data. For example, we might run any of the edge detectors discussed in Chapter 5 on an input image, and then use edge fragments, or groups of them, as segments of objects. Segmenting inputs refers to the decomposition of objects into meaningful substructures. This helps the designer recognize useful primitives from which to build sentences.

Just as numerical pattern recognition can be divided into several main branches (e.g., statistical, graph-theoretic, fuzzy, etc.), structural pattern recognition can be subdivided into two principal approaches by the way the structure that connects primitives together is represented. One group of methods depends on formal language theory for quantification of structural relationships, while the second group of methods use graphs (or relational data) to carry the structural information that links primitives together to build objects. We will explore some basic ideas of this second kind in Section 5.8, where we discuss spatial relations between objects such as "above" and "right of" in connection with the decomposition of complex objects as a means of scene description and object recognition with image data.

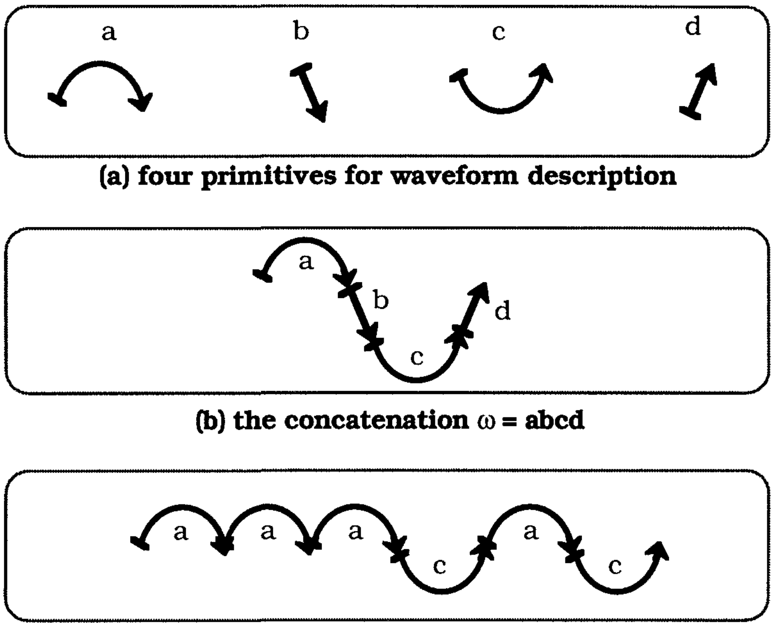
As in numerical pattern recognition, successful syntactic designs are an iterative and interactive procedure : trying new features roughly corresponds to defining new primitives, grammars, languages, etc.; and changing classifiers corresponds to using different automata, matching procedures, and so on. Process description again resides mainly in the hands of human designers, who rely not only on sensor data (such as images of handwritten characters), but on their innate ability to describe structural relationships between elementary parts of complex objects. It is beyond the intended scope of this book as well as our personal interests and backgrounds to present a detailed discussion of either the language or graph-theoretic approaches to structural pattern recognition. However, there has been a steady trickle of papers that use fuzzy models in both of these arenas since the early 1970's. Our objective in this section is to present you with enough information about this approach to pattern recognition so you can decide for yourself whether a plunge into the stream (the literature we cite) is worthwhile for the problems you want to solve.

A. Language-based methods

We ease into this topic with an intuitively reassuring example that displays some of the elements of the formal language approach. Wave form recognition is an application that is amenable to the syntactic approach because one-dimensional signals, like characters in alphabets, can be thought of as built from a sequence of fairly simple "strokes". Thus, we have a conceptual basis for decomposing signals into linked sets of strokes that approximate wave shapes. *String grammars* are appropriate for wave form decomposition, where the most fundamental operation is head to tail connection (or one-dimensional *concatenation*) of the chosen primitives.

Example 4.32 This example has its roots in the pioneering work of Shaw (1969), who proposed a more extensive set of primitives and production rules as the basis of a picture description language that was sufficiently rich to build grammars whose objects included blocks, "houses", and the 26 upper case characters A-Z used by the English language.

Choose as primitives the four arcs shown in Figure 4.96(a); notice that each one is directed in the sense that it has a tail and head. We allow only head to tail connections from left to right in this example - that is, there is but one rule for concatenation of primitives, and we indicate left to right concatenation by left to right juxtaposition. The primitives are the geometric entities, and in order to efficiently describe manipulations of them, we choose an alphabet of symbols, say A = {a, b, c, d} that correspond to the strokes shown in Figure 4.96(a): a = local max (Cap), b = negative slope, c = local min (Cup), and d = positive slope. With suitable production rules that tell us how we may combine the symbols in the alphabet, these primitives can be used to represent many regular wave shapes.



(a) four primitives for waveform description

(b) the concatenation ω = abcd

(c) the concatenation ζ = aaacac, $|\zeta| = 6$

Figure 4.96 Four primitives and waveshapes built from them

Figure 4.96(b) is a nice approximation to one cycle of a sine wave build by concatenating the four primitives into the sentence (or string) ω = abcd. The *length of the string* ω, denoted by $|\omega|$, is the number of symbols from the alphabet needed to make it, so $|\omega| = 4$ in Figure 4.96(b). Figure 4.96(c) shows a slightly more complicated waveshape (which uses fewer primitives but more production rules) that is represented by the sentence ζ=aaacac, $|\zeta| = 6$. These simple figures show you immediately why we might expect syntactic pattern recognition methods to be useful for signal processing.

A second application where the syntactic approach often surfaces is character recognition, both handwritten and printed, and the harder problems that it leads to such as word, sentence and paragraph recognition. Why? Because the fundamental object (the character), can be approximated pretty well by combinations of just a few primitives, which are again "strokes". We have met this idea already in the work of Siy and Chen (1974), Chi et al. (1996b) and Chi and Yan (1996). Siy and Chen held that all alphanumeric characters comprised essentially three "strokes": lines, curves, or circle and they suggested the set of 15 primitives shown in Figure 4.92, made from one of their three basic strokes. Siy and Chen did not use the language or methods that we now associate with syntactic pattern recognition in their work, but the most important elements of their work can be identified with various facets of the structural approach. Their "branch features" are primitives, their representation of each character as a set of three bit maps constitutes the selection of an alphabet and production rules, and their classifier, essentially a nearest prototype machine, is roughly analogous to parsing strings to assign each input to the best matching class.

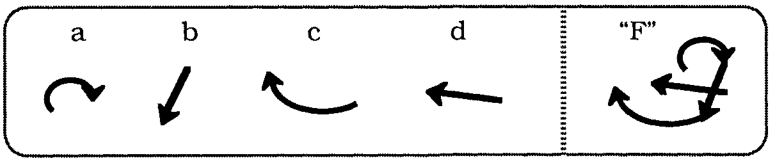


Figure 4.97 Four primitives for the letter "F" in Figure 4.64

Primitives for character recognition will be very similar to the ones shown in Figure 4.92 in that every character in any alphabet will be composed of a fairly small number of simple "strokes". For example, the letter "F" in Figure 4.64 is (roughly) composed of four strokes, as shown in Figure 4.97. This construction is a little more complicated than the waveforms in Figure 4.96, since the letter F cannot be built from a single concatenation of primitives. The "body" of the F is composed of the string abc, and could be built by head to tail concatenation, but the crossing arm of this character, represented by the primitive d, cannot be adjoined correctly to abc by head to tail attachment. Here the location of the crossing arm must be fixed by an operation that is not available with simple head to tail concatenation - in other words, we need a more extensive set of production rules in order to build a good approximation to "F" with these primitives.

Now we are ready to make a slightly more formal presentation of the ideas in Example 4.32. A *grammar* G is defined as a 4-tuple G = (V_T, V_N, P, S) where

$\boxed{\text{G1}}$ V_T is a finite set of *terminals* or primitive variables, such as the four waveshape fragments "cup", "cap", "positive line" and "negative line" in Example 4.32. When each primitive $v \in V_T$ is represented by a symbol, the set of symbols that represent V_T is called a vocabulary A. In Example 4.32 A= {a, b, c, d}, and once a vocabulary is chosen, it is customary to interchange V_T and A. Thus, we might also say that V_T = {a, b, c, d} in Example 4.32. We follow traditional notation for terminals, symbolizing them with lower case letters such as a, b, etc.

$\boxed{\text{G2}}$ V_N is a finite set of *nonterminals* (also called constants or nonterminal variables or subpatterns) that are used in intermediate stages during the construction of sentences. We did not specify any non-terminals in Example 4.32. We follow traditional notation for non-terminals, symbolizing them with upper case letters such as A, B, etc.

$\boxed{\text{G3}}$ P is a finite set of syntax or *production rules* (or rewriting rules) that are used in the generation and/or parsing of sentences. Strings in G are traditionally symbolized by lower case Greek letters like α and β, and the production rule that maps string α to string β is written as $\alpha \to \beta$. We did not specify any production rules in Example 4.32.

$\boxed{\text{G4}}$ $S \in V_N$ is a special member of V_N, called the *starting (or root) symbol* of a sentence. This non-terminal is used as the beginning point of sentences that can be generated by the grammar. We did not specify a starting symbol in Example 4.32.

We have used the traditional notation of syntactic pattern recognition for the terminals (V_T) and non-terminals (V_N) in a grammar G, which are sometimes called the *vocabularies* of G. The set $V = V_T \cup V_N$, with $V_T \cap V_N = \emptyset$ is called the *alphabet* or *total vocabulary* of G. We caution you that other authors use different terminology for many of the ideas given in the definitions below (e.g., for meanings of the words alphabet, vocabulary, etc.).

When α is a string, $\alpha^n = \underbrace{\alpha \cdots \alpha}_{\text{n times}}$. The null string is denoted by λ. The set of all finite length strings of symbols, including the null string λ, that can be built from a finite alphabet V is denoted by V^*; in particular, V_T^* is the set of all finite length strings of terminals in G. If P contains the rewrite rule $\alpha \to \beta$, $\eta = \omega_1\alpha\omega_2$ and $\gamma = \omega_1\beta\omega_2$, we say that string η directly generates a string γ, and write $\eta \underset{G}{\Rightarrow} \gamma$.

There are many types of primitives, sentence structures, grammars (strings, trees, webs, plexes, etc.) and production rules. Of these we discuss and illustrate only the simplest (and possibly most frequently encountered in pattern recognition) grammar - the string grammar. A simple string grammar to represent "sine waves" using the four primitives shown in Example 4.32 could be $G_{abcd} = (V_T, V_N, S, P)$ where

$$V_T = \{a, b, c, d\}; \qquad V_N = \{S\}; \qquad P = \{\, S \to abcdS,\ S \to abcd\}.$$

The set of all possible sentences L(G) which can be generated by a grammar G is called a *language* over G,

$$L(G) = \{\alpha : \alpha \in V_T^*;\ S \underset{G}{\Rightarrow} \alpha\} \quad . \tag{4.147}$$

Grammars are used two ways. In the *generation mode*, we use G to create sentences beginning with S, and ending with the string; when represented by a tree, this mode of operation results in a *derivation tree* - we derive the string from G using the symbols in V and the rules in P. Since we always begin a derivation from S, derivation trees are always generated top-down, ending up with the sentence at the leaves of the tree.

Conversely, if we are given a sentence which might be a member of L(G), finding out whether it is or is not a member of L(G) is called the analytic (parsing) mode - this is the mode we will be in during classifier operation. When represented by a tree, this mode of operation results in a *parse tree,* which can be either top down, or bottom up. You can imagine this process as beginning with an "unfilled" rooted tree: you know S and the leaves, and try to fill in the interior of the tree with valid productions. The issues that demand attention during parsing are computational efficiency and termination of the parse, and there are many schemes available to accomplish this step. As grammars become more complex, so do these two problems. A conflict occurs if a sentence parses in more than one language (or tree), which is highly possible when general grammars are constructed to represent actual shapes.

For example, we cannot generate the waveshape in Figure 4.96(c) with G_{abcd}, because the production rules needed to generate the string ζ = aaacac are not in P. Consider the grammar G_{db} with terminals as in Example 4.32 and non-terminals and production rules as follows:

$$V_T = \{a, b, c, d\};$$
$$V_N = \{S, A, B\};$$
$$P_{db} = \{\, S \to dA,\ A \to dA,\ A \to d,\ A \to dB,\ B \to bB,\ B \to b\,\}.$$

This set of production rules cannot generate the string ζ = aaacac either, because the only terminals that are manipulated by the production rules in P_{db} are d and b, even though a and c are in V_T. The grammar G_{db} generates strings of the type $\{\chi = d^n b^m \mid n > 1, m \geq 0\}$, which are piecewise linear curves that start with n positively sloped line segments followed by m negatively sloped line segments - that is "$\wedge$"-shaped objects. Figure 4.98 shows a few of the sentences that can be built with this grammar. If it was necessary to balance the number of b's and d's, different production rules would be needed. Would the grammar G_{ac} built by substituting a for d and c for b in the production rule set P_{db} generate the string ζ = aaacac? No, because the grammars G_{ac} and G_{bd} use the same terminals but have different productions, and hence, generate different languages. This is a key aspect of syntactic pattern recognition - finding production rules that will generate the sentences (objects) we wish to represent and recognize.

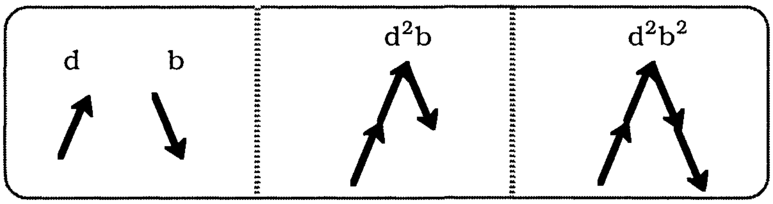


Figure 4.98 Some sentences in the grammar G_{db}

A grammar and language that can capture all the objects you want to recognize must be inferred somehow. Once L(G) is in hand, classification is often done with the aid of formal language theory (Chomsky, 1965). A grammar is formed for each class, and sentences are recognized as belonging to a given class if they can be parsed in the grammar for that class. If several classes are possible, all of the class grammars usually share the same primitive elements so that all grammars have an opportunity to generate the sentence. Recognition (of, e.g., a piece of an object) then reduces to parsing a sentence to see if it can be generated by the grammar for that object class. Starting with the symbol S, we look for a sequence of production rules which produce the sentence. For example, in the grammar G_{db}, the sentence ξ=dddbb would parse as

$$S \to dA \to ddA \to dddB \to dddbB \to dddbb.$$

Many syntactic pattern recognition applications deal with shape recognition. The first task is to convert real shape data into the string or tree or plex that formally represents each shape. In Example 4.32, the primitives are perfect semicircles and diagonal lines. If you were to draw a curve like either of the ones shown in Figures 4.96(b) or 4.96(c), you would be unlikely to make perfect matches to the 4 primitive shapes. This problem is compounded by automatic boundary extraction methods which are often used to "outline" an object (Chapter 5). Hence, before the shape can be analyzed via syntactic grammars, it must be encoded as a sentence in the appropriate languages. Mistakes during primitive classification can doom the parsing algorithms that follow. This is an instance where the principle of least commitment plays a major role.

Where do fuzzy sets, models and methods fit into syntactic pattern recognition? They have been inserted into the primitive extraction phase, into the collection of data from uncertain sources, into the production rules, and into the parsing activities carried out during classification. Fuzzy sets made early appearances in both the fuzzification of primitives and the construction of fuzzy grammars and fuzzy languages. Indeed, we have already seen how the 15 primitives in Figure 4.92 can be fuzzified (equation (4.133) and 10 others like it were used in Siy and Chen (1974), and are still in use by, for example, Chi et al. (1996b) and Chi and Yan (1996)).

Fuzzy grammars were defined very early in the evolution of fuzzy models, first appearing in the paper by Lee and Zadeh (1969). Formally, a *fuzzy grammar* is a 5-tuple $G_f = (V_T, V_N, S, P, m)$ where V_T, V_N, S, and P are as before, and m: $P \to [0,1]$ represents the grade of membership of each production rule in the grammar, i.e., some rules are more typical of the grammar than others, and so, they will have higher memberships. For a rule $a \to b$ in P, we can write

$$m(a \to b) = \rho\text{, or more simply, } a \xrightarrow{\rho} b \quad . \tag{4.148}$$

If $a \xrightarrow{\rho} b$ and γ and δ are arbitrary strings, then

$$\gamma a \delta \xrightarrow{\rho} \gamma b \delta \quad . \tag{4.149}$$

The *membership of a string* x in $L(G_f)$ is defined as

$$m(x) = m(S \to x) = \sup\{\min\{m(S \to \alpha_1), \ldots, m(\alpha_n \to x)\}\}, \tag{4.150}$$

where the supremum is taken over all derivation chains such as the chain $\alpha_1, \ldots, \alpha_n$ from S to x. Instead of the min (T_3 norm) in (4.150), any T-norm could be used. This is similar to the way in which stochastic grammars are defined, except that the interpretation of uncertainty in stochastic grammars is that of likelihood of use of a production rule, and the method for determining the probability of a string x in $L(G_f)$ is usually the sum over all derivations of the product of the rule probabilities. Tamura and Tanaka (1973) developed several early techniques for learning fuzzy grammars.

Example 4.33 In the positive slope or "∧" grammar G_{db}, suppose that the production rules are modified to include memberships:

$$P = \{S \xrightarrow{1.0} dA, A \xrightarrow{0.8} dA, A \xrightarrow{0.8} d, A \xrightarrow{0.6} dB, B \xrightarrow{0.6} bB, B \xrightarrow{0.6} b\}.$$

Then the membership of a string x=dddbb in the fuzzy language $L(G_{db,f})$ is m(x) = min {1.0, 0.8, 0.6, 0.6, 0.6} = 0.6 (note that there is only one derivation of this sentence). The string y = ddddd would have membership m(y) = min {1.0, 0.8, 0.8, 0.8, 0.8} = 0.8, i.e., positively sloped lines have higher membership than ∧'s in this fuzzy grammar. If the memberships are thought of as probabilities $G_{db,f}$ will become a stochastic grammar, and then the probability of x would turn out to be $p(x) = 1.0 \cdot 0.8 \cdot 0.6 \cdot 0.6 \cdot 0.6 = 0.173$. Stochastic grammars can have the property that sentences of small length are favored over longer ones. This is due to the fact that product is a much stricter intersection operator than minimum. If the product (T_2 norm) were used in (4.150) instead of the min (T_3 norm), then m(x) and p(x) would coincide.

Fuzzy automata are important in fuzzy syntactic pattern recognition. For example, E. T. Lee (1982) developed an approach to represent approximate shapes by fuzzy tree automata and process the shapes based on syntactic pattern recognition. Lee (1972a, b) had earlier experimented with fuzzification of sets of shape primitives which were similar in geometric content to the primitives being used by Shaw (1972), with fuzzification quite like that used by Siy and Chen (1974). In Lee (1982) a set of three primitives (isosceles triangle, rectangle, and cross) are first fuzzified, and then used as the basis for a fuzzy grammar. For example, Lee (1982) proposed that the membership function for a "fuzzy isosceles" triangle with interior base angles $B°$ and $C°$, in the set of crisp isosceles triangles be

$$m_{IT}(B°, C°) = 1 - \frac{|B° - C°|}{90°} \quad .$$

When a triangle is isosceles, $B° = C°$ so $m_{IT}(B°, C°) = 1$, and otherwise, $m_{IT}(B°, C°) < 1$. Similarly, the membership of a quadrangle with interior angles $A°, B°, C°$ and $D°$ in the set of crisp rectangles was defined as

$$m_R(A°,B°,C°,D°) = 1 - \left(\frac{|A° - 90°| + |B° - 90°| + |C° - 90°| + |D° - 90°|}{360°} \right).$$

Using these fuzzified primitives, Lee (1982) built syntactic representations of "approximate houses" by concatenating "approximate isosceles triangles" on top of "approximate rectangles". His constructions (Figure 1 of Lee, 1982) for three approximate houses named s_1, s_2 and s_3 are replicated in our Figure 4.99.

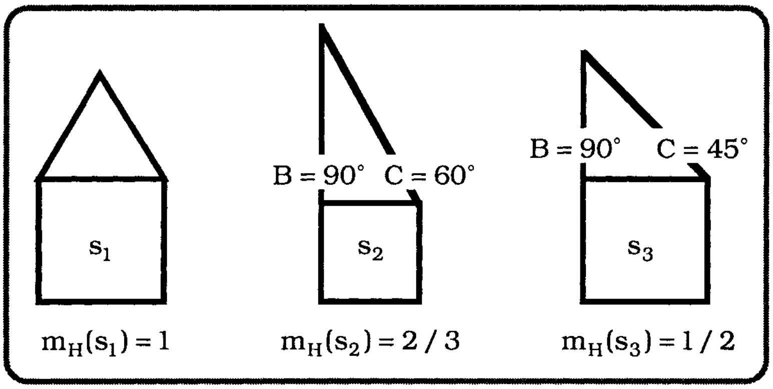


Figure 4.99 Some approximate houses, Lee (1982)

The three membership values shown below the approximate houses in Figure 4.99 are computed in Lee (1982) using a well defined fuzzy tree automaton. Lee also describes how to build "approximate churches" and "houses with high roofs" using fuzzy grammars. We remark that in order to perform a concatenation operation such as "place the roof on top of the house", it is necessary to resolve the question of how to define the spatial relationship "on top of", a topic we discuss in Section 5.8. Another point worth emphasizing here is that Lee's primitives, P={triangle, rectangle, cross}, are themselves decomposable into strings of simpler primitives. For example, the roof of house s_1 in Figure 4.99 might correspond to the string d^2b^2 in the grammar G_{db} discussed in connection with Figure 4.98. This is how we ultimately break down a complicated object in the formal language approach to syntactic pattern recognition.

Now we can cast pattern recognition questions in terms of formal languages. In order to build classifiers capable of identifying different objects using this approach, we follow a procedure that might go roughly like this. Assume that the basic objects are handwritten symbols, and we want to read them automatically with a computer vision system. First we acquire data by digitizing a scene, perhaps run an threshold operator on them, thin them to get their skeletons, and then decompose them into sentences in one or more of the c languages $\{L(G_i)\}$ over the grammars $\{G_i\}$. We will have chosen a set of primitives, and once particular objects are acquired, we will look for the production rules that generate correct representations of the objects in at least one of the chosen (or inferred) grammars. The same set of primitives may support many useful grammars, and finding the primitives, the production rules, and the grammars can all be part of the "training" procedure. Once the languages $\{L(G_i)\}$ that capture the objects of interest are known, we turn the process around during classifier operation. Now a new object comes into the system. We convert it into a sentence using the primitives at our disposal, and then try to parse it in one or more of the grammars available. When we get a one or more matches, we have (a) class label(s) for the object.

You might be surprised to discover how many of the algorithms we have already discussed in this book can be converted into similar (if not the same) techniques for use with strings and/or string grammars. Fu (1982) discusses straightforward extensions to the syntactic case (at least for string grammars) of the following crisp algorithms that we have discussed for either the object or relational data cases: hard c-means (our section 2.2); single linkage clustering via the minimal spanning tree (our section 3.3); the crisp 1-np and 1-nmp rules (our Section 4.2); and the k-nn rule (our Section 4.4).

One of most fundamental ideas underlying most of the algorithms we have discussed prior to this section, including all of the clustering and classifier designs mentioned in the previous paragraph, is the distance $\delta(\mathbf{x}, \mathbf{y})$ between vectors $\mathbf{x}$ and $\mathbf{y}$ in $\Re^p$. There are several equivalent notions for strings. The most common metric used in the setting of strings is the *Levenshtein metric*, which is defined in terms of three string transformations - substitution, deletion and insertion. For strings α, β in V_T, Levenshtein (1966) defined these for any $\omega_1, \omega_2 \in V_T^*$ as follows:

$$\omega_1 a \omega_2 \overset{T_S}{\mapsto} \omega_1 b \omega_2, \ \forall a, b \in V_T; a \neq b \quad \text{(substitution)} \ ; \qquad (4.151a)$$

$$\omega_1 a \omega_2 \overset{T_D}{\mapsto} \omega_1 \omega_2, \ \forall a \in V_T \quad \text{(deletion)} \ . \qquad (4.151b)$$

$$\omega_1 \omega_2 \overset{T_I}{\mapsto} \omega_1 a \omega_2, \ \forall a \in V_T \quad \text{(insertion)} \ . \qquad (4.151c)$$

With the three string transformations in equations (4.151), the *Levenshtein distance* $\delta_{LEV}(\omega_1, \omega_2)$ between two strings $\omega_1, \omega_2 \in V_T^*$ is defined as the smallest number of transformations needed to derive one string from the other (either way, since this distance will be symmetric). For example, if $\kappa_1 = \text{bbdbabc}$ and $\kappa_2 = \text{bdbbabbc}$, we can produce κ_2 from κ_1 with 3 transformations: insert "b" between the symbols a and b; substitute "d" for the second b; and finally, substitute "b" for the (new) second d. Thus, $\delta_{LEV}(\kappa_1, \kappa_2) = 3$. Two things to notice: the sequence of transformations used in the calculation is not unique (but the result is); and the distance between strings of different lengths is well-defined. Fu (1982) gives a weighted form of δ_{LEV} which allows you to weight different types of errors differently.

Having a way to measure distances between strings in different string languages, opens many doors. For example, you can construct a minimal spanning tree on sets of strings. The distance between two languages $L(G_1)$ and $L(G_2)$, or as a special case, between a sentence ω and a language $L(G)$, can be defined directly by using any standard measure of the distance between pairs of sets. For example, any of the set distances shown in Figure 3.3 serve this purpose. Just imagine that the points in Figure 3.3 are sentences, that $\hat{\delta} = \delta_{LEV}$, that $X = L(G_1)$ and that $Y = L(G_2)$.

Now look back at the 1-np, 1-nmp and k-nn rules in equations (4.2), (4.7) and (4.38), respectively. All of these classifiers, built for feature vectors in $\Re^p$, need only prototypes and a way to measure distance. Suppose you are lucky enough to have a set of c string grammars $\{G_i\}$ that generate c string languages $\{L(G_i)\}$, and for each language, you have, say, n_i crisply labeled sentences $\{\alpha_j^i \in V_{T_i}^*; j = 1, \ldots, n_i\}$, with $\sum_{i=1}^{c} n_i = n$. Then the n sentences $\{\alpha_j^i; i = 1, \ldots, c; j = 1, \ldots, n_i\}$ together with the distance δ_{LEV} enable you to use the k-nn rule in (4.38) directly to classify any input string.

To use the 1-np and 1-nmp rules, you need prototypes. Fu (1982) calls the string α_q^i, selected from the n_i strings $\{\alpha_j^i; j = 1, \ldots, n_i\}$ in the i-th class whose indices satisfy

$$q = \underbrace{\arg\min}_{1 \le j \le n_i} \left\{ c_j^i = \sum_{s=1}^{n_i} \left(\frac{\delta_{LEV}(\alpha_j^i, \alpha_s^i)}{n_i} \right) \right\} \quad , \tag{4.152}$$

the "representation" or *cluster center* of the n_i strings $\{\alpha_j^i\}$. Notice that the "cluster center" α_q^i is not *built from* the sentences already labeled class i; it *is* one of the sentences in this class. Once we have a way to find prototypes, the crisp 1-np rule at (4.2) and the crisp 1-nmp rule at (4.7) can be implemented directly in the syntactic domain.

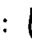
Could you fuzzify any of these designs in the syntactic string grammar domain? Some of them are already done. For example, the soft k-nn rules in Table 4.19 translate directly into soft k-nn rules for sentences in string grammars. The c-means clustering algorithms can all be *imitated* using prototype calculations similar to the one at (4.152) and the necessary conditions for U shown in Table 2.2. Doing this leads to c-means type clustering algorithms that can be used to find clusters in an unlabeled set of n strings which (presumably) come from one of c languages.

Table 4.60 shows how to implement a syntactic relative of the hard c-means (HCM) clustering algorithm, which we will call *string grammar hard c-means* (sgHCM), the prefix "sg" standing for *string grammar*. This is not an alternating optimization algorithm, because the update equations used at each half-iterate do not satisfy any criterion of optimality. A more accurate term is *alternating cluster estimation* (ACE), and it means exactly the same thing here that it does in the numerical case (Runkler and Bezdek, 1998b, 1998c, 1999): update functions for each half of the cycle are simply picked from a set of logical choices, and iteration proceeds to termination, either by small successive changes in the estimates of U or by exceeding a maximum iterate limit. We won't have a very good idea of where this type of clustering leads in terms of the formal languages that underlie the strings it groups together, but it does provide a way to design a *syntactic nearest prototype* (1-snp) classifier.

The sgHCM algorithm in Table 4.60 appears in Fu (1982) as Algorithm 9.4 in a different notation. Scalar multiplication of strings by real numbers is undefined, so the quantities $u_{ij,t}\alpha_j$ and $u_{is,t}\alpha_s$ in sgHCM appear to be incorrect. However, the memberships here are crisp, so the multipliers are either 0's or 1's. If we define $0 \cdot \alpha \equiv \lambda$, the null string, and $1 \cdot \alpha \equiv \alpha$, the notation in Table 4.60 makes sense.

Table 4.60 The sgHCM clustering algorithm

Store	n unlabeled finite strings $X = \{\alpha_k ; k = 1,\ldots,n\}$
Pick	☛ number of clusters: $1 < c < n$ ☛ maximum number of iterations: T ☛ termination measure: $E_t = \lVert \mathbf{V}_t - \mathbf{V}_{t-1} \rVert$ = big value ☛ termination threshold: $0 < \varepsilon$ = small value
Guess	☛ initial string prototypes: $\mathbf{V}_0 = (\hat{\alpha}_{1,0},\ldots,\hat{\alpha}_{c,0}), \hat{\alpha}_{i,0} \in X, i = 1,\ldots,c$
Iterate	$t \leftarrow 0$ REPEAT $t \leftarrow t+1$ For k = 1 to n For i = 1 to c $D_{ik,t} = \delta^2_{LEV}(\alpha_k, \hat{\alpha}_{i,t-1})$ Next i Next k For i = 1 to c $n_{i,t} \leftarrow 0$ For k = 1 to n $u_{ik,t} = \begin{Bmatrix} 1; & D_{ik,t} \le D_{ij,t}, j \ne i \\ 0; & \text{otherwise} \end{Bmatrix}$ $n_{i,t} \leftarrow n_{i,t} + u_{ik,t}$ Next k Next i For i = 1 to c $q = \underbrace{\arg\min}_{1\le j\le n_{i,t}} \left\{ c_{i,j} = \sum_{s=1}^{n} \left(\frac{\delta_{LEV}(u_{ij,t}\alpha_j, u_{is,t}\alpha_s)}{n_{i,t}} \right) \right\}$ $\hat{\alpha}_{i,t} = \alpha_q$ Next i UNTIL (t=T or $E_t \le \varepsilon$) $(U, \mathbf{V}) \leftarrow (U_t, \mathbf{V}_t)$

Fu illustrates sgHCM with a set of 51 unlabeled samples of one of the 9 upper case characters {D, F, H, K, P, U, V, X, Y}. Fu notes that of the nine possible classes, there are four shape-similar pairs, namely (D, P), (H, K), (U, V) and (X, Y), and one "odd" shape, the letter F. Each sample begins as a continuous line pattern on a 20×20 grid. This image is digitized, and then a string representing it is generated by traversing the chain encoding (Gonzalez and Woods, 1992) of the letter cell by cell. A primitive from a set of 4 is then generated from the chain code for each three consecutive cells, and the set of primitives needed to traverse the sample becomes the string for that letter. The four primitives are lines that look roughly like this: a = /; b = \; c = \; d = —. For example, the fourth sample resembles a "U", but is distorted so that it is rather too tall for its width, and has uneven sides, looking roughly like this: . The string representing this sample is cbbb×dabbbb, where the symbol × is one of three concatenation operators taken from Shaw (1962). See Fu (1982) for a more detailed account of how the string representation for a character is generated. The shortest string in the data set had just 5 primitives (a nice, well behaved "Y"), while the longest string had 13 (a not so unruly, but very curvy "D").

Applying the sgHCM algorithm to this data with c = 9 fixed (using prior knowledge as to the number of clusters avoided the question of how many to look for) resulted in 9 crisp clusters of strings (and, therefore, of the objects that the strings represented). The relabeling error (that is, number of mislabeled characters when the data are subsequently labeled by visual inspection) of the hard 9-partition of this data is 11 mistakes in 51 tries - an "error rate" of about 22%. Since the sgHCM algorithm is unsupervised, this may not be such a bad result, and what's more, at this point you have labeled prototypes (which here are labeled strings in some language) of the nine letters, and so, a complete set of parameters to implement the 1-np rule in equation (4.2) using the Levenshtein distance. In a realistic application domain, you may need to be careful about the computational complexity associated with computing δ_{LEV}, which is a combinatorial optimization problem.

What about versions of *string grammar fuzzy and possibilistic c-means* (sgFCM and sgPCM)? We believe that both of these algorithms can be developed to cluster strings, although the generalizations in these cases may not be as straightforward as that of sgHCM in Table 4.60. A modified version of FCM has been used for preprocessing in a syntactic model that uses string grammars for the recognition of handwritten Chinese characters (Cheung and Chan, 1986), but to our knowledge, sgFCM and sgPCM algorithms per sé have yet to be developed. The point is not that string grammar versions of the c-means models are better or worse than any other clustering algorithms for string grammars. The point is that you can often transform pattern recognition methods that are familiar in the numerical data domain into methods in the syntactic domain that bear at least some resemblance to their numerical relatives, and there is as much opportunity to soften models in this domain as there is in the numerical arena. The *theory* underlying syntactic classifier algorithms developed in this mold, however, may challenge the best theoretical computer scientist you know (who is none of us, that's for sure).

B. Relation-based methods

This approach to syntactic pattern recognition uses graphs to represent structural relationships between the primitives and nonterminals. The nodes represent elements of V_T and V_N, and the edges carry relational information about the structure between elements that comprise an object. One of the most important changes that is made by using graphs instead of grammars to represent structural relationships is that the style of object recognition changes. To classify sentences in formal languages, you need a good parser; to classify objects that are represented by graphs, you need measures of graph similarity. Thus, the relational approach often has a style that is very much like nearest prototype classifier design, but the prototypes are digraphs, relational graphs or attributed graphs, and measures of distance used with numerical data are replaced by graph-matching techniques based on measures of graph similarity. Advocates of the relational approach argue that it should be used when each structure can be represented by a crisp prototype, or when there are not enough training data to accurately infer useful grammars for each of the c classes.

Like all graph theory, the use of relational graphs for syntactic pattern recognition is 9 parts definitional to 1 part operational. We are not going to give you sufficient technical information that enable you to build syntactic pattern classifiers using the relational approach directly. Again, as in the previous subsection, we want instead to show you how fuzzy models have been inserted into this field. When you are interested, you will again have to dive into the cited literature. There you will find the details (and in them, perhaps the devil as well).

The basic structure of the relational model begins with the idea of a *semantic net* (sometimes called a *relational graph)*. We start with a digraph G = (V, E) where $V=\{v_j\}$ are the vertices of G ; and E is the set of edges in G, $(v_i, v_j) \in E \Leftrightarrow r_{ij} = 1$ (don't confuse this G with the G we used from grammars in the previous subsection). Terminal nodes in V "contain" primitives of strings, and non-terminal nodes in V will contain intermediate strings that are non-terminal strings. When we add semantics to the edges in E, we obtain a semantic net.

For example, consider the top part of Figure 4.100, which shows three boxes, X, Y and Z, on a supporting surface which is not part of the structure being described. There are six obvious relations between the three boxes X, Y and Z : is above, is below, left of, right of, larger than and smaller than. The bottom half of Figure 4.100 shows the relationships (position and relative size) between these three objects as a semantic net, which, as you can see, is a digraph on three nodes with semantic information added to the directed edges.

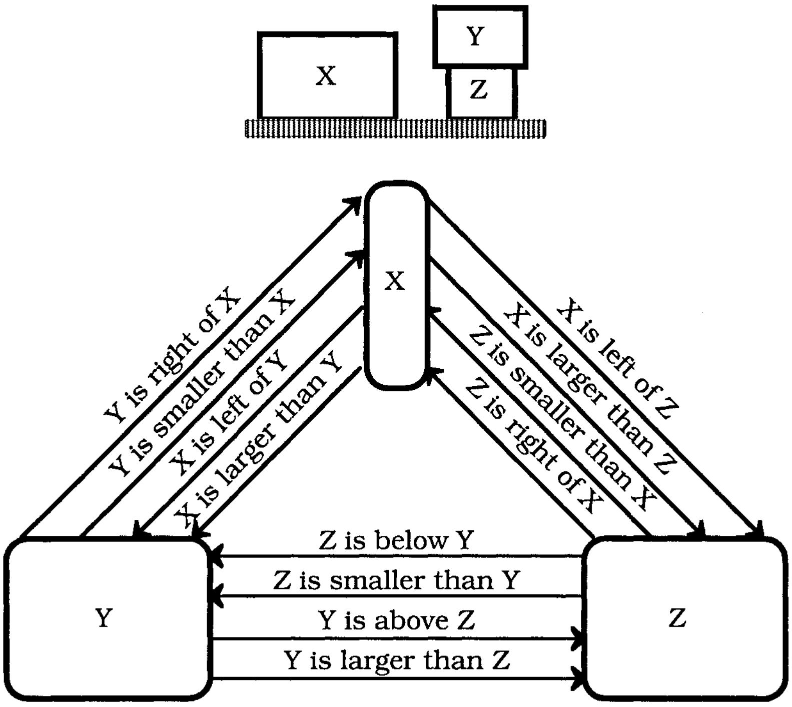


Figure 4.100 Structural representation by a semantic net

If you wanted to build a classifier based on this approach, first you would develop one or more relational graphs for each class of objects (in essence, multiple prototype graphs), and then to label an input object, you would represent it the same way, and match it to each of the class prototypes, using a rule just like equation (4.2) or (4.7) with the appropriate changes in **V** and δ. Figure 4.101 illustrates this idea.

The only relation represented by the graphs in Figure 4.101 is the structural relation "is on top of T". The objects on the left and right in this figure, two sets of stacked blocks, represent crisp classes 1 and 2, and the object z between them is to be classified. First, all three objects are represented by the relational graphs shown just below the objects. Then a measure of graph similarity, shown as S in Figure 4.101, is used to compare the similarity of each of the graph prototypes to the unlabeled input graph. There are many measures of similarity for pairs of graphs based on concepts such as counts of in-degrees, out-degrees, numbers of nodes and/or edges, differences in the minimal spanning tree, etc. For example, using either the counts of outdegrees or numbers of nodes for the graphs in Figure 4.101, we find that G_2 is more similar to G_z than G_1 is (in fact, G_2 and

G_z match exactly in these two measures), so z is labeled class 2 as depicted in Figure 4.101.

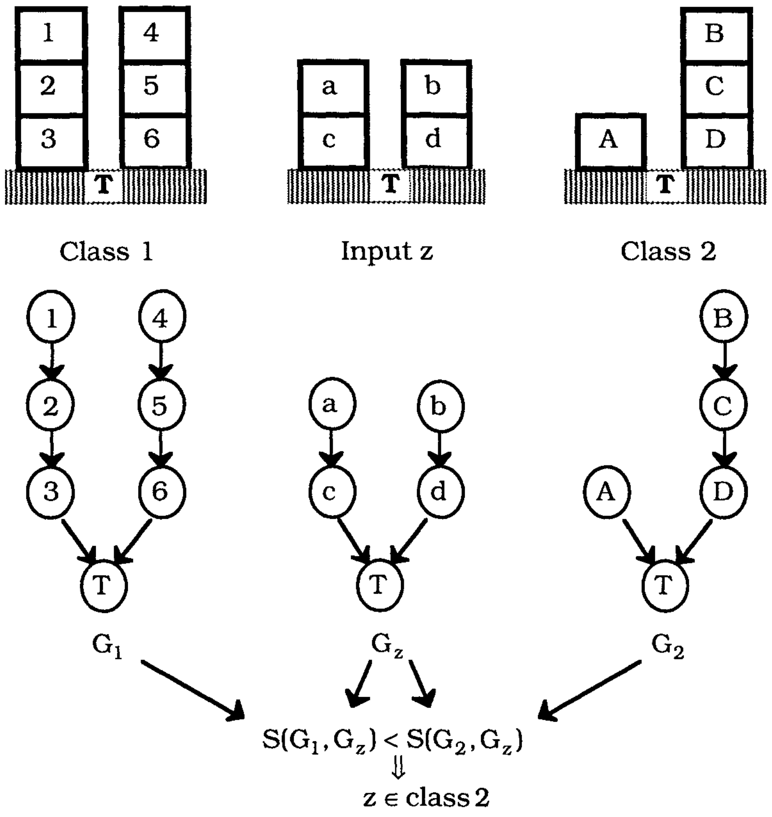


Figure 4.101 Prototype classification with relational graphs

Please compare Figure 4.101 to Figure 4.5; from this you will see that we are again doing nearest prototype classification, but the data used to implement the 1-np design originate through structural relationships instead of numerical measurements. Since the fuzzy models we want to discuss develop their own (fuzzy) measures of similarity, we will not stop here to discuss crisp measures, but instead, will refer you to Shapiro and Haralick (1985) for a representative discussion of this topic.

Relational graphs are limited by several things. First, it can be computationally expensive to match them. Second, they concentrate entirely on structural properties. As defined, relational graphs cannot represent possibly important and measurable quantitative and qualitative properties such as weight, length, color, and so on. One approach to enriching the representational structure of semantic nets is the *attributed graph* (Tsai and Fu, 1979), which enables you to include both numerical and symbolic attributes of primitives at nodes in the semantic net. Thus, object representation becomes a combination of structural, numerical and symbolic attributes. An attributed graph is a graph in which both the node set V and the edge set E can have attributes associated with them. In general then, each node $v \in V$ can take attributes (numerical or linguistic variables) from a set of node attributes, say $A(v) = \{A_1, \ldots, A_I\}$. The i-th node attribute may take, say, J_i values (numerical or linguistic) values, $S_i = \{a_{ij} : j = 1, \ldots, J_i\}$. The set

$$L(V) = \{(I_i, a_{ij}) : i = 1, \ldots, I;\ j = 1, \ldots, J_i\} \quad , \qquad (4.153)$$

is the set of all possible node attribute-attribute value pairs. A primitive is said to be *valid* if it is a subset of L(V) in which each attribute appears just once, and we follow Chan and Cheung (1992) in calling the set of all *valid vertex primitives* Π. The edge set E of G = (V, E) is treated similarly. Edges $e \in E$ are associated with a set of I' attributes, say $E(e) = \{E_1, \ldots, E_{I'}\}$. The i-th edge attribute may take J_i' values, $T_i = \{e_{ij} : j = 1, \ldots, J_i'\}$. The set

$$L(E) = \{(E_i, e_{ij}) : i = 1, \ldots, I' ;\ j = 1, \ldots, J_i'\} \quad , \qquad (4.154)$$

is the set of possible relational attribute-edge value pairs. A relation is said to be *valid* if it is a subset of L(E) in which each attribute appears just once. Suppose Θ is the set of all *valid edge primitives*. With these attribute-values sets for the nodes and vertices of a graph we are ready to define an attributed graph (Chan and Cheung, 1992).

G = ((V,σ), (E,δ)) is an *attributed graph* over (L(V), L(E)) ⇔
(i) V is associated with $\sigma : V \mapsto \Pi$ = *vertex interpreter*; and (4.155)
(ii) E is associated with $\delta : E \mapsto \Theta$ = *edge interpreter*.

The vertex and edge interpreter functions map nodes and edges of the graph G into attributed nodes and edges; (V, σ) is called an *attributed vertex set*; and (E, δ) is an *attributed edge set*. This enables us to associate structural, numerical and symbolic information with each element of the graph. We illustrate the idea of attributed graphs using part of an example from Chan and Cheung (1992). These authors argue that attributed graphs are more useful for handwritten character recognition than the formal language approach because the strokes that comprise a character must be correctly ordered in a string grammar model, and they do not need to be in an attributed graph. Their work is based on the set of stroke primitives shown in the upper half of Figure 4.102, which they take as the basic strokes needed to make Chinese characters: H (line), V (line), P (curve) and N (curve).

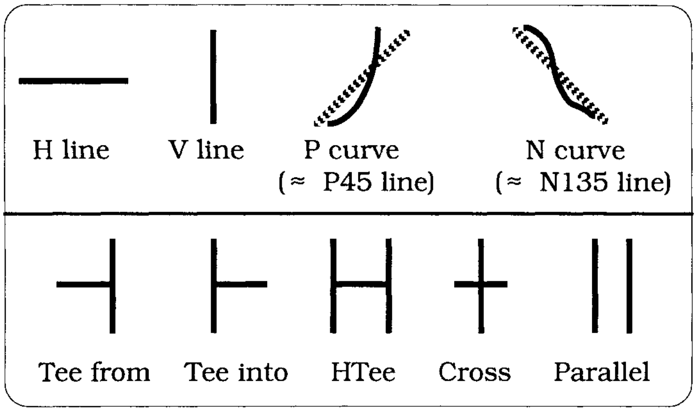


Figure 4.102 Primitive stroke types and joint types for Chinese characters (Chan and Cheung, 1992)

Chan and Cheung assert that a natural way to represent each of the four strokes in Figure 4.102 is by an ordered pair, viz., the vertex of the stroke and the relationship between strokes, interpreted as edges in an attributed graph. First idealizing the ill-defined P and N curves in Figure 4.102 as P and N lines (see Figure 4.92) with angles of 45° and 135° measured counterclockwise from the positive x axis, Chan and Cheung set up the following attributed graph, which is based on four linear *stroke types* (H, V, P45 and N135), two *stroke lengths* (long and short), five *joint types* (Tee from, Tee into, HTee, Cross, Parallel as shown in the lower half of Figure 4.102), three *vertical structural relationships* (on top of, below, no vertical relation) and three *horizontal structural relations* (left of, right of, no horizontal relation). We summarize this construction in Table 4.61.

The bottom third of Table 4.61 shows a sample pair of valid vertex and relational attribute-value pairs, each selected by applying the uniqueness constraint required by the definitions of valid primitive. Chan and Cheung argue that, while crisp attributed graphs could be built and matched for handwritten character recognition using this structure, the second coordinate of each 2-tuple in L(V) and L(E) is really fuzzy in the application domain of interest. For example, strokes are not always vertical, horizontal, etc., and spatial relations such as above and below are often only partially fulfilled (see Figure 5.50). Using this rationale, Chan and Cheung introduce fuzziness into the attributed graph by adding membership functions for the attribute-values in both the vertex and edge domains of the definition in (4.155).

Table 4.61 Attributes for Chinese character recognition

A_1 stroke type	A_2 stroke length	E_1 joint type	E_2 vertical relation	E_3 horiz. relation
a_{11}= V	a_{21}= long	e_{11}= Tee from	e_{21}= above	e_{31}= left of
a_{12}= H	a_{22}= short	e_{12}= Tee into	e_{22}= below	e_{32}= right of
a_{13}= P45		e_{13}= HTee	e_{23}= none	e_{33}= none
a_{14}= N135		e_{14}= Cross		
		e_{15}= Parallel		

L(V)	L(E)
(stroke type, V)	(joint type, Tee from)
(stroke type, H)	(joint type, Tee into)
(stroke type, P45)	(joint type, HTee)
(stroke type, N135)	(joint type, cross)
	(joint type, parallel)
(length, short)	(vert. rel, above)
	(vert. rel, below)
	(vert. rel, none)
	(horiz. rel, left)
	(horiz rel, right)
	(horiz. rel, none)
$\Pi_{e.g.}$	$\Theta_{e.g.}$
(stroke type, V)	(joint type, cross)
(length, long)	(vert. rel, above)
	(horiz. rel, none)

Chan and Cheung define a fuzzy attributed graph as:

$$\begin{aligned}&\tilde{G} = ((V,\tilde{\sigma}),(E,\tilde{\delta})) \text{ is a } \textit{fuzzy attributed graph} \text{ over } (\tilde{L}(V),\tilde{L}(E)) \Leftrightarrow \\ &\text{(i) V is associated with } \tilde{\sigma}: V \mapsto \tilde{\Pi} = \textit{vertex interpreter}; \qquad (4.156)\\ &\text{(ii) E is associated with } \tilde{\delta}: E \mapsto \tilde{\Theta} = \textit{edge interpreter};\end{aligned}$$

where $\tilde{\Pi}$ and $\tilde{\Theta}$ are the sets of valid primitives and relations on edges, validity again meaning each attribute appearing just once, and $(V,\tilde{\sigma})$, $(E,\tilde{\delta})$ are (fuzzily) attributed vertex and edge sets, respectively. The sets $\tilde{L}(V)$ and $\tilde{L}(E)$ still represent all of the possible 2-tuples with first coordinate a vertex or edge attribute, respectively; but the second coordinate becomes a set of values (one for each attribute value) of a membership function on the attribute in question. Thus, instead of attribute value a_{ij} for the jth value of vertex attribute i, we now have $m_i(a_{ij})$, where m_i is a membership function on the i-th attribute, i = 1,...,I. Similarly, membership functions $\{m_1',\ldots,m_{I'}'\}$ are defined on the relational edge attributes. The fuzzy attribute graph of an object as defined in (4.156) reduces to a crisp attribute graph as in (4.155) when all the membership functions $\{m_1,\ldots,m_I\}$ and $\{m_1',\ldots,m_{I'}'\}$ are crisp.

The advantage of using the fuzzy graph model in (4.156) is that attributes can have memberships in each of the attribute values to which they apply. For example, using the ordering in Table 4.61, the pair (stroke type, *) in L(V) can assume a form in $\tilde{L}(V)$ such as (stroke type, (m_{11}(stroke) = 0.7, m_{12}(stroke) = 0.1. m_{13}(stroke) = 0.9, m_{14}(stroke) = 0)), which indicates memberships of the stroke being evaluated in each of the vertical, horizontal, P45 and N135 directions.

The fuzzy attributed graph adds an intuitively satisfying element to its crisp counterpart for this application. But the use of membership functions for the elements of the valid primitive and relation sets $\tilde{\Pi}$ and $\tilde{\Theta}$ complicates the use of similarity measures that assess the extent to which graph representations of objects are similar. Chan and Cheung (1992) introduce three measures for assessing the extent to which a pair of fuzzy attributed graphs agree. The three definitions require $\tilde{G}_1$ and $\tilde{G}_2$ to be *monomorphic*. Two fuzzy attributed graphs $\tilde{G}_1$ and $\tilde{G}_2$ are said to be *monomorphic* if they are connected by a 1-1 mapping that preserves incidence relations (determination of the mapping is known to be np-complete, Aho et al., 1974). In equations (4.157), we assume that $\tilde{G}_1$ and $\tilde{G}_2$ are monomorphic. Now we can state definitions for the three measures.

Feasibility : a measure of similarity between primitives $v_1 \in \tilde{G}_1$ and $v_2 \in \tilde{G}_2$ in two fuzzy attributed graphs,

$$\alpha(v_1,v_2) = \bigwedge_{i=1}^{I}\left\{\bigvee_{j=1}^{J_i}\left\{m_i^1(a_{ij}(v_1)) \wedge m_i^2(a_{ij}(v_2))\right\}\right\} \quad . \tag{4.157a}$$

In (4.157) m_i^1 and m_i^2 are membership functions for the i-th vertex attribute in the two fuzzy graphs.

Compatibility: a measure of similarity between edges $e_1 \in \tilde{G}_1$ and $e_2 \in \tilde{G}_2$ in two fuzzy attributed graphs,

$$\beta(e_1,e_2) = \bigwedge_{i=1}^{I'}\left\{\bigvee_{j=1}^{J_i'}\left\{m_i^{1'}(e_{ij}(e_1)) \wedge m_i^{2'}(e_{ij}(e_2))\right\}\right\} \quad . \tag{4.157b}$$

where $m_i^{1'}$ and $m_i^{2'}$ are membership functions for the i-th relational attribute in the two fuzzy graphs.

Degree of match: a measure of similarity between two fuzzy attributed graphs $\tilde{G}_1$ and $\tilde{G}_2$,

$$\gamma(\tilde{G}_1, \tilde{G}_2) = \bigwedge_{i \in V_1} \left\{ \alpha(i, h(i)) \cdot \left\{ \bigwedge_{(i,j) \in E_1} \left[\beta(e_1(i,j), (e_2(h(i), h(j)) \right] \right\} \right\}. \qquad (4.157c)$$

where h(i) is the vertex in $\tilde{G}_2$ that matches vertex i in $\tilde{G}_1$, and $e_k(i,j)$ is the edge joining nodes i and j in $\tilde{G}_k$, $k = 1,2$.

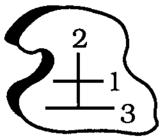
Using the concept of graph matching embodied in (4.157c), Chan and Cheung say that $\tilde{G}_1$ and $\tilde{G}_2$ are *λ-monomorphic* when $\tilde{G}_1$ and $\tilde{G}_2$ are monomorphic and the degree of match $\gamma(\tilde{G}_1, \tilde{G}_2) \geq \lambda$. Computing (4.157c) requires a method for establishing matched pairs of vertices and edges; Chan and Cheung use a tree search algorithm due to Akinniyi et al. (1986) to establish the needed monomorphisms.

After proving some properties of fuzzy attributed graphs, Chan and Cheung provide an example of their use in representing one Chinese character in terms of a set of simpler primitives called radicals, each of which is itself decomposed into the four stroke primitives shown in the upper panel of Figure 4.102. A key point is that each of the radicals is a *crisp, human-derived* template that is thought of a one of the constituents of more complex templates, and subsequently, characters. The crisp templates (radicals) illustrated in the paper are

土 contained in 王 ; and 日 contained in 目 . (4.158)

For example, values for representation of the crisp attributed graph for the template ± , shown as Table III in Chan and Cheung (1992), are reproduced in our Table 4.62.

Table 4.62 Attribute values for the template ±



	length		stroke type			
Stroke	long	short	H line	V line	P curve	N curve
1	0.00	0.00	1.00	0.00	0.00	0.00
2	0.00	0.00	0.00	1.00	0.00	0.00
3*	0.00	0.00	1.00*	0.00	1.00*	0.00

The values marked by * in Table 4.62 are the ones shown in Table III of Chan and Cheung (1992) for the third stroke. It makes more sense to us to retain the value 1.00 in the H line column for stroke 3 as shown, and a have a value of 0.00 for the P curve column, but our rendering in Figure 4.102 is by hand and eye, and none of us know enough about Chinese characters to make a call on this - it may be that the combination of H line and P curve really produces the required stroke. Either the P curve (not P line!) value is a typographical error, or we don't appreciate fine differences in the strokes well enough. Chan (1996) extracts the stroke sequence (H, V, H) for this prototype using the four primitives (H, V, P45, N135) with a learning method to be discussed shortly, so we suspect our supposition is correct: stroke 3 should have the four memberships (1, 0, 0, 0) for the strokes (H, V, P, N). It also seems strange to us that all of the length memberships in Table 4.62 are 0.00; perhaps there are some incorrect values here as well.

Chan and Cheung (1992) develop a table of fuzzy attribute values for the character 去 by first thinning its image, segmenting it, and then defining membership functions for each of the vertex and relational attribute values needed to represent 去 as a fuzzy attributed graph. Then they match this character to the attributed graph of the radical shown in Table 4.62. The two graphs are reproduced in Figure 4.103.

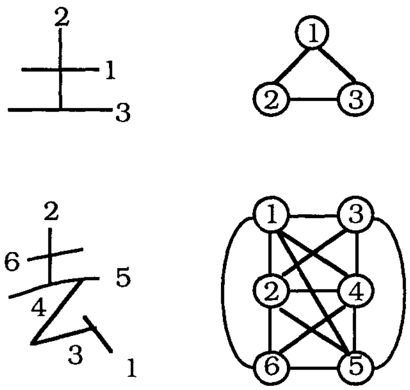


Figure 4.103 Graphs of the characters 去 and 土

When 去 is matched against 土, the radical is extracted twice, once from a monomorphism between vertices {1,2,3} of 土 and {6,2,4} of 去; and vertices {1,2,3} of 土 and {6,2,5} of 去. Using equation (4.157c), these two subgraph matches produce the following degrees of match: $\gamma = 0.97$ for the vertex pairs {(1,6), (2,2), (3, 4)}; and $\gamma = 1.00$ for the vertex pairs {(1,6), (2,2), (3, 5)}. Certainly the strokes {6, 2, 5} in the character 去 in Figure 4.103 are a better match to {1,2,3} for

土 than the strokes {6,2,4}, so the degree of match agrees with a visual assessment of the complex character (don't place all your evaluation faith on our reproduction in Figure 4.103, which was done "by eye", not by obtaining the exact characters from the authors).

Finally, Chan and Cheung (1992) give some statistics for a run of their system (which, incidentally, used a modified version of fuzzy c-means discussed in Cheung and Chan (1986) for preclassification) on 8,980 samples of the 240 most frequently used Chinese characters. They do not specify the source of the data, nor how many crisp, human-trained templates (represented as crisp attribute graphs) were used. Without FCM preclassification, 8,086 of the labeled test samples were correctly labeled, 140 samples were incorrectly labeled, and the remainder were "rejects" = undecided, so the test error rate without preclassification was about 9.95% (we count undecided as mistakes). With FCM preclassification, the error rate dropped to 9.2%. This system was not compared to any other method, nor was the data set shared by other studies we know of, so it's pretty hard to place the accuracy of these results in the overall context of handwritten character recognition. On the other hand, it is one of the few examples we can offer of a complete, working syntactic approach to pattern recognition that incorporates fuzzy models.

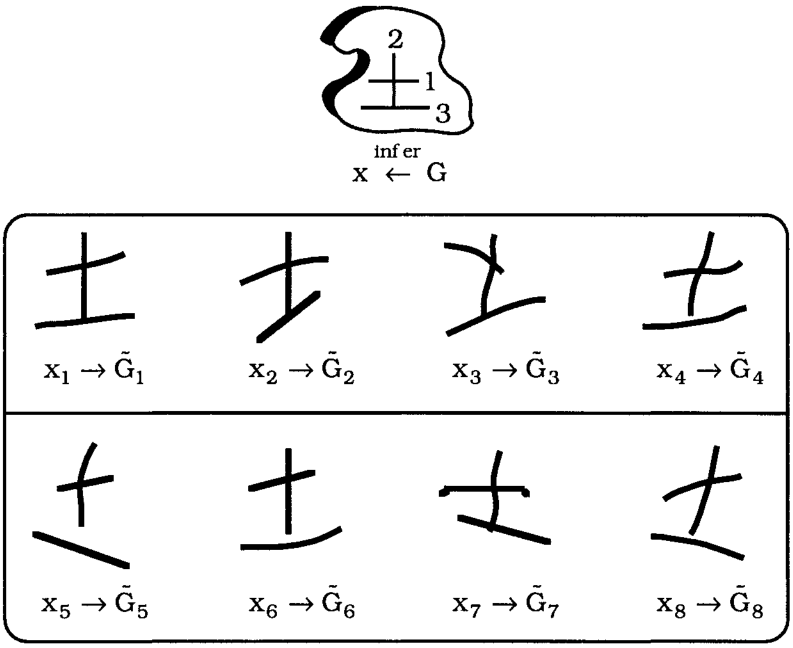

2
1
3
infer
$x \leftarrow G$
$x_1 \rightarrow \tilde{G}_1$
$x_2 \rightarrow \tilde{G}_2$
$x_3 \rightarrow \tilde{G}_3$
$x_4 \rightarrow \tilde{G}_4$
$x_5 \rightarrow \tilde{G}_5$
$x_6 \rightarrow \tilde{G}_6$
$x_7 \rightarrow \tilde{G}_7$
$x_8 \rightarrow \tilde{G}_8$

Figure 4.104 Inferring 土 $= x \leftarrow G \leftarrow \{\tilde{G}_1, \ldots, \tilde{G}_8\}$ from training data

In a sequel, Chan (1996) states that a deficiency of the work of Chan and Cheung (1992) is that crisp template graphs such as the one shown in Table 4.62 are derived by humans. Chan asserts that this is tedious, time-consuming and error prone, and in the 1996 paper he presents an interesting algorithm that *learns* crisp templates from a set of training data. The training data are (crisply labeled) fuzzy attributed graphs of handwritten characters. Figure 4.104 shows the set of 8 samples of the character ± that (roughly) correspond to Figure 1 in Chan (1996). Which of these 8 characters do you think best matches the character ± ?

In Figure 4.104 the sample characters are all crisply labeled as ± and are denoted by $\{x_1,...,x_8\}$. Fuzzy attribute graphs $\{\tilde{G}_1,...,\tilde{G}_8\}$ of $\{x_1,...,x_8\}$ are derived using fuzzy membership values in each of the four strokes H, V, P45 and N135 shown in the upper half of Figure 4.102 for each of the three strokes sequenced and numbered as {1,2,3} in Figure 4.103. The unknown character that is to be inferred is x = ± , whose crisp attribute graph is denoted by G at the very top of Figure 4.104. Chan (1996) exhibits tables of memberships of each of the 8 training data strokes {1,2,3} in each of the four primitives H, V, P45 and N135. For example, the memberships of the 8 training data strokes for the first stroke (the upper of the two horizontal strokes needed to make ± as shown in Figure 4.103) are listed in Table 4.63.

Table 4.63 Memberships for $\{x_1,...,x_8\}$, *Stroke 1*, in H, V, P45, N135

x	H	V	P45	N135
1	0.85	0.00	0.22	0.00
2	0.83	0.00	0.19	0.00
3	0.37	0.00	0.00	0.49
4	0.91	0.00	0.13	0.00
5	0.84	0.00	0.21	0.00
6	0.98	0.00	0.39	0.00
7	1.00	0.00	0.03	0.00
8	0.94	0.00	0.31	0.00

Take a look at the 8 training data in Figure 4.104: do you agree with the membership values shown in Table 4.63 *for stroke 1* of these eight characters? The horizontal component (column H in the table) seems to agree with a visual assessment: sample x_7 has the "most horizontal" stroke, and its memberships reflect this; only character x_3 has a negative slope in stroke 1, and again, the memberships do reflect this, assigning stroke 1 of x_3 a membership of 0.49 in the fuzzy primitive N135 = negatively sloped line segment at 135°; and so on.

Chan (1996) develops a training algorithm that guarantees that the crisp attribute graph G inferred from the training data is λ-monomorphic. First, each pair of training graphs are matched to each other to establish vertex correspondence. Then *each of the strokes* needed to make up the character being inferred is expressed as a "polynomial" in variables that correspond to the primitives used to comprise the stroke (here, the four variables are H, V, P45 and N135). Chan (1996) devotes several pages to definitions and results concerning these polynomial forms. This theory, along with the information shown in Table 4.63 and two tables like it for strokes 2 and 3 (that are in Chan's paper which are not reproduced here) are used to create the polynomials (in the four variables H, V, P and N) shown in (4.159a). We let P=P45 and N=N135 to shorten the expressions in (4.159):

$$p_1(H,V,P,N) = 0.37H + 0.49HN + 0.37HP + 0.39HPN. \qquad (4.159a)$$

$$p_2(H,V,P,N) = 0.3V + 0.39VP \qquad ; \text{and} \qquad (4.159b)$$

$$p_3(H,V,P,N) = 0.31H + 0.61HN + 0.31HP + 0.61HPN. \qquad (4.159c)$$

Chan then takes the term from each of the polynomials in (4.159) with the minimum number of strokes as the correct stroke for the crisp character being inferred. Thus, from the first terms in each of the polynomials in (4.159) we have the three ordered strokes {1,2,3} = {H, V, H}, so the method correctly infers from the training data that $\pm = x \leftarrow G$. Finally, Chan (1996) gives the following values for the degree of match γ in (4.157c) between G and each of the fuzzy attribute graphs $\{\tilde{G}_1, \ldots, \tilde{G}_8\}$ of the characters shown in Figure 4.104 as $\gamma(G, \tilde{G}_j)$ = 0.85, 0.83, 0.37, 0.43, 0.31, 0.84, 0.81, 0.30 as j runs from 1 to 8. Looking back at Figure 4.104, these values assert that character x_1 at 0.85 is the best match to $\pm$, very closely followed by x_6 and then x_2; and that character x_8 is the worst match, but only very slightly worse than x_5. Do you agree with this assessment of the matches between $\pm$ and the training data?

Chan (1996) states that the overall complexity of his training algorithm is "essentially" linear or $O(n)$, n being the number of training data. No example is given to illustrate the results of this method on a set of test data. Nonetheless, these two papers provide you with a nice example of a relational approach to structural pattern recognition with fuzzy models.

The last method we discuss in this subsection, reported in Srinivasan and Kinser (1998), also uses a relational approach to structural decomposition of objects in images (and again, images of ...... agghhh...... handwritten characters!). The structural information that relates primitives to each other is not carried by a relational graph. Instead, each primitive is associated with a membership function that aggregates structural relationships to other primitives, and then the structural information is integrated with spatial location data (coordinates of other primitives) to produce scores for various possibilities submitted to the system.

Srinivasan and Kinser assert that one key to mammalian image recognition is the process of *foveation*, defined by them as the ability to perceive and then rapidly change focal points (regions of interest) in an input image. Foveation points are thought to be corners, and to a lesser extent, line segments of objects in the image, so these are chosen as the primitives for structural descriptions of objects in images. This idea stands in sharp contrast to our previous examples of sets of primitives, at least for character recognition, which have taken various strokes (segments of arcs) as the building blocks of structural decomposition. Figure 4.105 illustrates the five primitives $\{a_1,a_2,a_3,a_4,a_5\}$ that Srinivasan and Kinser (1998) nominate for the letter "A".

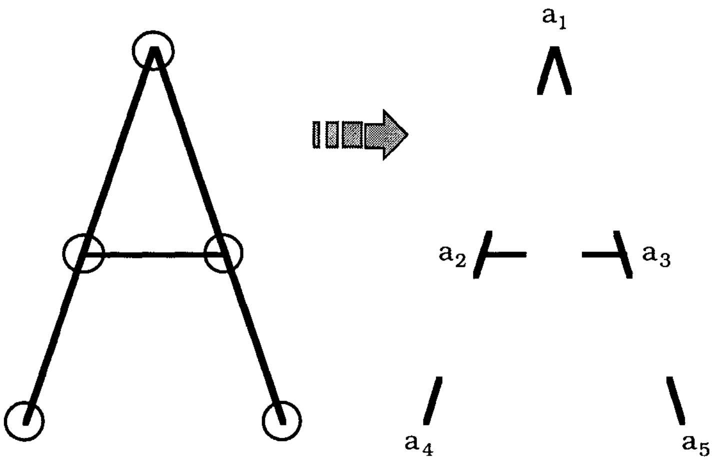


Figure 4.105 Five primitives for the crisp, prototypical letter "A"

In Srinivasan and Kinser's model, each letter in an alphabet may require a different set of primitives. Thus, the 26 upper case letters used in the English language might require, say, 100 primitives for prototypical representation. This notion of primitives is in some sense less primitive (!) than previous schemes that rely on many fewer primitives. In fairness to these authors, we point out that this method has a much more general objective (automatic target recognition) than the papers we have discussed that focus on the specific application of handwritten character recognition. Srinivasan and Kinser use character recognition as a nice way to illustrate various points of their model, and do not claim that it will compete well in this particular application domain.

How are the prototypes for various characters found? Much like Chan and Cheung (1992), an earlier effort discussed in Srinivasan et al. (1996) depended on human derived crisp prototypes for each letter. And in a manner very like Chan (1996), Srinivasan and Kinser (1998) then turned to a trainable learning model that could be used to derive primitives from training data. The learning model discussed in Srinivasan and Kinser has several elements. First, foveation points are detected by a *pulse coupled neural network* (PCNN), which essentially functions like a combined edge and corner detector. The PCNN (Johnson, 1994) is a biologically motivated computational structure that attempts to model the visual cortex of the cat. Srinivasan and Kinser assert that a by-product of the PCNN's inherent ability to segment images is that it collects foveation points (edges and corners).

Once the foveation points are found, each foveation point from the PCNN image is transformed into a new image by applying a "barrel" transformation which is centered at the detected foveation point. The purpose of the barrel transformation is to distort the image, thereby placing more emphasis on intensities in a neighborhood of the foveation point. The functional form of the barrel transformation is given in polar coordinates. For example, the r (radius) component of each point is transformed as $r_{new} = r_{old}^{b} / d^{b-1}$, where d is half of the frame width and the parameter b controls the amount of distortion introduced at this location. Srinivasan and Kinser show the five new images that result from applying this barrel transformation to each point in $\{a_1, a_2, a_3, a_4, a_5\}$ for the letter "A" in Figure 4.105.

Once primitives are extracted, each prototype (such as the set of five in Figure 4.105) is used to develop a set of fractional power filters (Kumar, 1992) based on Fourier coefficients gotten from the image. This set of filters provide one part of the classification strategy, because they are used to compute the correlation between the trained filter (of a particular primitive) and a detected primitive in an image that is to be classified. For example, the set of primitives in Figure 4.105 would result in a set of five correlation filters which act as peak detectors for each foveation point produced by the PCNN.

Structural information about the relationships of primitives to each other is imbedded in a set of membership functions as follows. For each foveation point in a primitive, a "fuzzy fan" is constructed that looks for each of the other primitives that are expected for the template being used. The fuzzy fan acts just like an angle - limited sweep searchlight whose intensity falls off as the angle from the center of the search increases on either side of the expected location of the target (which here is another primitive in this particular template). Figure 4.106 (Figure 8 of Srinivasan and Kinser, 1998) shows the four fuzzy fans (not to be confused with the FANs = fuzzy aggregation networks in Section 4.7) that are centered at the primitive a_2 that search for the other four primitives in the letter "A" that are shown in Figure 4.106.

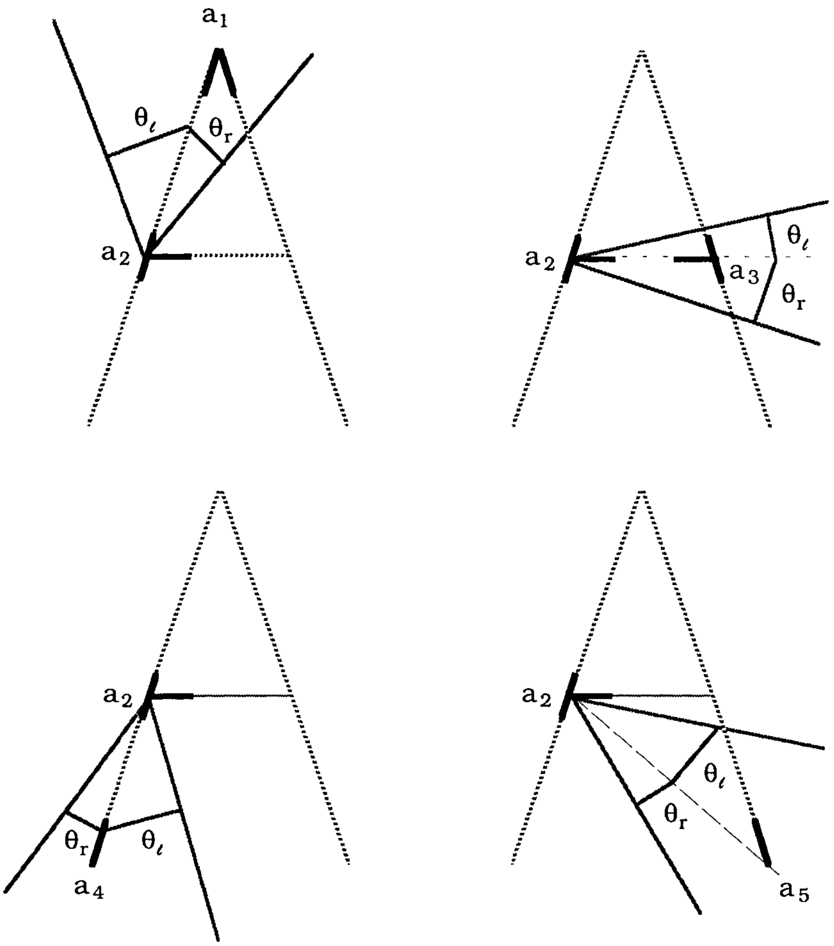


Figure 4.106 Fuzzy fans from a_2 to: a_1, a_3, a_4 and a_5

In each of the four views in Figure 4.106 imagine yourself positioned at the center of primitive a_2. Suppose that you sight directly towards the known center of the prototypical primitive a_1 as shown in the upper left panel of Figure 4.106. Since a_1 is directly in your line of sight, the membership of this sighted crisp primitive will be 1 in the fuzzy set of locations near the expected (angular) location for a_1. On the other hand, if instead of a_1, the foveation point that you see is either left or right of the center of search by an angle θ_ℓ or θ_r, the membership of the sighted point will decrease as the angle from center increases. Srinivasan and Kinser (1998) specify limits on each set of sweep angles. For a searchlight centered at a_2, the four sets of limit angles depicted in Figure 4.106 starting at the upper left panel and working clockwise, i.e., a_1, a_3, a_5 and then a_4, are: $\theta_\ell = \{30°, 10°, 25°, 30°\}$, $\theta_r = \{15°, 15°, 20°, 15°\}$. The membership function $m_{a_2 \to a_k} : [0, 30°] \mapsto [0,1]$ decreases linearly with θ from 1 at $\theta = 0$ to 0 at the left and right limits of each search pattern. Thus, the structural relationship (here angular information) between a_2 and the four other foveation points that can be related to *just this primitive* for the letter "A" is captured by the values of the four membership functions $m_{a_2 \to a_k}$, k = 1, 3, 4, 5.

For the letter "A" each of the 5 primitives shown in Figure 4.105 will produce 4 membership values, so application of the structural template for this letter to any test input results in a set of 20 membership values. In operation then, each character will have a crisp template which consists of : a set of primitives such as the ones in Figure 4.105; a set of correlation filters, one for each primitive; and a set of (angle measuring) membership functions, one for each primitive. When an input image is submitted to the recognition system the PCNN detects all foveation points in it. Each foveation point is expanded into a set of images for each possible template. Next, a given point is compared to all templates by computing its "fuzzy score". Srinivasan and Kinser (1998) use any one of four scoring indices. Finally, the input character with the highest fuzzy score is declared the winner, and the input character receives this crisp label.

Srinivasan and Kinser (1998) give some very limited results based on training and testing with a few dozen samples of the letters "A" and "M". Error rates are not discussed at length, so the general utility of this model as a character recognition system is very hard to assess (and, as we have already mentioned, these authors really have other fish to fry anyway). However, this is a nice example of how fuzziness can be used to incorporate structural information into classifier design that is not dependent on either the formal language or relational graph approach, so we think it has a lot of pedagogical value in the context of this section.

4.11 Comments and bibliography

Feature analysis

We first stated and illustrated the importance of feature analysis in Section 2.5, and have iterated this point many times over in this chapter, and will do so again in Chapter 5. Using many features increases the time and space complexity of classifier design. More surprisingly, increasing the number of independent features used during supervised learning beyond some (theoretically unknown) optimal number can actually degrade classifier performance! Hughes (1968) demonstrated that when the number n of samples available for design is fixed, increasing the number of features p beyond a certain size is counterproductive - that is, the apparent error rate of D will increase as p increases. Another point worth repeating is that the quality of a set of features depends importantly on the algorithm that uses them. Thus, features that endow a k-nn classifier with a low error rate might not be useful for training good 1-np designs.

In some cases, mapping the original features into a new set can actually improve performance in classification problems. This is effectively what is done by multilayered neural networks, where we can regard in outputs of any hidden layer as new features derived from the input values to the layer in question (Haykin, 1994). Chiang and Gader (1997), and Gader et al. (1997b) demonstrate this quite effectively in the omnipresent handwritten word recognition domain. The problem is the same as that discussed in Example 4.12, where dynamic programming is used to combine groups of primitives (pieces of characters resulting from an oversegmentation of a word) to generate the match confidence between that actual image and a string from a lexicon. The key to improved performance is in generating "good" upper and lower case character confidences for the various unions of primitives. The baseline system used was a pair of MLP's (one for upper case and one for lower case) which had standard input feature sets and 27 output nodes: one for each character and one node for "non-character". It is this last situation that is problematic: how do you characterize "non-characters"?

The approach taken in (Chiang and Gader 1997, and Gader et al. 1997b) was to train a 15×15 Self-Organizing Feature Map (SOFM) on the original feature data only for valid characters. After training, the activation levels of the 225 SOFM nodes for a given input were used as features to train a MLP. In a test (using standard US Postal Service data sets), the baseline MLP's did considerably better than those trained with SOFM activations at doing isolated character recognition (77.5% vs. 73.9% correct in testing). However, the goal is not isolated character recognition, but handwritten word recognition. Hence, this is again a situation that calls for the principle of least commitment. In fact, valid characters produced well-defined activation regions, while non-characters generally had uniformly low activation values across the SOFM. Putting both neural network confidence generation devices into the dynamic programming module demonstrated the advantage of the feature mapping. Using the SUNY "BD" city data set (317 words), the transformed feature networks produced a 10% increase in word recognition over the baseline on lexicons of average size 100 (89.6% vs. 79.8% correct in testing). The paper in *IEEE Computer* (Gader et al., 1997b) contains a very hip picture - reproduced here as Figure 4.107 - which demonstrates the topological properties of the SOFM in a digit recognition problem - the node prototypes(weight vectors) for a 10×10 SOFM trained on the raw digit images are displayed as images. Note how the "prototype images" blend into each other in the topological display space.

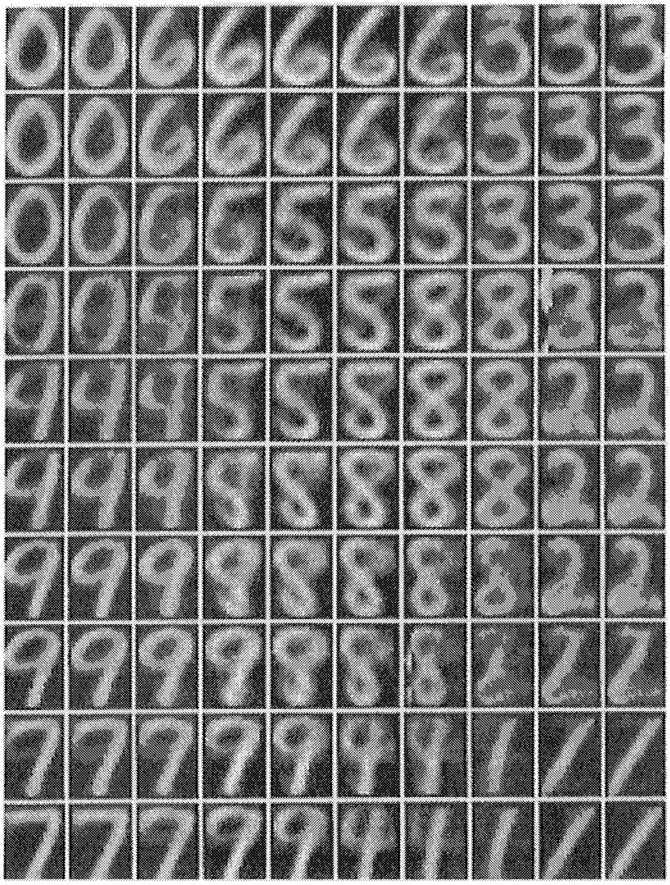


Figure 4.107 A SOFM represents fuzziness of the character classes

Any and all numerical features that can be extracted from objects can be used as a basis for fuzzy clustering and classifier design. However, not much work has been done on extracting fuzzy numerical features or features from fuzzy subsets of objects. We will discuss some of the topics mentioned in this paragraph in more detail in Chapter 5, but we include a few sentences on them here because you might expect to find this discussion at the end of Chapter 4. For two dimensional fuzzy subsets Rosenfeld (1979, 1984, 1992) and Rosenfeld and Haber (1985) extended many concepts from crisp geometry to fuzzy geometry, and generalized many terms that are traditionally used in the analysis of spatial properties of objects in binary images to the fuzzy case.

Some of the spatial properties that were defined by Rosenfeld that will be discussed in Chapter 5 include fuzzy area, fuzzy perimeter, fuzzy height, fuzzy extrinsic diameter, fuzzy intrinsic diameter, and fuzzy elongatedness. Pal and Rosenfeld (1988), Pal and Ghosh (1990) and Pal (1992b) have defined similar geometric attributes such as index of area coverage, degree of adjacency, length and breadth, and have developed low- and intermediate-level vision algorithms based on such attributes. Dubois and Jaulent (1987) showed that some of Rosenfeld's definitions of the geometric properties of fuzzy regions namely area, height and perimeter correspond to expected values in evidence theory. Krishnapuram et al. (1993a), Krishnapuram and Medasani (1995), and Medasani et al. (1999) consider the computation of fuzzy features from real images.

Apart from the fact that the definitions for properties of fuzzy regions reduce to the corresponding crisp definitions when the images are binary, no other theoretical justification has been provided in the literature for the use of fuzzy set theory to measure geometric and non-geometric properties of image regions (Medasani et al., 1999). In addition to our discussion on spatial relations in Section 4.10.B, several other authors have developed methods of defining fuzzy spatial relationships among regions in the plane (Keller and Sztandera, 1991, Keller and Wang, 1996, Krishnapuram et al., 1993a, Miyajima and Ralescu, 1994, Wang and Keller, 1999a). In Chapter 5 fuzzy spatial relations will be defined and used in image processing applications.

Gitman and Levine (1970) augment the measured features with the "importance" of each feature, and cluster with this added information as part of the data. Bezdek and Castelaz (1977) report that a fuzzy 1-np classifier (the prototypes being the cluster centers generated by FCM) increases by about 10% the apparent probability of correct classification above (an estimate of) the asymptotic error rate of *all* k-nn classifiers for a set of n=300 stomach disease patients each represented by 11 binary-valued features. (Bear in mind that the apparent error rate is a finite sample based statistic - the asymptotic optimality of k-nn rules via Cover and Hart's famous theorem (1967) is well known.) Di Gesu and Maccarone (1986) used cluster analysis together with possibility theory for selecting the most significant variables from electron spin resonance spectroscopy measurements on patients with a brain injury. Bezdek and Chiou (1988) use FCM clustering of labeled data followed by feature extraction with principal components, Sammon's algorithm or triangulation to produce visual displays of high dimensional data. Petersen et. al (1997) study the use of fuzzy decision trees (Cios and Sztandera, 1992) to select subsets of features from biosignals collected to assess the depth of anesthesia of medical patients.

Pal and Chakraborty (1986) use fuzziness measures such as the index of fuzziness and entropy to compute interset and intraset ambiguities for feature evaluation. Pal (1992a) extended this idea to evaluate the importance of any subset of features, to provide an average quantitative index of goodness and a comparison of the algorithm with statistical measures like divergence, J-M distance, and Mahalanobis distance. The application of Pal's algorithm has also been demonstrated on six class, three feature vowel data, four class five feature consonant data, and three class fifteen feature mango leaf data. One drawback of this approach is that it can be used only to assess features for a pair of classes (c=2).

When $c > 2$, it may happen that feature f_1 is good for discriminating between class i and j, while feature f_2 may be a better discriminator between classes k and q. Further, some other feature f_3 may be, on average, a better discriminator for the classes i, j, k, and q taken together. Thus, Pal and Chakraborty's feature evaluation index is not particularly useful for assessing the goodness of a feature with respect to all c classes taken jointly. To get around this problem Pal (1992b) extended his earlier work by defining the average *feature evaluation index* (AFEI) as the weighted sum of the FEI's for all pair-wise classes, where the weights are sample-based estimates of the prior probabilities of the classes used to compute the FEI. Thus, the AFEI depends on the cardinalities of the different classes - an undesirable dependency. De et al. (1997) further modified the AFEI by defining an *overall feature evaluation index*, which is not directly influenced by the size of the classes, and which considers all possible pairs of classes. They compare these fuzzy indices to several non-fuzzy methods for feature selection based on multilayer perceptron neural networks.

Recently Thawonmas and Abe (1997) proposed a feature selection method based on class regions generated by the fuzzy classifier discussed in Abe and Lan (1995). Their (Thawonmas and Abe, 1997) feature selection algorithm eliminates irrelevant features based on an index they call the *exception ratio*, which is computed using the degree of overlap between class regions. The exception ratio is defined so that given two feature subsets of the same cardinality, the feature set with the lowest sum of exception ratios is expected to contain the most relevant features. Based on this idea their algorithm uses a backward selection search (Fukunaga, 1991) which starts with all the given features and eliminates irrelevant features one by one. Thawonmas and Abe evaluate the quality of the selected features using the fuzzy classifier described in Abe and Lan (1995) and a FFBP neural network. The reported performance is quite satisfactory.

Among the many non-fuzzy techniques that have been employed for feature selection, methods based on the k-nearest neighbor rules have been particularly effective. See Devijver and Kittler (1982) or Dasarathy (1990) for good introductions to this vast (non-fuzzy) literature. A simple but effective method to reduce the number of subsets considered is called *forward sequential search*. In this method all subsets of 1 feature are used. In a standard training/testing paradigm such as "leave-one-out" or n-fold cross validation (jackknifing) as discussed in Section 4.1, these subsets are scored for overall recognition. Leave-one-out requires that the classifier is trained on all but one sample, that sample is applied as a test, the result is noted, and the process is repeated until all training samples have been left out for testing. The recognition rates are compiled from the tests. This technique is resource consuming, but is good, particularly if the size of the training set is limited. The idea is that most (all but one vector) of the data is used to build the classifier, so it should behave like the final version using all the data. But each time, an unused data point is shown to the algorithm for testing. N-fold cross validation is just a less exhaustive version of leave-one-out. Here, all but one nth of the data is used to train the system, the left out portion is scored, and the process is repeated n times.

After the best single feature is chosen, subsets of two features are considered. However, instead of looking at all such sets, only the sets which contain the single best feature are used. Scoring rates for the two feature subsets should increase over those for one feature. This process is repeated always adding one feature to the winner of the previous step. Clearly, it is possible that the best two features do not contain the best one feature, but this approach is a reasonable compromise to having to try all subsets. What should happen is that after a while, the increase in scoring rate upon adding features will level-off. That's when you can stop the process.

Genetic algorithms have also been used for feature selection and extraction (Kuncheva and Bezdek, 1998, Pal et al., 1998). See Velthuizen et al. (1996) for a study of the effectiveness of using GAs for feature selection when various fitness functions (including FCM functional J_m) are used to evaluate linear combinations of the original features.

Prototypes and prototype classifiers

Multiple prototype classifier design is not heavily represented in the literature. Chang (1974) discussed one of the earliest (non-fuzzy) methods for generating multiple prototypes from labeled data, and illustrated it by finding 14 prototypes in the Iris data that were consistent (zero resubstitution errors). A modified version of Chang's algorithm given in Bezdek et al. (1998a) achieves consistency for Iris with 11 prototypes. Dasarathy (1994a) discusses a technique for finding what he calls a minimal consistent subset of the training data (recall that a set of labeled prototypes is consistent if they produce zero resubstitution errors). This technique selects points from the labeled data (cf. Figure 4.1) as opposed to *extracting* points from it (cf. Figure 4.2), and finds 15 vectors *in* the Iris data that provide a consistent 1-nearest neighbor (Section 4.4) design for Iris. Kuncheva and Bezdek (1998) show that Dasarathy's method is not minimal by finding 11 consistent points in Iris using a genetic algorithm technique. Yen and Chang (1994) develop a nearest multiple prototype classifier by modifying FCM, resulting in a method they call MFCM. The best results they report for Iris are 8 errors using 7 relabeled MFCM prototypes. Yan (1993) uses a two layer feed forward neural network to construct prototypes from the training data.

We offer a *conjecture* about the efficacy of using sequential versus batch models to generate multiple prototypes for the 1-nmp classifier. Sequential updating of the prototypes in CL models such as LVQ, SOFM, GLVQ-F, FOSART and SCS encourages "localized" prototypes which are able, when there is more than one per class, to position themselves better with respect to subclusters that may be present within the same class. This leads us to conjecture that batch algorithms are at their best when used to erect 1-np designs; and that sequential models are more effective for 1-nmp classifiers. When c is small relative to n (e.g., c = 5 regions in an image with n = 65,536 pixel vectors), batch models (Chapter 2) probably produce more effective prototypes, because they take a global look at the data before deciding what to do; but if c = 256 (e.g., when using a VQ algorithm for image compression), sequential updating may hold an advantage, as it localizes the update neighborhood, and that objective is more in line with sequential models. Karayiannis (1997c) discusses a general methodology for constructing fuzzy LVQ-type competitive learning algorithms.

Another factor that must be weighed here is the number of parameters, say n_p, that an algorithm is asked to learn. The integers c (number of classes) and p (number of features), along with n (number of samples) determine n_p in almost all of these models.

Usually p and n are fixed, so n_p increases with c (not necessarily linearly). There is very little theory that relates to the adequacy of the four parameters c, p, n and n_p as functions of each other, and whatever you believe about this, your decision about what prototype generator to use should also reflect the difference between the type of optimization done by sequential (local) and batch (global) methods.

We pointed out in section 4.4 that one way to regard the k-nn rule is as an extreme form of the multiple prototype classifier (or vice versa), where every point in the training data is regarded as a prototype for its class. In this case the 1-nn and 1-np rules coincide. We have not devoted much space to algorithms that *select* a subset $\hat{X} \subset X_{tr}$ of "high quality" points from the training data X_{tr} (as illustrated in Figure 4.1), because there have not been many fuzzy models developed towards this end. This is an important and viable option for both crisp and soft classifier design, and we want to devote a paragraph or two to this topic for system designers, who, after all, just want the best possible classifier.

Kuncheva and Bezdek (1999) develop a setting for *generalized nearest prototype classifier design* that provides a common framework for a number of prototype generators discussed in Chapter 4. These authors discuss prototype classifiers based on clustering and relabeling (Section 4.23.B); radial basis function networks (Section 4.8.C); learning vector quantization (Section 4.3.D); nearest neighbor rules (Section 4.4); and Parzen windows (Duda and Hart, 1973). Five questions are discussed by Kuncheva and Bezdek: (i) how many prototypes to look for? (ii) how to look for the prototypes? (iii) how to label the prototypes? (iv) how to define similarity between the training data and prototypes? and (v) how to combine the label information with the similarities?

Numerical examples based on the (real) Iris data and the 2-spirals data lead Kuncheva and Bezdek (1999) to conclude that methods which don't use the labels during the extraction or selection of the prototypes (such as clustering and relabeling) cannot generally be expected to compete with supervised learning methods that use the labels actively during acquisition of the prototypes. Their best result for the Iris data is 2 resubstitution errors using c = 3 *selected* prototypes (points in Iris) found by an edited 1-nn rule. Compare this with the results in Table 4.10, where LVQ needs c = 7 *extracted* prototypes (points built from Iris) to achieve 3 resubstitution errors on Iris. Of course, we have seen in Example 4.21 that a standard multilayered perceptron can achieve 0 resubstitution errors on Iris without using prototypes. This may leave you wondering - why bother with prototype classifiers at all? Well, it's certainly a point worth thinking about. Perhaps the best answer we can make is to reiterate that they are simple, cool, and effective in many problems, and beyond this, they are pleasingly intuitive. Why? We remind you of our quote by Pavlidis (1977, p. 1): "the word *pattern* is derived from the same root as the word *patron* and, in its original use, means something which is set up as a perfect example to be imitated. Thus pattern recognition means the identification of the ideal which a given object was made after.".

k-nn rules

Apparently Jozwik (1983) wrote the first paper on the fuzzy k-nn rules. Keller et al. 's (1985) version of fuzzy k-nn rules was discussed at length in Section 4.4. Bezdek et al. (1986c) gave a somewhat different presentation of fuzzy k-nn rules, and made a comparison between their approach and that of Jozwik. Kuncheva and Bezdek (1999) discuss a very general framework for generalized nearest prototype classifiers that includes soft k-nn rules as one instance of their model.

There are many other interesting variations of the k-nn rule. For example, Dudani (1976) weighted the i-th nearest neighbor $\mathbf{x}_{(i)}$ of the point $\mathbf{z}$ to be classified as follows. Let $\{\delta_{(i)} = \delta(\mathbf{x}_{(i)}, \mathbf{z}) : i = 1, \ldots, k\}$ be the ascendingly ordered set of nn distances to $\mathbf{z}$, and define

$$w_{(i)} = \left\{ \begin{array}{cc} \dfrac{\delta_{(k)} - \delta_{(i)}}{\delta_{(k)} - \delta_{(1)}} & ; \delta_{(k)} \neq \delta_{(1)} \\ 1 & ; \delta_{(k)} = \delta_{(1)} \end{array} \right\}, i = 1, \ldots, k \quad .$$

Then vector $\mathbf{z}$ is assigned to the class for which the sum of the weights is the maximum among the k-nearest neighbors.

Denoeux (1995) recently proposed a classification scheme which integrates the evidence aggregation characteristic of Dempster-Shafer theory and the voting feature of k-nn rules. Let $X_{(k)} = \{\mathbf{x}_{(i)} : i = 1, \ldots, k\}$ be the k-nearest neighbors of z, and suppose that $\mathbf{x}_{(i)}$ comes from class q. The point $\mathbf{x}_{(i)}$ increases our belief that $\mathbf{z}$ could be a member of class q but it does not provide evidence that ensures 100% confidence in this belief. This evidence can be represented by a *basic probability assignment* (bpa). Let C = {1, ..., c} be the index set on the c classes, P(C) be the power set of C, and put: $m_i(\{q\}) = \alpha_i$; and $m_i(C) = 1 - \alpha_i$, where $m_i(A) = 0 \; \forall \; A \in \{P(C) - \{C, \{q\}\}\}$ and $0 < \alpha_i < 1$. A bpa like this can be written for each $\mathbf{x}_{(i)} \in X_{(k)}$, and the k bpa's can then be combined using Dempster's rule to get a composite bpa. If the composite bpa provides the maximum support for class q, then $\mathbf{z}$ is assigned to class q. The success of this scheme depends on the choice of the $\{\alpha_i\}$. Denoeux suggested using

$\alpha_i = \alpha_0 e^{-\gamma_i \delta(\mathbf{x}_{(i)}, \mathbf{z})^\beta}$, $\beta \in \{1, 2, \ldots\}, \gamma_i > 0 \ \forall \ i$, and $0 < \alpha_0 < 1$. According to Denoeux (1995), a good choice for α_0 is $\alpha_0 \approx 0.95$, and he asserts that this algorithm often performs better than the standard crisp k-nn classifier.

There is another family of nn classification algorithms for *univariate data* that uses ranks instead of distances. The *rank nearest neighbor* (rnn) classifier was first proposed by Anderson (1966) for c = 2 classes. Instead of using the distance to z from the training samples the ranks of the training samples are used. Anderson's work was further investigated by Das Gupta and Lin (1980) and later extended by Bagui (1993) to more than two classes. Bagui's algorithm sorts (ranks) the training data along with z and classifies z as follows : (1) if both the immediate left hand (LH) and right hand (RH) neighbors belong to the same class, then classify z to that class; (2) if z is either the smallest or the largest element then classify z to the class of its immediate rnn; (3) if the immediate LH and RH rnn's belong to two different classes then classify z to either class arbitrarily. This algorithm is known as the *1-stage univariate rank nearest neighbor* (1-Urnn) rule. The 1-Urnn rule can be generalized in two ways (Bagui and Pal, 1995) : (1) like the k-nn rule, the 1-Urnn is extended to consider m rnns on either side of z resulting in the *m-stage univariate rank nearest neighbor* (m-Urnn) rule; (2) extension of the m-Urnn rule to multivariate data, i.e., the *m-stage multivariate rank nearest neighbor* (m-Mrnn) rule.

The m-Urnn rule, like the 1-Urnn rule, has three steps. The first two steps of 1-Urnn remain the same for m-Urnn. Step (3) is changed to : (3') If the immediate LH and RH rnn's belong to two different classes check the second LH and RH rnn's. If they belong to the same class assign z to that class; else check the 3rd LH and RH rnn's and so on. The process is continued until we get both qth LH and RH rnn's, q < m are from the same class; otherwise z is arbitrarily assigned to either class of the mth LH or RH rnn.

Although the m-Urnn scheme has interesting theoretical properties, classification rules for univariate data are not very useful for practical applications as we hardly ever encounter real problems for univariate data. To overcome this limitation Bagui and Pal (1995) generalized the m-Urnn rule to multivariate data. The m-Mrnn rule applies the m-Urnn rule to each of the p features. For every feature j, m-Urnn can produce two types of outcomes which can be represented as either (1) $u_{ji} = \begin{Bmatrix} 1 & ; i = s \\ 0 & ; i \neq s \end{Bmatrix}$ (here the m-Urnn rule for the jth feature unambiguously suggests class s); or (2) $u_{ji} = \begin{Bmatrix} 0.5 & ; i = s \\ 0.5 & ; i = r \\ 0.0 & ; i \neq s, r \end{Bmatrix}$ (here the m-Urnn rule at the mth stage for the jth feature suggests either of classes s or r). Then compute $u_i = \sum_{j=1}^{p} u_{ji}$. If $u_q = \underbrace{\max\{u_i\}}_{i}$ is unique, then **z** is assigned to class q; otherwise, **z** is arbitrarily assigned to any class among the set of labels that attains the maximum value.

Fuzzy integrals

We have often used the term "the fuzzy integral". This is a bit misleading because "the fuzzy integral" is a general term that identifies a wide family of functionals which comprise two basic subgroups - the Sugeno and Choquet integrals. There are many variations of fuzzy integrals in each of these subgroups. Keller et al. (1994a) summarize a few of the many variants that have been used in pattern recognition. A new book on fuzzy integrals contains extended discussions about many tools based on this technology that are germane to our field (Grabisch et al., 1999).

Rule-based classifiers

Safavian and Landgrebe (1991) provide a nice survey of many topics connected with decision trees (with the curious and notable exception that they completely ignore the important *interactive* ID3 and C4.5 methodologies of Quinlan (1983, 1986, 1993)), and we have borrowed heavily from their paper for some of the descriptive material in Section 4.6. Jang (1994) and Jang et al. (1997) give a nice account of the use of CART in connection with structural identification in fuzzy systems. We have not been able to find any work on the fuzzification of CART itself, or the use of CART in fuzzy classifier tree design. Some of the fuzzy decision trees discussed in Section 4.6 can be regarded as fuzzifications of C4.5 trees, even though the authors refer to ID3 as the tree building principle used, because C4.5 adds the idea of cutpoints for continuous variables to the basic structure of ID3.

The work of Tani and Sakoda (1992) is often mistakenly cited as having a "fuzzy ID3" method in it, but these authors apply crisp ID3 as we have given it in Section 4.6.C to the system identification problem for a TS system. In this respect Tani and Sakoda is very similar to Jang (1994), which is often quoted as describing a "fuzzy CART" method, but which uses crisp CART to initialize some of the parameters of TS systems.

Lee (1992) gives a straightforward application of Wang and Suen's (1987) fuzzy decision tree classifier to Chinese character recognition data. Lee's example uses 64 apparently different (from Wang and Suen's) features extracted from each character, including mesh counts, crossing counts, contour line lengths, peripheral areas, connective pixels and Fourier descriptors. Lee also provides a comparison of error rates using the 1-nn rule, Chang and Pavlidis's fuzzy decision tree with branch-and-bound-backtracking search, and Wang and Suen's fuzzy decision tree method with pruning and probabilistic similarity matching. In his study the two fuzzy decision trees had recognition rates that were roughly twice as good as the 1-nn rule, with the Wang-Suen design being slightly better than the Chang-Pavlidis classifier.

Yuan and Shaw (1995) discuss a modification of ID3 for categorical data that uses the fuzzy entropy of Deluca and Termini (1972) to measure the vagueness of each linguistic term in a set of given memberships associated with each attribute value in categorical data. They assess the ambiguity of each attribute by averaging a measure built from normalized possibility distributions of memberships for each object due to Hagashi and Klir (1983). This can be done for each attribute value, and for the given possibilistic label vectors attached to the training data. Tree induction follows the standard ID3 design, except that the impurity function for node splitting is based on classification ambiguity instead of probabilistic entropy.

Yuan and Shaw give one example of their technique using a set of n = 16 examples with 4 linguistic variables that collectively possess 8 linguistic values. Their data supposedly have c = 3 crisp target classes, but each training datum is labeled by a possibilistic $\mathbf{u}_i \in N_{pc}$. The data also come mysteriously equipped with a complete set of fuzzy label vectors attached to each of the four attributes for each of the 16 cases. Yang and Shaw apply their method to this data to find a fuzzy decision tree with three levels and 6 pure leaves with crisp labels. They allude to computing path firing strengths for test inputs, but are not specific, saying only that the consequent of rule i is set equal to its premise membership. Running the tree on the training data produces 3 errors in 16 tries - the resubstitution error is nearly 19% on 16 inputs. It's hard to imagine that this version of fuzzy decision tree classification will be useful for real data.

Weber (1992) reports on a version of fuzzy ID3 that can be used with numerical or categorical feature data. Tree induction again follows the standard ID3 design, except that Weber's impurity function for node splitting is based on Deluca and Termini's (1972) fuzzy entropy. Weber conditions the training data by fuzzifying it with probabilities of fuzzy events. In a very similar spirit, Cios and Sztandera (1992) investigate the use of four measures of fuzziness as impurity functions to accelerate the tree building process in their version of ID3 for ordinal data. They report that the best one of the four tried was Kosko's (1992) fuzzy entropy combined with Dombi's generalized fuzzy set operations (see Klir and Yuan, 1995 for a nice discussion of various intersection and union operators, including those of Dombi).

Hall (1996) describes the use of the *crisp* C4.5 decision tree classifier (Quinlan, 1993) for detecting microcalcifications in mammograms.

His work used 40 images from the Nijmegen data base (see Bezdek and Sutton, 1998). The images were paired views of breasts of 21 patients, each of whom had microcalcifications in at least one breast. Each image was 2048×2048 in size with 12 bit intensity values. One or more clusters of microcalcifications in each of the 40 images were marked (circled) by two radiologists (type GT1 ground truth). Thus, each circled area ("cluster") contains pixels corresponding to both microcalcified and normal tissue. He alludes to generalizing this to a fuzzy decision tree, but we are unaware of a follow-up study.

Bensaid et al. (1998) present a straightforward application of Chi and Yan's (1996) fuzzy ID3 method that we discussed in Section 4.9 to the classification of electrocardiogram data. The one new wrinkle added to Chi and Yan's method by these authors is that they use a feed forward cascade correlation neural network (Fahlman and Lebiere, 1990) instead of a multilayered perceptron to perform the "optimized defuzzification" as done by Chi and Yan. Bensaid et al. report that with 53 training data, 53 test data, and 48 validation data, they achieved zero error rates on both the test and validation data sets. They also state that with the same data, a feed forward cascade correlation network commits 7.5% errors on the test set, and 14.6% errors on the validation set. Finally, they state that crisp rules from a decision tree built with ID3 commit 27% errors on the validation data. Bensaid et al. opine that pruning the crisp ID3 decision tree before fuzzification of its rules would likely lead to a smaller tree.

Another decision tree based classifier for real data (crisply labeled, with c classes) is called *real ID3* (RID3) by its creators (Pal et al. 1997, Pal and Chakraborty, 1997). RID3 is a p-level decision tree, where p is the number of features, that uses a ranking of the features. Features are used in the order of their importance. Every internal node has exactly c children. A node in level k is associated with a threshold value and a prototype with the top-ranked k feature values of the prototypes. Of the k components, the first k-1 components are inherited from the parent node. The prototypes are computed as the centroids of the respective classes. Each leaf node represents a crisp class uniquely - that is, RID3 builds pure decision trees.

An unlabeled data point $\mathbf{z}$ starts at the root of the tree, and at each level RID3 tests the similarity of $\mathbf{z}$ to some prototype. This is done by computing membership values using the fuzzy c-means formula in (2.7a), and then assigning the input datum to the clan of the node with the highest membership value, provided the highest membership value is greater than the associated node threshold. Otherwise, input $\mathbf{z}$ traverses but one path in the tree, ending up at a pure, crisply labeled leaf with a high level of agreement. RID3 has an "adaptive" feature in that during the initial tree creation process, all thresholds are set to 0.5, and they are further refined using a genetic algorithm.

When $\mathbf{x} \in \Re^p$ has continuously valued real features, learning the rules of a crisp BDT amounts to finding the constants along each coordinate axes that define hyperplanes that give tree T its crisp rule patches. For example, the hyperplanes in Figure 4.24 have 8 parameters that we could write as a parameter vector $\mathbf{w} = (a,b,c,d,e,f,g,h)^T$. When T is interpreted as a network, $\mathbf{w}$ is a network weight vector as in Section 4.9. These are the parameters we need to acquire by training with the IO data. The goal is to find a $\mathbf{w}$ so that the resultant tree classifier is consistent, $E_{D_{DT,\mathbf{w}}}(X_t|X_t) = 0$.

Cast in these terms, it is not surprising to learn that many authors have given methods for transforming decision trees into various types of neural network classifiers (Cios and Liu, 1992, Sankar and Mammone, 1993, Sethi, 1990, 1995). Sethi (1995) gives a nice introduction to the conversion of soft decision trees into various neural networks. Sethi regards a decision tree that has internal node functions which compute probabilities for the outcome of a test, and inferencing based on Bayes rule at each leaf, as a soft decision tree, and so his results are directly applicable to many types of fuzzy decision trees. In particular, the fuzzy decision tree of Chang and Pavlidis (1997) is a special case of the mathematical structure used by Sethi, so the tree at Figure 4.32 and many of its descendants can be implemented as neural network classifiers.

Sethi argues that the search problem attacked by Chang and Pavlidis and others- vi, that all paths in a soft decision tree must be traversed to their leaves before an input can be labeled - is handily solved by mapping the soft decision tree onto a feed forward neural network. At the very least this seems computationally attractive, since (at least conceptually) all M paths from the root to the leaves can be traversed in parallel in the NN implementation. After this conceptual hurdle is cleared, an additional advantage of this scheme is that the node decision functions $\{\phi_k\}$ at internal vertices in the tree representation become node functions $\{\phi_k\}$ in the hidden layer neurons of the neural network. This paves the way for any of the node functions discussed in Section 4.7.C or elsewhere to become candidates for decision functions in soft decision trees. For example, the hyperplane/sigmoidal combination that is so heavily favored at computing nodes in FFBP networks and many of the methods for training NNs that you know and love become ways to implicitly construct soft decision trees. We say implicitly, because soft decision trees can be converted to multilayered neural networks, but to our knowledge there is no general procedure for the converse operation of finding a soft decision tree that corresponds to a given multilayered network.

Given the equivalence of some types of soft decision trees and certain neural networks, it should come as no surprise that many authors have compared these two styles of classifier design (Weiss and Kapouleas, 1989, Atlas et al., 1990, Shavlik et al., 1991, Chi and Jabri, 1991, Sethi, 1995). For example, Atlas et al. compare classification and regression trees built with CART to multilayered perceptrons. Their opinion is that theoretical evidence to favor trees or NNS is inconclusive, but their computational experiments indicate that multilayered perceptrons are always at least as good as CART built decision trees. We conjecture that this is because CART is at its best in regression analysis, and that C4.5 might have done a lot better. Sethi (1995) shows more: he *transforms* decision trees into NNs and compares various features of the two styles. Cios and Liu (1992) give an ID3 algorithm for ordinal data that converts decision trees into hidden layers in FFBP neural networks. The slant in this paper is a little different - these authors propose using ID3 to determine the structure of the NN. In some of these studies, the neural implementation is identified as a better choice for one reason or another; but in others, the decision tree seems to be the preferred structure. From our knothole, this aspect of classifier design is still to close to call.

With the exception of the work on character recognition (Chang and Pavlidis, 1977, Wang and Suen, 1983, 1984, 1987, Chi and Yan, 1996, Chi et al., 1996a, b) and perhaps, the electrocardiogram data used by Bensaid et al. (1998), none of the papers we are aware of apply fuzzy decision tree classifiers to real or challenging data. Examples with 8 or 12 or 16 cases are nice in a paper, since they are manageable for writing, and useful pedagogically. But we need classifiers to do real work, and tree classifiers have a lot of strikes against them. For one thing, it's hard to postulate a real situation where data of the type used by, say, Yuan and Shaw (1995) or Janikow (1998) provides a natural and easily obtainable representation of a real problem. Then there is the matter of complexity. Trees usually have a lot of leaves (the tree in Figure 4.39 has 5 leaves for n = 8 training examples). For large data sets, the number of leaves (and hence, the number of rules obtained using decision trees to extract them) can be staggering - recall that Wang and Suen (1987) used several thousand leaves in *pruned* decision trees for classification of handwritten characters, and we know of applications of crisp decision trees that discuss leaves on the order of 10^5. Evaluating all paths to the leaves in soft decision trees of this size can take time, compared to, say, computing distances to c nearest prototypes.

You might argue that the conversion of a decision tree to a neural network shows that decision trees are useful, but we would counter that in this case, why not just start with a neural network? Well, maybe there are some good answers to our hypothetical question. First, it's hard to see what the rules are when they are imbedded in a neural network, and the tree structure may provide a useful explanation device for computations made by the neural network (provided the number of rules is pretty small). Second, crisp decision trees don't usually require the excessive training times that are often needed in say, back-propagation style training schemes for neural networks (Section 4.7). Perhaps most importantly, trees provide us with an alternate way to look at a very useful computational structure, and this is always good, because different views can only increase our understanding of the underlying models we choose and use.

Our overall impression of fuzzy decision trees for classification is that they are not really competitive with some of the other techniques discussed in this chapter, and they are not better than their crisp counterparts (especially C4.5) that have evolved within the machine learning community. It is curious, in light of the heavy emphasis placed on pruning trees after building them in the machine learning community, that so little effort has been expended towards this end in the fuzzy decision tree literature.

Function approximation

We have covered four methods for building MA and TS systems from data: decision trees, rule extraction by clustering, neural networks, and heuristic methods. And we have ignored many more methods than we have discussed! Superficially, this field seems important for pattern recognition only if the fuzzy system is a rule-based classification system. Although this criterion would seem to rule out 95% or so of all the papers published about "system identification", the fact is that many of the methods developed to find, for example, a set of rules for a fuzzy controller, can also be used to find a rule-based classifier. This is not always true of course, because of the special nature of the output training data, which in classifier design are almost always crisp label vectors, not observations of (usually) smooth system variables. Moreover, as we have seen in subsection 4.6.F, pattern recognition methods (in this case clustering) are being used as tools to build fuzzy systems for function approximation, and in this respect our interest in this area is entirely appropriate.

Hall and Lande (1998) discuss two methods for extracting fuzzy rules for function approximation with crisp decision trees based on modifications of the strategy developed by Sugeno and Yasukawa (1993). They assume the training data XY has continuously valued inputs and outputs. The output training data set Y set is fuzzified by partitioning it with fuzzy c-means clustering, which provides a fuzzy label vector for each target output. Cluster validation during this step is guided by the validity function of Fukuyama and Sugeno (1989). Unpruned decision trees are generated with Quinlan's C4.5 (Section 4.6.B), and combined with the fuzzified outputs using two strategies: a fuzzy controller generator model that generates fuzzy

CLIPS rules; and a model that uses fuzzy entropy. Two numerical examples are given that compare these two models to each other. One example is based on approximation of the SISO quadratic function $S(x) = x^2 + x + 168$; the authors report a generalization MSE of 2.71% on test sets with 122 samples from the interval [50, 111], and state that a typical design used either 18 or 22 rules. Hall and Lande also discuss rule extraction with the well-known Box-Jenkins (1970) gas furnace data, a perennial favorite for function approximators.

Nie and Lee (1996) extract rules for function approximation from labeled IO data by generating point prototypes in the product set XY (Section 4.6.G). They develop three variations of LVQ (Section 4.3.D), several of which use fuzzy sets, that produce prototypes suitable for implementing rules based on the strategy illustrated in Figure 4.56. The set of prototype vectors $\overline{\mathbf{V}}^{XY}$ shown in Figure 4.56 are replaced by a set $\mathbf{V}^{XY}$ of "rule center vectors" that are found using any of the three variations of LVQ. The prototypes have radii (user-specified in two algorithms, learned in the third) associated with them that allow Nie and Lee to define spherical neighborhoods as shown in Figure 4.56. These authors define five different measures of similarity between an input $\mathbf{x}_k$ and prototype $\mathbf{v}_i^{XY}$, and use these measures as a basis for several thresholds (sum of similarities, maximum similarity) that control the number of prototypes (and hence, number of rules) discovered in XY by their algorithms.

Once an initial rule base is established, Nie and Lee (1996) refine it by merging rules i and j if two rule centers are "close enough", determined by comparing $\left\|\mathbf{v}_i^{XY} - \mathbf{v}_j^{XY}\right\|$ to the radius of the jth neighborhood. Three numerical examples are given to demonstrate the effectiveness of their techniques for function approximation. One example is based on approximation of the SISO function $S(x) = 3e^{-x^2} \cdot \sin(\pi x)$. Nie and Lee construct XY by dividing [-3, 3] into 399 equal length subintervals, and computing S(x) at each of the 400 subinterval endpoints. Then 300 of the 400 points are drawn randomly for training, and the remaining 100 are reserved for testing. In this example all three methods determined 30 rules, with MSE test errors ranging from 0.0068 to 0.0114.

Setnes et al. (1998) use the Gustafson-Kessel (GK, Section 2.2.A) clustering algorithm to identify parameters for a first order Takagi-Sugeno fuzzy system. This work is very similar in spirit to our example 4.17, the main difference being the type of clustering employed. The numerical example given in Setnes et al. is based on approximation of the SISO function $S(x) = 3e^{-x^2} \cdot \sin(\pi x) + \eta$, with a Gaussian noise term $\eta = n(0, 0.15)$ added to the function used by Nie and Lee (1996). Setnes et al. use c = 7 clusters chosen by

Krishnapuram and Freg's (1992) average within cluster distance validity measure. These authors report a resubstitution MSE of 0.0028 on n = 300 training data computed with S(x) for $x \in [-3, 3]$ using only 7 TS rules. This supports our assertion that higher order systems have better approximation capabilities than lower order ones do. And it also shows, in conjunction with our previous discussions of Kim et al. (1997) and Runkler and Bezdek (1998) that the clustering algorithm you choose for rule extraction in 1-st order TS systems need satisfy just one main criterion: it should produce prototypes that naturally generate linear approximations to clusters.

We pointed out in Section 4.6.G that one of the advantages of using clustering algorithms for parametric estimation in fuzzy systems was that non-point prototype clustering algorithms often provide direct estimates of the output functions of low order TS systems (here we include point prototype algorithms such as the GK (Section 2.3.A) and FCE (Section 2.3.B), whose covariance matrices afford estimates of linear clusters). Many rule extraction studies exemplify various ways for using point prototypes obtained by clustering algorithms such as the c-means models, especially when designing MA fuzzy systems. We are not aware of any study that uses *selected* prototypes in the training data (from, for example, edited 1-nn rules or genetic algorithms) as opposed to *extracted* prototypes from the data. However, considering the success of selection in prototype classifier design, we think that this is probably a viable alternative to finding point prototypes by clustering for use in rule extraction problems. Of course, selection can be used this way only for rule extraction methods that use points in the given IO spaces; when you seek lines, planes, curves, etc., extracting non-point prototypes cannot be done by selection.

We want to conclude our comments on this topic by referring you to pages 8-10 in Dubois et al. (1998), titled "*Fuzzy Systems: Modeling versus Explaining*". These three pages contain the most intelligent and thought provoking critique of the use of fuzzy systems for function approximation you will ever read. Dubois et al. point out the inherent conflict between the use of fuzzy sets for function approximation, which aims to mimic and reproduce data accurately, with building an "intelligent" system with fuzzy sets whose intent is to articulate knowledge from data. They argue that fuzzy systems have lost some of their original appeal because they are prized more now as universal approximators to functions, and less valued as a means to build numerical functions from heuristic knowledge, nor as a tool for the linguistic summarization of data. From this viewpoint it might be argued that the only method we have discussed for designing rule based classifiers that fits into the original framework of fuzzy models envisioned by Zadeh is our subsection 4.6.G on heuristic methods of classifier design.

Dubois et al. point out that in the general scheme of things, fuzzy systems are not equipped to compete with well developed methods of function approximation (e.g., Powell, 1990) because they: may not be general enough to capture a wide class of functions, are not very simple because many rules will be needed, are not particularly efficient computationally, and do not generally extrapolate as well as more standard methods. They ask : "why should we bother about if-then rules, and about the 'readability' of fuzzy rules as knowledge chunks if the aim is to build a numerical function that best fits a data set?" And they answer their own question a little later in the same paragraph this way: "It is questionable whether the present trend in fuzzy engineering, that immerses fuzzy logic inside the ***jungle of function approximation*** methods will produce path-breaking results that put fuzzy rule-based systems well over already existing tools". It's hard to add anything to this that is either more elegantly stated, or, in our opinion, more accurate - so we won't try. Instead, we advise you to enter the jungle at your own risk.

Fuzzy neurons and neural networks

It's hard to know what to write here. There are probably 5,000 papers about fuzzy-neuro, neuro-fuzzy, neuro-soft, pseudo genetic - neuro, evolutionary-neuro, quasi adaptive fuzzy neuro, computationally intelligent-generalized-soft-evolutionary-neural-self organizing-adaptive, ... well, you get the idea. The list of classifiers based on some combination of all these technologies just goes on and on. Where will it stop? We don't know. Will it stop at an optimal solution? Perhaps, but only for a very limited class of problems. When will it stop? When the funding dries up. Has the addition of fuzziness to the standard neurons and neural models such as LVQ, ART, FFBP made any of them better classifiers? We are tempted to say "perhaps not", because we believe that, given enough time, almost any of the classifiers discussed in this chapter, and most of those that weren't, will produce pretty similar results on "run-of-the-mill" data sets.

We chose not to enter the terminology fray on this topic in the main body of this chapter because sometimes buzz-wordology distracts you from the main point - does it work? From an engineering point of view, it is foolish to discard a model simply because there may be another one out there that does just as well. If you have a problem to solve, and you find a scheme that solves it that falls within your envelope of development criteria (cost, time, size, etc.) - well, that's good enough. Published papers, in the main, provide system designers with a rich supply of potentially useful tools, and the plethora of "fuzzy NN" architectures are among them.

OK, having devalued the terms "neuro-fuzzy" and "fuzzy neural networks" a bit, and vented our cynicism about the buzzwords of the day, we will make an attempt to at least align you with what *some of* *us* perceive to be the main tracks related to these two terms. Then it will be up to you to decide where to get off the train. Neuro-fuzzy hybridization is done broadly in two ways : neural networks can be equipped with the capability to handle fuzzy information; or fuzzy systems can be augmented by neural networks to enhance their flexibility, speed and adaptability of the fuzzy system. The first set of models may be called *fuzzy-neural-networks* (FNN); whereas the second set may be called *neural-fuzzy-systems* (NFS).

FNNs and NFSs are grouped together in the literature under the popular name *neuro-fuzzy computing.* A neural network may be called fuzzy when either the input signals and/or connection weights and/or the outputs are fuzzy subsets or membership values of fuzzy sets (Lee and Lee, 1975, Buckley and Hayashi, 1994). Usually, fuzzy numbers (represented by triangular functions for ease of computation) are used to model fuzzy signals. The fuzzy neuron of Lee and Lee (1975) allows the excitatory and inhibitory inputs, and the outputs to be fuzzy. In other words, it entertains graded inputs and outputs. Following this, many models of fuzzy neurons besides the ones discussed in Section 4.7 have been proposed (e.g., Gupta and Qi, 1991) and developed (Yamakawa, 1990). Neural networks with fuzzy neurons fall in the category of FNNs as they are capable of processing fuzzy information.

Another kind of FNN does not enhance or change the capability of the NN but makes its implementation more efficient. It is well known that back-propagation learning is very slow and the choice of learning parameters such as the learning rates and momentum factors is an important factor in determining rate of convergence of iterate sequences. If the learning rate is high, the network may oscillate; if it is low, then convergence may be slow. Neuro-fuzzy hybridization can accelerate back-propagation training by an adaptive choice of the learning rate using the fuzzy control paradigm. Let δE be the change in error E and $\Delta_{\delta E}$ be the change in δE. We can use fuzzy rules like the following to adapt the learning rate and the momentum factor (Choi et al., 1992). Haykin (1994) summarizes this procedure in greater detail.

IF	δE is *small*
AND	$\Delta_{\delta E}$ is *small*
AND	sign(E) = constant for *several* iterations
THEN	increase learning rate and momentum a *little.*

In a neural-fuzzy system designed to realize the process of fuzzy reasoning the connection weights of the network correspond to the parameters of fuzzy reasoning (Keller and Tahani, 1992a, b, Keller et al., 1992, Pal et al., 1998). Using back-propagation type learning algorithms, the NFS can identify fuzzy rules and learn membership functions of the fuzzy reasoning system. Usually it is easy to establish a one-to-one correspondence between the network and the fuzzy system. In other words, the NFS architecture has distinct nodes for antecedent clauses, conjunction operators and consequent clauses. There can be, of course, another black-box type NFS where a multi-layer network is used to learn the input-output relation represented by a fuzzy system. For such a system the network structure has no relation to the architecture of the fuzzy reasoning system.

There are many variations of ART and fuzzy ART. Serrano-Gotarredona et al. (1998) discuss hardware implementations of four such architectures : ART1 and ARTMAP are used for clustering and classifier design of binary input data, respectively, while Fuzzy ART (FART) and Fuzzy ARTMAP are used, respectively, for clustering and classifier design with continuously valued input data.

Like any sequential competitive learning model, ART outputs are dependent on the sequence of data feed. Shih et al. (1992) report that in their experiments with optical character recognition, the number of categories identified when the characters E, F and L are fed in different sequences are different. For example, when the sequence is E,L,F then L belongs to the category of E, and F forms a new category; but when the sequence is F,L, and E, three different categories are recognized. Even with adjustment of the vigilance parameter such differences found to continue. For each training pattern, ART1 performs the vigilance test for the winning neuron and if the test fails the next winning neuron is tried. Thus when a quite different pattern comes, ART1 checks the vigilance for all exemplars and finally creates a new node or exemplar. This is time consuming, as correctly pointed out by Shih et al. (1992), who remarked that a threshold may be set up on the activation response so that neurons with lower activation need not be considered. Thus, Shih et al. (1992) proposed an *improved ART* (IART), more specifically an improved version of ART1 which essentially modifies the vigilance testing.

Recall from Section 4.8 that in ART1 the vigilance test assesses the similarity between an input $\mathbf{x}$ and a chosen exemplar $\mathbf{v}_J$ as follows, $\mathbf{v}_J$ is the exemplar vector associated with the winner L_{2J}.

$$\frac{\langle \mathbf{x}, \mathbf{v}_J \rangle}{\|\mathbf{x}\|_1} \geq \rho_1 \quad . \qquad (4.160)$$

In the context of image processing, for example, equation (4.160) represents the percentage of pixels of the input pattern that are present in the exemplar pattern. In addition to (4.160), Shih et al. (1992) suggested using a second criterion for vigilance testing, namely

$$\frac{\langle \mathbf{x}, \mathbf{v}_J \rangle}{\|\mathbf{v}_J\|_1} \geq \rho_2 \quad . \tag{4.161}$$

where $\mathbf{v}_J$ is the exemplar vector associated with the winner node L_{2J} in the output layer. Resonance is assumed to occur only if both (4.160) and (4.161) are satisfied, and then new input is associated with the exemplar and the weights are updated. Thresholds ρ_1 and ρ_2 in (4.160) and (4.161) need not be equal. Except for this additional vigilance test, the rest of IART remains almost the same as ART1.

Shih et al. (1992) used IART for optical character recognition based on square 16×16 input images. With $\rho_1 = \rho_2 = 0.8$, two pairs of characters, {G, O} and {P, R}, were grouped into the same category. Making the vigilance test more strict by choosing $\rho_1 = \rho_2 = 0.9$, they obtained a higher rate of correct classification. Shih et al. also used IART to design a neural architecture for image enhancement.

Newton et al. (1992) proposed an "adaptive" clustering algorithm that combines elements of simple leader clustering, fuzzy c-means, and ART1. Kim and Mitra (1994) discuss a second, improved version of this algorithm which purports to be more precise because it incorporates a different vigilance criterion and a new distance measure. These two algorithms have been used in applications such as vector quantization for low bit rate image coding (Mitra and Yang, 1999) and image segmentation (Mitra et al., 1999).

Training radial basis function (RBF) networks has become a topic de rigeur in the last few years. Many writers that are knowledgeable in the field of approximation theory seem to concede that "the right" RBF network affords wonderful approximations to arbitrarily complex functions. A quick web search against "RBF training" will produce 50-100 papers on this topic, and as we mentioned in Section 4.8.C, this topic could easily consume a chapter of its own. To avoid the embarrassment of not discussing the many fine papers that are available on this topic, we want to mention just one method for training RBF networks that is related to other material in this book.

Medasani and Krishnapuram (1997) offer a modification of the GMD-AO algorithm (Section 2.2.C) that is accomplished by adding a second term to the likelihood function which is similar to that in the competitive agglomeration (CA) algorithm (see equation (2.75)). The role of the second term is to assess the number of components in the Gaussian mixture, so it is essentially a "built-in" cluster validity functional for the GMD model. This algorithm can be used with labeled data to determine the mixture parameters for each class. When the RBFs to be determined are Gaussian as in (4.124), Medasani and Krishnapuram suggest pooling the parameters obtained from clustering each class to initialize the hidden RBF layer (the output layer of the left-half net). The number of RBF nodes is determined automatically via the hidden validity function. These authors then fine tune the RBF parameters of both layers simultaneously using standard gradient descent on the squared error between the observed and target outputs. This approach seems to result in less susceptibility to local trap states than tuning an RBF network that has been randomly initialized.

Classifier Fusion

Keller et al. (1987) provide examples of both temporal fusion and sensor fusion in automatic target recognition utilizing linguistic averaging (Dong et al., 1985). Linguistic averaging is an application of the extension principle to arithmetic functions. They combined the outputs of two classifiers with fuzzy estimates of motion, and fused the results of two classifiers on each of two sensors together in a hierarchical network. The individual estimates of class confidence and consistent motion confidence were represented by triangular fuzzy numbers. Examples included in Keller et al. (1987) demonstrate how such a fusion methodology can (re)acquire objects when they move behind other objects, and that this method also reduces random clutter due to sensor motion. For the sensor fusion examples, it was shown that a human (or some AI-type agent) which was supplying classifier/sensor reliabilities in a non-numeric, i.e., linguistic, sense could be incorporated into the fusion mechanism.

Fuzzy neural networks, such as those presented in Section 4.7, have been used for classifier fusion. Krishnapuram and Lee (1992b) demonstrate how a multilayer network of Type I fuzzy neurons (FANs with multiplicative hybrid operators, Φ_M) can be trained to perform target recognition by combining evidence from FLIR and TV sensor data. These authors show how the trained FANs not only do an excellent job of fusing the information, but the interpretation of the nodes provides added insight into the strength of the sensors and features.

Bloch and Maitre (1995) describe various properties of fuzzy morphology, and show how morphological tools can be used for data fusion and decision making in fuzzy set frameworks. Families of fuzzy mathematical morphology operators for erosion, dilation, opening, and closing are investigated. A thorough comparison of six definitions of morphology and fuzzy sets is presented in terms of logic and decision theory. In Bloch (1996c), a classification scheme is developed for the main operators used in numerical data fusion to combine information from multiple sensors. Three classes of operators based on properties such as decisiveness and ability to handle conflicting information are discussed. Context independent, *constant behavior* operators do *not* consider external information because they exhibit the same behavior for any values of the information to be combined, whereas context independent *variable behavior* operators *do* depend on the values of the variables to be combined (e.g., behaving one way if both values are low). Finally, context dependent operators depend on global knowledge or measures (e.g., reliability) of the sources to be combined. Synthetic examples are provided to demonstrate how probability and Bayesian inference, fuzzy sets, possibility theory, MYCIN-like systems, and Dempster-Shafer evidence theory fit into the proposed classification. Criteria for determining the choice of operator are also described.

Syntactic pattern recognition

Decision tree classifiers and rule-based systems often deal with structural properties of data. Sometimes the connection to data substructure is implicit, as for example in most of the clustering algorithms and classifiers we have discussed. On the other hand, some trees and rule-based systems contain information which allows classification based on specific and explicit structural properties of objects. Examples of this type of classification include chromosome and sometimes character recognition. We will meet other examples of this type in Chapter 5. If you expand the concept of syntactic pattern recognition to include "structural and syntactic" classifiers, for example, (Goos et al., 1996), then both decision trees and rule-based classifiers (Section 4.6) fit in this category. Hall (1973) demonstrated the equivalence of And/Or graphs and context-free grammars. Hence, fuzzy variations of these models arguably belong to the category of fuzzy structural and syntactic systems.

There was a flurry of papers that used fuzzy models in syntactic pattern recognition in the first ten years following Zadeh's 1965 paper - that is, the '70s. This trend mirrored a fairly widespread interest in crisp structural approaches that was evident in the late 1960s and 1970s. After a period of relative quiet in the 1980s and early 1990s, there has been a resurgence of sorts for (crisp and soft) syntactic methods in the late 1990s.

We mentioned in Section 2.1 that the first Ph.D. thesis on fuzzy sets was written by Bill Wee (1967) at Purdue University. His work was directed towards the use of fuzzy automata for applications in syntactic pattern recognition (Wee and Fu, 1969). Another early Ph.D. thesis on fuzzy sets, titled *Fuzzy Languages and their relation to Automata*, was written by E. T. Lee (1972a) at the University of California, Berkeley. Lee produced a number of papers on the use of fuzzy tree automata for classifying chromosomes, leukocytes (cancer cells), handwritten numerals and even applied his method to an early attempt at cataloging images in pictorial databases using fuzzy query languages (Lee and Zadeh, 1969; Lee, 1972b, Lee, 1975, Lee, 1976a, b, Lee, 1977a, b). In view of Lee and Lee (1970), probably the first paper on fuzzy neurons, it is clear that E. T. Lee was another real pioneer in several areas that have enjoyed a lot of growth in the last few decades. Other very early work in this area includes Thomason (1973), Tamura and Tanaka (1973) and DePalma and Yau (1975).

In Pal et al. (1983b), algorithms were developed to define fuzzy memberships of curves in the classes "vertical", "horizontal" and "oblique". These definitions were used in Pal and Bhattacharyya, (1990) to assist the shape encoding process for cell abnormalities. Crisp syntactic pattern recognition was then used to classify the abnormalities. Pathak and Pal (1986) use fuzzy feature extraction for primitive shapes and develop detailed fuzzy grammars and fuzzy fractional grammars (DePalma and Yau, 1975) to recognize eight different levels of skeletal maturity of the wrist from x-ray images.

Parizeau and Plamondon (1995) demonstrate an excellent use of fuzzy syntactic approaches in modeling and classifying allographs for cursive script recognition. They use attribute memberships for primitive encoding, and utilize fuzzy shape grammars (Parizeau et al., 1993) to assist in the recognition of highly uncertain objects (handwritten script). What is impressive about this work is that: (1) the authors have addressed a difficult problem; (2) fuzzy set theory is embedded throughout the model, and (3) they tested the approach on a fairly large database with excellent results. This paper exemplifies the power of the principle of least commitment in complex system design (which, as you have noticed by now, some of us really believe in a lot - others of us prefer the principle of least remitment, but that's another story).

In another interesting application, Senay (1992) used fuzzy grammars to build a command language for an "intelligent" user interface. The concept is that the flexibility of fuzzy sets provides better opportunity to accommodate human variability in command syntax. Kaufmann and Rousseeuw (1990) discuss an algorithm called PAM (partitioning around medoids) - a refinement of their k-medoids algorithm, that is similar to Fu's sgHCM algorithm (Table 4.60).

And finally, if you enjoy reading theorems and proofs in the context of formal language theory, with a just a *hint* of a relationship to fuzzy syntactic pattern recognition, papers that might interest you include Peeva (1991) and Hwang et al. (1998).

5 Image Processing and Computer Vision

5.1 Introduction

Digital image processing is the study of theories, models and algorithms for the manipulation of images (usually by computer). It spans a wide variety of topics such as digitization, histogram manipulation, warping, filtering, segmentation, restoration and compression. Computer vision deals with theories and algorithms for automating the process of visual perception, and involves tasks such as noise removal, smoothing, and sharpening of edges (low-level vision); segmentation of images to isolate object regions, and description of the segmented regions (intermediate-level vision); and finally, interpretation of the scene (high-level vision). Thus, there is much overlap between these two fields. In this chapter, we concentrate on some of the aspects of image processing and computer vision in which a fuzzy approach has had an impact. We begin with some notation and definitions used throughout the chapter.

Let $\mathbf{f}:\Re^p \mapsto \Re^q$ denote a *function* from $\Re^p$ to $\Re^q$. The *domain* and *range* of $\mathbf{f}$ are subsets of $\Re^p$ and $\Re^q$, $\mathcal{D}_\mathbf{f}$ and $\mathcal{R}_\mathbf{f} = \mathbf{f}[\mathcal{D}_\mathbf{f}]$ respectively. The *graph* of $\mathbf{f}$ is $\mathcal{G}_\mathbf{f} = \{(\mathbf{x}, \mathbf{f}(\mathbf{x})) : \mathbf{x} \in \mathcal{D}_\mathbf{f}\} \subset \mathcal{D}_\mathbf{f} \times \mathcal{R}_\mathbf{f}$. For example, let $f:\Re \mapsto \Re^+$ be $f(x) = x^2$. Suppose we restrict $\mathbf{f}$ to $[-1, 1]$, then $\mathcal{D}_f = [-1,1]; \mathcal{R}_f = [0,1]$; and $\mathcal{G}_f = \{(x, x^2) : x \in [-1,1]\}$. Plotting the graph $\mathcal{G}_\mathbf{f} \subset \Re \times \Re^+$ yields the familiar parabolic segment above the interval $[-1,1]$. When $q = 1$, f is not boldface type.

The constructions made next are for images with two spatial dimensions, but most of what we say generalizes to images with N spatial dimensions, and to non-spatial dimensions such as time. Let $IJ = \{(i, j) : i = 1, \ldots, m; j = 1, \ldots, n\} \subset \Re^2$ be a rectangular array or lattice of integers that specify (mn) *spatial locations* (pixel addresses). In what follows, ij may be used as a short form for (i,j). Next let $Q = \{q : q = 0, 1, \ldots, G-1\} \subset \Re$ be the integers from 0 to G -1. G is the set of *quantization (or gray) levels* of some *picture function* $\mathbf{p}:\Re^2 \mapsto \Re^N$. It is commonly assumed that $\mathbf{p}$ is continuously differentiable at least once. Confinement of $\mathbf{p}$ to the lattice IJ (which is done automatically by the digitizing scanner that realizes samples of $\mathbf{p}$) creates the $m \times n$ digital image denoted by $\mathbf{P}_{IJ}$.

Integer N is the number of bands collocated in time that are measured by a sensor. When N=1, $\mathbf{P}_{IJ}$ is a unispectral image, denoted by P_{IJ}; otherwise, it is multispectral, denoted as $\mathbf{P}_{IJ}$. For example, N=1 for gray level images, N = 3 for most *Magnetic Resonance* (MR) and color images; N = 6 for *Coastal Zone Color Scanner* (CZCS) images, and so on. For 6 bit images $G = 2^6 = 64$; for 12 bit images $G = 2^{12} = 4096$, and so on. There are databases with 16 and 24 bit images nowadays, but our presentation and examples are generally confined to 8 bit images, which have 256 intensity levels in them.

We often define a function over the spatial extent of an image that is continuous, differentiable, integrable, etc., and we may want to integrate it, differentiate it, or otherwise manipulate it as if its domain had continuously valued variables. For example, the membership function m_F of a fuzzy region in an image is defined only at each pixel of a spatial region in IJ, so its domain is discrete. However, when we deal with integrals, derivatives, etc., the domain of m_F needs to be a contiguous plane subregion S within IJ. How should we write the sum and integral of m_F over S? When S is discrete, we can legitimately write $S \subset IJ$ and use exact notation for sums such as $\sum_i \sum_j m_F(i,j)$. To avoid notational complication, when S is a region within the boundaries of IJ and we want S to support integration, etc., we will simply write integrals, for example, as $\int\int m_F(u,v)dudv$, with the understanding that integration over all of $\Re^2$ will be zeroed except on S, the domain of positive support for the integrand.

It is worth noting that $\mathbf{P}_{IJ} \subset IJ \times Q^N \subset \mathcal{G}_{\mathbf{p}}$. In words, the digital image is a discrete subset of the graph of the picture function composed of the lattice IJ and the values of **p** on this lattice. More generally, it is advantageous to regard several images derived from $\mathbf{P}_{IJ}$ as subsets of the graph of some function defined on the lattice IJ.

A *window* W_{ij} related to pixel ij in any image is a subset of the lattice IJ. Thus, a window is a collection of addresses with dimensions $m_{\mathbf{W}} \times n_{\mathbf{W}}$, and when m_W and n_W are odd integers, we will assume that W_{ij} is centered at pixel ij. If the spatial location (i, j) of W_{ij} is clear, we may use a single subscript (W_i) for a window centered at (i, j). Rule based systems for image processing often extract feature vectors from W_i. For consistency with previous chapters, we will in this chapter only differentiate the spatial locations of a set of pixels centered at (i, j) by calling them $\{X_i\}$, and as in previous chapters, $\mathbf{x}_i = \{x_{ij}\}$ will denote features extracted from the intensities $\{I(X_i)\}$ at these locations.

There is uncertainty in many aspects of image processing and computer vision. Visual patterns are inherently ambiguous, image features are corrupted and distorted by the acquisition process, object definitions are not always crisp, knowledge about the objects in the scene can be described only in vague terms, and the outputs of low level processes provide vague, conflicting, or erroneous inputs to higher level algorithms. Fuzzy set theory and fuzzy logic are ideally suited for dealing with such uncertainty. For example, consider the following rule of thumb in image filtering (or low-level vision) for smoothing:

IF	a region is *very noisy*
THEN	apply a *large* window-based smoothing operator.

Here, the antecedent clause is vague, and the consequent clause is a fuzzy action that can be described only in imprecise terms. By constructing fuzzy rules in terms of condition-action relations, we can easily represent this type of knowledge.

As another example, consider the task of segmenting an image into object regions. Typically, object boundaries and surfaces need to be described in compact terms for further processing. However, object boundaries are often blurred and distorted due to the imaging process. Moreover, in some cases, object boundaries are truly fuzzy. For example, if we are trying to segment the image of a face, how do we decide where the nose ends and the cheek begins? Crisp segmentation does not preserve uncertainty of this type. An alternative approach is to preserve the uncertainty inherent in the image as long as possible until actual decisions have to be made. In this approach, each object in the image is treated as a fuzzy region represented by a fuzzy set. Such an approach would be consistent with Marr's (1982) *principle of least commitment.*

To perform high-level vision tasks such as image understanding, we need to represent properties and attributes of image regions and spatial relations among regions. Fuzzy rule-based systems are ideally suited for this purpose. For example, in a rule-based outdoor scene understanding system, a typical rule may be:

IF	a region is *rather green* and *highly textured*
AND	the region is *somewhat below* a sky region
THEN	the region contains trees with *high confidence*

Terms such as rather green and high confidence are vague. A similar comment applies to spatial relations such as *somewhat below*. Fuzzy set theory provides a natural mechanism to represent such uncertainty and vagueness effectively. The flexibility and power provided by fuzzy set theory for knowledge representation makes fuzzy rule-based systems very attractive for high-level vision when compared with traditional rule-based systems. Furthermore, rule-based approaches must address the problem of conflict resolution when the preconditions for several (partially) conflicting rules are simultaneously satisfied. There are sophisticated control strategies to solve this problem in traditional systems. In contrast, we have already seen several examples of fuzzy rule-based classifier systems in Chapter 4, where problems such as these are attacked by manipulating certainty factors and/or firing strengths to combine the rules. We will see several new examples of this in Chapter 5.

This chapter is *not* a comprehensive survey of all literature that deals with fuzzy approaches to various aspects of image processing and computer vision. We don't consider, but will try to point you towards, fuzzy approaches to important topics such as compression, restoration and coding. Our goal is to introduce you to several basic and instructive techniques that use fuzzy models to address representative problems in image processing and computer vision. This chapter touches upon: image enhancement, edge detection, edge following, thresholding, segmentation, region labeling, boundary and surface description, fuzzy geometry and properties of fuzzy regions, spatial relations between image regions, perceptual grouping and high-level vision.

5.2 Image Enhancement

The earliest paper on image enhancement with fuzzy sets is due to Pal and King (1981), who discuss extraction of fuzzy properties from gray tone images to be used for contrast intensification. Image enhancement is usually one of the first procedures applied to an image in a computer vision task. According to Gonzalez and Woods (1992), the principal objective of enhancement techniques is to process a given image so that the result is more suitable than the original image for a specific application. Typically, we want the enhancement process to be capable of removing noise, smoothing regions where gray levels do not change significantly, and emphasizing (sharpening) abrupt gray level changes. It is, however, hard to incorporate all these requirements into a single framework, since smoothing a region might destroy a line or an edge, and sharpening might lead to unnecessary noise. A good enhancement process is, therefore, required to be adaptive so that it can process each region differently based on the region properties.

Since fuzzy logic can easily incorporate heuristic knowledge about a specific application in the form of rules, it is ideally suited for building an image enhancement system. This has led to the development of a variety of image enhancement methods based on fuzzy logic. Here we briefly review some of them.

Russo and Ramponi (1992) present an image sharpening method which amplifies large gray level differences and diminishes small gray level differences. Russo (1993), and Russo and Ramponi (1994a, 1994b) propose fuzzy rule-based operators for smoothing, sharpening, and edge detection. They use heuristic knowledge to build rules for each of the operations. For example, the original smoothing operator is based on the following heuristic rules:

IF	a pixel is *darker* than its neighboring pixels
THEN	make it *brighter*
ELSE IF	a pixel is *brighter* than its neighboring pixels
THEN	make it *darker*
ELSE	leave it unchanged

In this basic approach, the gray level differences between a given pixel and its neighbors are inputs, and "gray level increment" is the output variable. Assuming that the gray level range is [0, G-1], simple triangular fuzzy sets, *medium positive* and *medium negative* are defined over the interval [-G+1, G-1] to represent *brighter* and *darker* for the input variables, and triangular numbers *small positive, zero,* and *small negative* are defined over the same domain for the increment specified by the consequents of the rules. The inferred output value is added to the original gray level of the pixel.

The general *fuzzy inference ruled by else-action* (FIRE) paradigm introduced by Russo (1993) will be discussed later in this section. Mancuso *et al.* (1994) propose a fuzzy filter for dynamic range reduction and contrast enhancement using a fuzzy rule based approach. The method is based on Peli and Lim's (1982) algorithm. Peng and Lucke (1994) propose a nonlinear fuzzy filter for image processing. Additive Gaussian noise and non-additive impulse noise are considered. Averaging filters can effectively remove Gaussian noise, and order statistics filters such as the median filter can effectively remove impulse noise. Peng and Lucke use fuzzy logic to combine these two methods.

Law et al. (1996) present a fuzzy-logic-based method for image filtering which controls the orientation and size of a Gaussian kernel. They use the local *gradient* and *straightness* as the input variables for the fuzzy rules that control the kernel. *Gradient* and *straightness* are computed based on the gray levels and gray level differences in a local window. We first describe the computation of the gradient and straightness and then discuss image filtering.

Figure 5.1 depicts the computation of the gradient and straightness for a given center pixel. For each possible direction in the window, a dividing line through a pair of opposing pixels as shown in Figure 5.1 is used to evaluate steepness. The *steepness* is computed as the difference between the average gray level of two regions, one on either side of a dividing line. To settle the issue of several directions having the same steepness, they also measure the reflective *symmetry* of the gray level pattern in the window with respect to a line which is perpendicular to the dividing line that is used for the steepness computation. The value of the gradient is then computed using the fuzzy rules in Table 5.1.

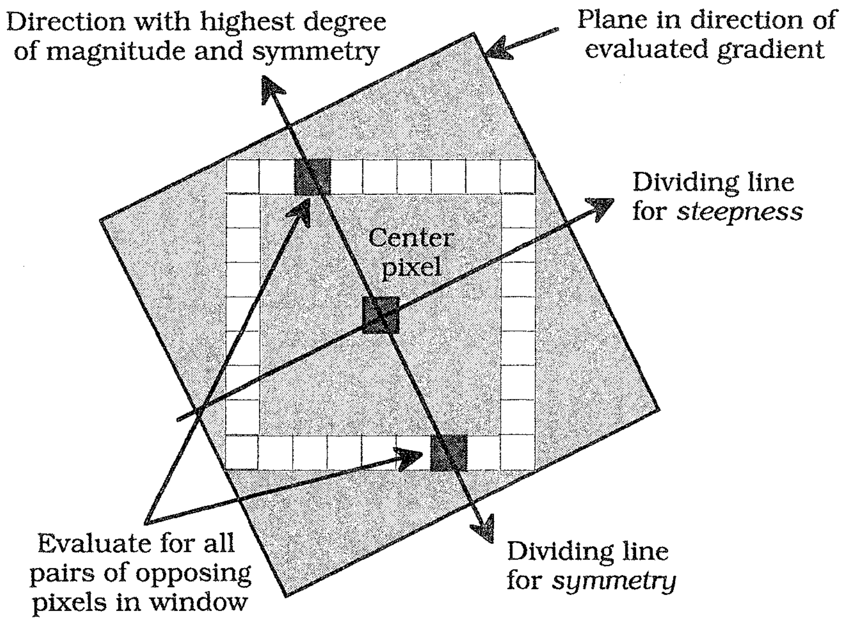

Direction with highest degree of magnitude and symmetry
Plane in direction of evaluated gradient
Dividing line for *steepness*
Center pixel
Evaluate for all pairs of opposing pixels in window
Dividing line for *symmetry*

Figure 5.1 Computation of Law et al.'s gradient

Table 5.1 lists the fuzzy rules used by Law et al. (1996) for determining the gradient. Conceptually, symmetry plays no role when the steepness is small, so an alternative to the first two rules is the simpler rule : IF *steepness* is small THEN gradient is low. The outputs of the pair of rules in Table 5.1 and this single rule may not be exactly the same, but they will be close. On the other hand, there may be some conceptual and implementational advantages to retaining the rules in their original form, because this form has the same number of input variables for each rule.

Table 5.1 Fuzzy rules for determining the gradient

steepness	symmetry	gradient
small	low	low
small	high	low
large	low	medium
large	high	high

The gradient is computed by defuzzifying the output of the fuzzy rules in Table 5.1, and the direction for which it is maximum is taken as the true gradient direction. *Straightness* is determined by comparing pixels translated along the direction of the edge (i. e., perpendicular to the gradient direction). If the edge is straight, the translated pixel should line up with a pixel having a similar value. Evaluating all pixels in the window yields a value for straightness.

Smoothing is done by convolving the image with the Gaussian kernel

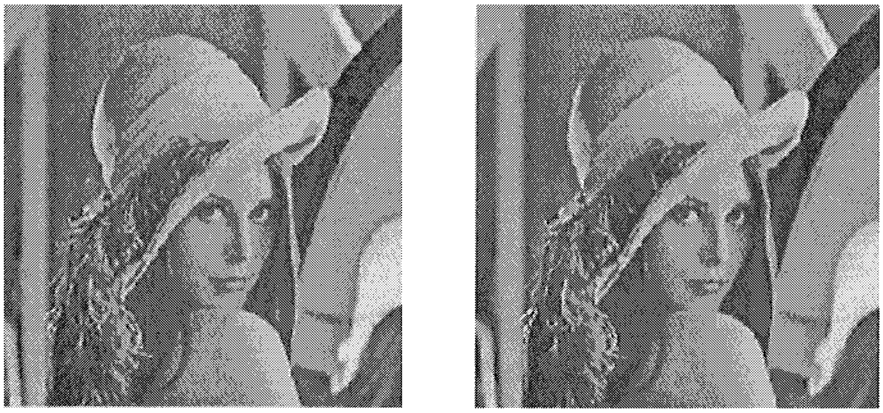
$$W(s,t) = \frac{1}{2\pi}\left(\frac{1}{\sigma_u}\exp\left[-\frac{s^2}{2\sigma_u^2}\right]\right)\left(\frac{1}{\sigma_v}\exp\left[-\frac{t^2}{2\sigma_v^2}\right]\right), \tag{5.1}$$

where u is the edge direction and v is the gradient direction. Since we do not want to smooth out edges and details, the values of σ_u and σ_v are controlled by the fuzzy rules shown in Table 5.2. Membership functions for linguistic labels such as *small* and *large* in Table 5.2 can be found in Law et al. (1996). Again, the first two rules have an easier conceptualization : IF gradient is *small* THEN σ_u and σ_v are *large*. Our remarks about the equivalence of the single rule replacement of the pair of rules in Table 5.1 apply to this case too.

Table 5.2 Fuzzy rules for controlling σ_u and σ_v

gradient	straightness	σ_u	σ_v
small	low	large	large
small	high	large	large
large	low	small	small
large	high	small	large

Example 5.1 Figure 5.2(a) shows the original 256×256 Lena image, ; Figure 5.2(b) shows the Gaussian filtered image using variances σ_u and σ_v of the Gaussian controlled by the fuzzy rules in Table 5.2. The filter size was 7×7 and the range for σ_u and σ_v was 0.1 to 5.25.



(a) Original image **(b) Gaussian kernel**

Figure 5.2 Lena filtered with a Gaussian kernel

Image enhancement almost always means replacing the gray-level value of every pixel in an image with a new value depending on some type of local information. If the intensities in the vicinity of a pixel are relatively smooth, then the new value may be taken as a (possibly weighted) average of the local values. On the other hand, if the local region contains edge or noise points, a different type of filtering should be used. This gives rise to a conditional and adaptive smoothing technique. In other words, we could create a bank of filters, and one of them could be selected at each pixel depending on the local information. However, if a different enhancement algorithm is selected at each pixel, the result may not be pleasing or useful. Moreover, in many cases the selection criteria for the filter can be expressed only in imprecise or vague terms. To overcome these problems, Choi and Krishnapuram (1995, 1997) use a fuzzy logic approach. The filter selection criteria constitute the antecedent clauses of the fuzzy rules, and the corresponding filters constitute the consequent clauses of the fuzzy rules.

Choi and Krishnapuram adopt the FIRE paradigm of Russo (1993) for image enhancement. The rule base $\mathcal{R} = \{R_1,\ldots,R_{M+1}\}$ consists of M+1 fuzzy rules. This system is specialized to image enhancement, and is set up as follows. Let $X = (X_1,\ldots,X_N)$ denote the spatial locations of N pixels in a window W_i within an image that has odd side lengths with center at pixel X_i, and let $\mathbf{I}(X) = (I(X_1),\ldots, I(X_N))^T$ be the vector of gray levels at these spatial locations. The gray level $I(X_i)$ is to be replaced. The LHS of rule R_j has for its input a vector $\mathbf{x}_j \in \Re^{p_j}$ of features such as gradient, steepness, smoothness, symmetry, etc. which is extracted from W_i, and in the general case, different rules can have different numbers of input variables. The rule base, written in the form of equation (4.74), is:

$$\begin{aligned}
&R_1\text{: IF } \alpha_1(\mathbf{x}_{p_1}) \text{ THEN } I_1(X_i) = F_1(\mathbf{I}(X)) \\
&\vdots \\
&R_j\text{: IF } \alpha_j(\mathbf{x}_{p_j}) \text{ THEN } I_j(X_i) = F_j(\mathbf{I}(X)) \\
&\vdots \\
&R_M\text{: IF } \alpha_M(\mathbf{x}_{p_M}) \text{ THEN } I_M(X_i) = F_M(\mathbf{I}(X)) \\
&R_{M+1}\text{: ELSE } I_{M+1}(X_i) = F_{M+1}(\mathbf{I}(X))
\end{aligned} \quad , \tag{5.2}$$

F_j is the output function of the j-th *filter* and p_j is the number of input variables in the j-th rule. Equations (5.2) have the appearance of a TS system, but there are two major differences. First, the dimension of the input vector $\mathbf{x}_{p_j} \in \Re^{p_j}$ for rule j is not necessarily fixed across the rules - instead, there might be a different number of inputs for different rules in $\mathcal{R}$. Second, the argument of each RHS output function is not the same as the argument of the LHS, although both arguments are functions of the intensities in W_i.

As in Chapter 4, $\alpha_j(\mathbf{x}_j)$ denotes the firing strength of rule j. The defuzzified output of (5.2) is computed using either the TS or MA style defuzzification. MA outputs are some function of the firing strengths, consequent values, rule composition and defuzzification operator (see Figure 4.29),

$$I(X_i) = \Theta(\boldsymbol{\alpha}(\mathbf{x}), \{I_j(X_i)\}, \cup, \mathcal{D}_F) \quad \text{, where} \tag{5.3}$$

the else clause firing strength satisfies the constraint

$$\alpha_{M+1} = 1 - \max_{j \in \{1,...,M\}} \left\{\alpha_j(\mathbf{x})\right\} \quad . \tag{5.4}$$

For a particular set of choices in (5.3) we can get the standard TS output form, now used for image enhancement:

$$I(X_i) = \sum_{k=1}^{M+1} [\alpha_k(\mathbf{x}_k) \cdot I_k(X_i)] \Big/ \sum_{k=1}^{M+1} \alpha_k(\mathbf{x}_k) \quad . \tag{5.5}$$

As pointed out in Chapter 4, (5.5) can be interpreted either as the output of a 0-th order TS model; or as the output of an MA model using the height defuzzification method with $I_k(X_i) = F_k$ as the peak value of the consequent membership function.

Choi and Krishnapuram (1997) discuss a form of enhancement that is based on the estimation of a prototypical intensity for a given set of gray levels. Again let $X = (X_1, \ldots, X_N)$ denote the spatial locations of N pixels in a window W_i within an image that has odd side lengths with center at pixel X_i, and let $\mathbf{I}(X) = (I(X_1), \ldots, I(X_N))^T$ be the vector of their gray levels. The gray level $I(X_i)$ is to be replaced.

First we consider replacement of $I(X_i)$ by an estimate obtained by minimizing the objective function

$$J_A(\mathbf{u}_i, X; I(X_i)) = \sum_{\substack{j=1 \\ i \neq j}}^{N} u_{ji}\left(I(X_i) - I(X_j)\right)^2 \quad . \tag{5.6}$$

In (5.6) $\mathbf{u}_i = (u_{1i}, \ldots, u_{Ni})^T \in N_{pN}$ is *specified by the user*, and minimization is done with respect to $I(X_i)$. Here we interpret u_{ji} as the degree (or possibility) to which intensity $I(X_i)$ represents $I(X_j)$. If we assume that the membership function underlying the values $\{u_{ji}\}$ is bell-shaped and centered at $I(X_i)$, and that the membership values depend on gray level differences and spatial distances between the X_i and its neighbors, then a possible choice for the membership function that generates the $\{u_{ji}\}$ is

$$u_{ji} = \omega_{ji} \cdot \exp\left\{-\frac{1}{\beta_i}\left(I(X_i) - I(X_j)\right)^2\right\}, 1 \le j \le N \qquad \text{, where} \tag{5.7}$$

$$\omega_{ji} = \exp\left\{-\frac{1}{\alpha}\left\|X_i - X_j\right\|^2\right\} \qquad . \tag{5.8}$$

In (5.8) $\left\|X_i - X_j\right\|$ represents the Euclidean distance between (centers of) the pixel locations X_i and X_j. The authors state that for small windows, the effect of ω_{ji} on u_{ji} in (5.8) is negligible, and recommend setting it equal to 1 for all j. In this case, the selection of a value for α is moot, but more generally, this parameter would deserve some attention. The parameter β_i is a scale parameter which can be determined on the basis of the variations in pixel intensity values in a given spatial window. This is discussed later.

Differentiating J_A in (5.6) with respect to $I(X_i)$ and setting the result equal to zero leads to *filter A*, an update rule for the intensity of pixel X_i :

$$I_A(X_i) = \sum_{\substack{j=1 \\ j \ne i}}^{N} u_{ji}\left(1 - \frac{d_{ji}^2}{\beta_j}\right) I(X_j) \Bigg/ \sum_{\substack{j=1 \\ j \ne i}}^{N} u_{ji}\left(1 - \frac{d_{ji}^2}{\beta_j}\right) \qquad , \tag{5.9}$$

where $d_{ji}^2 = \left(I(X_i) - I(X_j)\right)^2$. If d_{ji}^2 is larger than β_i, then the weight for pixel X_j will be negative. If d_{ji}^2 is equal to β_i, then the weight for pixel X_j will be zero. Otherwise, the weight for pixel X_j will be positive. The negative weight has the effect of sharpening an edge.

Filter A in (5.9) assumes that $I(X_i)$ is a prototype for its neighboring pixels. If $I(X_i)$ is noisy (that is, if $I(X_i)$ is quite different from the intensities in W_i), this assumption fails, and we need a different updating scheme. When $I(X_i)$ is noisy, we can update the gray level of the center pixel in such a way that the new value maximizes the degrees of membership to which its neighbors represent the center pixel. In other words, we would like to maximize $u_{1i} \otimes \cdots \otimes u_{Ni}$, where

$\otimes$ ="and" is an aggregation operator, i.e., any T-norm or weighted mean.

Different objective functions arise from different choices for $\otimes$. Here we illustrate two methods: multiplication (the standard T_2 norm) and arithmetic averaging (the unweighted mean). Choosing multiplication and continuing to use (5.7) and (5.8), we have

$$J_B(\mathbf{u}_i, X; I(X_i)) = \prod_{j=1}^{N} u_{ji} = \prod_{j=1}^{N} \omega_{ji} \exp\left\{\frac{-d_{ji}^2}{\beta_j}\right\} \quad . \tag{5.10}$$

Setting the first derivative of J_B with respect to $I(X_i)$ equal to zero leads to *filter B*, an update rule for *noisy* center pixel X_i:

$$I_B(X_i) = \sum_{j=1, j\neq i}^{N} \left(\frac{I(X_j)}{\beta_j}\right) \Bigg/ \sum_{j=1, j\neq i}^{N} \left(\frac{1}{\beta_j}\right) \quad . \tag{5.11}$$

If we choose the averaging operator for $\otimes$, the objective function becomes

$$J_C(\mathbf{u}_i, X; I(X_i)) = \frac{1}{N} \sum_{j=1}^{N} u_{ji} = \frac{1}{N} \sum_{j=1}^{N} \omega_{ji} \exp\left\{\frac{-d_{ji}^2}{\beta_j}\right\} \quad , \tag{5.12}$$

and the corresponding update rule defining *filter C* is

$$I_C(X_i) = \sum_{j=1, j\neq i}^{N} \left(\frac{u_{ji} I(X_j)}{\beta_j}\right) \Bigg/ \sum_{j=1, j\neq i}^{N} \left(\frac{u_{ji}}{\beta_j}\right) \quad . \tag{5.13}$$

Since β_i is a scale parameter, it should reflect the variance of the gray level differences between the center pixel and its neighboring pixels. We can simply take the mean of d_{ji}^2 in the neighborhood as β_i, i. e.,

$$\beta_i = \frac{1}{N-1} \sum_{j=1, j\neq i}^{N} d_{ji}^2 \quad . \tag{5.14}$$

The examples shown in this chapter use the estimate at (5.14) for β_i. However, the mean value is sensitive to outliers (impulse noise). A more robust estimate, such as the *median of absolute deviations*, or MAD (Rousseeuw and Leroy, 1987), can also be used.

We need conditions under which each of the update equations (5.9), (5.11), and (5.13) should be used. If a given center pixel is an impulse noise pixel, the degree to which its neighboring pixels can represent this center pixel will be small. If the center pixel is in a homogeneous region, the degree to which its neighboring pixels represent the center pixel will be large. Again using (5.7) and (5.8) as a basis for the filter design, we define the *total compatibility* of X_i with its neighbors in X as

$$\mu_{TC}(X, \mathbf{I}(X)) = \sum_{j=1, j \neq i}^{N} u_{ji} \Big/ \sum_{j=1, j \neq i}^{N} \omega_{ji} \quad . \tag{5.15}$$

The value of μ_{TC} in (5.15) is a measure of the homogeneity of the intensities of the pixels in X, or the degree to which the neighboring pixels represent a center pixel. The smoothest case occurs when all the intensities in X are equal, and then $\mu_{TC}(X, \mathbf{I}(X)) = 1$. The most impulsive case is when $I(X_i)$ is 0 or Q-1 and the remaining N-1 intensities are Q-1 or 0, respectively. Then μ_{TC} takes its minimum value for any choice of α and β_i; and for the small window choice advocated by Choi and Krishnapuram (with β_i given by (5.14) and $\omega_{ji} = 1 \, \forall \, j$), $\mu_{TC} = e^{-1}$.

We can use (5.15) to build the following set of fuzzy rules for image enhancement:

$$\begin{aligned} &R_1 : \text{IF } \mu_{TC} \text{ is } \textit{small}, \text{ then } F_1(\mathbf{I}(X)) = I_B(X_i) \text{ (Impulse noise removal)} \\ &R_2 : \text{IF } \mu_{TC} \text{ is } \textit{large}, \text{ then } F_2(\mathbf{I}(X)) = I_C(X_i) \text{ (smoothing)} \\ &R_3 : \text{ELSE } F_3(\mathbf{I}(X)) = I_A(X_i) \text{ (edge sharpening)} \end{aligned} \tag{5.16}$$

Membership functions for the linguistic values *small* and *large* are defined on the range of μ_{TC}, which is a subset of (0, 1]. The final value for the intensity of each pixel is computed using (5.5).

Example 5.2 We show some examples of Choi and Krishnapuram's approach to image enhancement with fuzzy rules. For small windows, the effect of ω_{ji} in (5.8) is negligible. To reduce computation time, all of the ω_{ji}'s were set equal to 1 in (5.15). Figure 5.4(a) shows the original Lena image. Membership functions for the linguistic values small and large are shown in Figure 5.3.

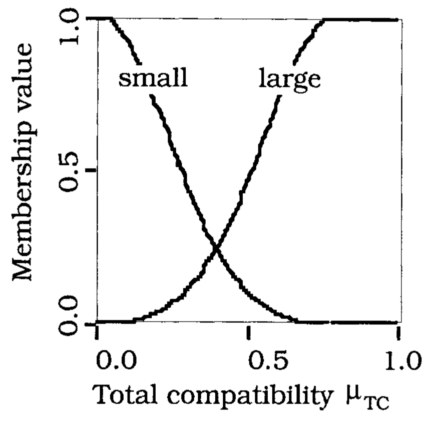


Figure 5.3 Total compatibility of center pixel with neighbors

To create the noisy image shown in Figure 5.4(b), samples from the mixture distribution $0.95n(0, 25) + 0.05n(0, 10000)$ were added to the original intensities (see equations (2.17) and (2.18)). Values < 0 and $>$ 255 can easily occur, and were set to 0 and 255, respectively. The first noise component represents a zero-mean Gaussian with $\sigma=5$, and the second component represents a zero-mean Gaussian with $\sigma=100$. The second component approximates impulse noise because values from this component will almost always saturate the affected pixel intensity.

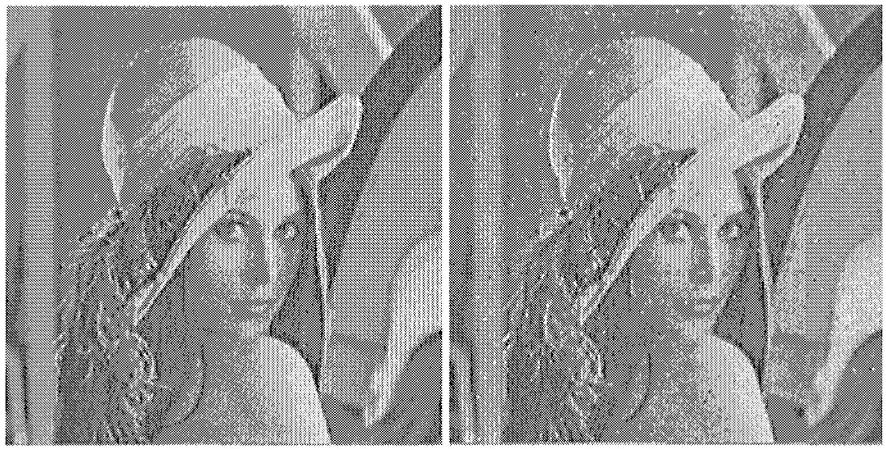
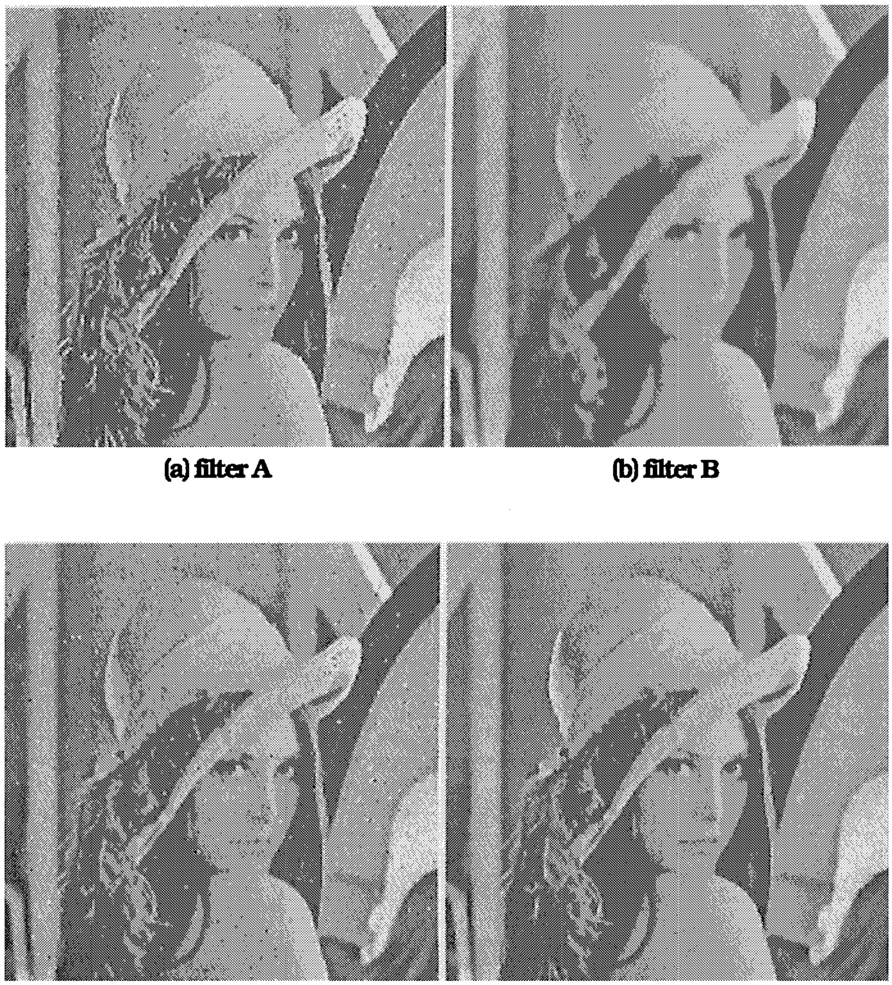

(a) Original Lena **(b) Noisy Lena**

Figure 5.4 Lena and her noisy derivative for Example 5.2

Figures 5.5(a)-(c) show the images produced by applying filters A, B and C to noisy Lena. Using any of these filters alone on the entire image is not particularly effective at noise removal. Figure 5.5(d) shows the result of applying a crisp version of the filtering system in (5.16) to noisy Lena, whereby only one of the filters A, B or C was used at each pixel, whichever had the largest firing strength α_k of the antecedent clause (see (5.4)). Certainly it's the best image in this set of four views.

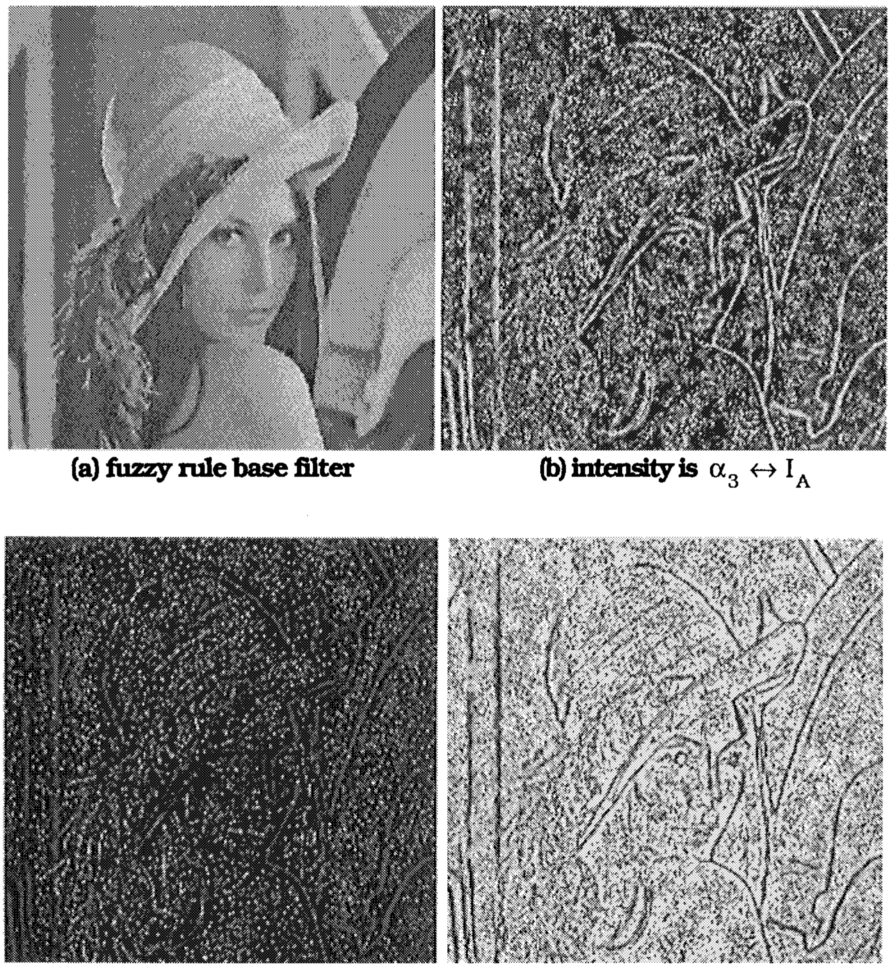


(a) filter A **(b) filter B**

(c) filter C **(d) either A or B or C**

Figure 5.5 Enhancement of noisy Lena by various filters

Figure 5.6(a) shows the result of applying the fuzzy rule-based filtering system to noisy Lena, which does a better job of removing noise without smoothing out the details.



(a) fuzzy rule base filter **(b) intensity is** $\alpha_3 \leftrightarrow I_A$

(c) intensity is $\alpha_1 \leftrightarrow I_B$ **(d) intensity is** $\alpha_2 \leftrightarrow I_C$

Figure 5.6 Fuzzy rule-based filtering of noisy Lena

Figures 5.6(b)-(d) are images whose intensities are proportional to the firing strengths of the three rules : $\alpha_1 \leftrightarrow I_B$, $\alpha_2 \leftrightarrow I_C$, $\alpha_3 \leftrightarrow I_A$ for filters A, B, and C in the fuzzy rule-based filtering system. The weights are scaled by 255 and therefore, a brighter value indicates a larger weight. Filter A has a small weight in most of the image except in edge regions. Filter B has a large weight in locations contaminated by impulse noise. Filter C has a large weight in most regions and a small weight in edge regions.

For comparison, Figure 5.7(a) shows the result of the 5×5 median filter and Figure 5.7(b) shows the result of the Saint Marc filter (Saint Marc et al. 1991) applied to noisy Lena. The parameters of the Saint Marc filter are chosen adaptively during execution.

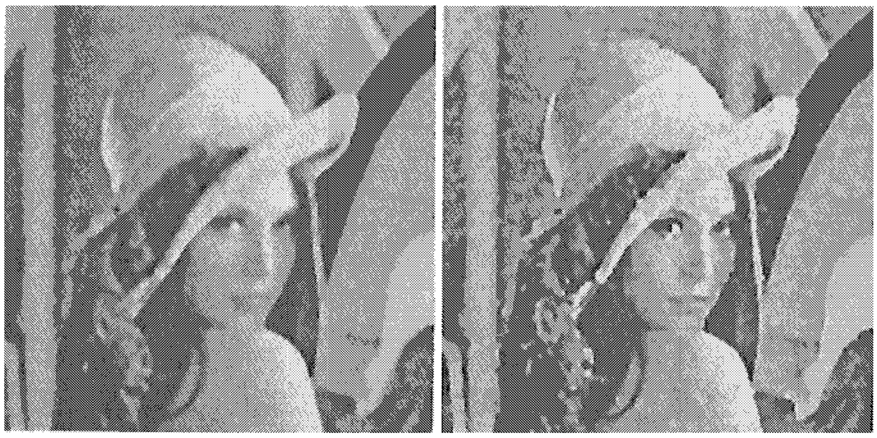


Figure 5.7 Two other well known filters applied to noisy Lena

Table 5.3 shows that the overall *root-mean-squared* (RMS) error between the intensities of the original Lena and the intensities enhanced by the fuzzy filtering system is smaller than that of the crisp version rule-based enhancement in Figure 5.5(d).

Table 5.3 RMS errors produced by various filtering schemes

	Filter A	Filter B	Filter C	Crisp RB	Fuzzy RB
Size	5x5	5x5	5x5	5x5	5x5
Error	14.71	12.44	11.05	7.93	6.94

The problem with enhancement is that, like many other forms of image processing, quantitative assessment of an individual filter is unavailable. Performance analysis of these schemes is ultimately subjective. Which filter does the best job? You make the call.

5.3 Edge Detection and Edge Enhancement

Edge detection is a critical part of many computer vision systems. Ideally, edges correspond to object boundaries, and therefore edge detection provides a means of segmenting the image into meaningful regions. However, the definition of what constitutes an edge is rather vague, heuristic, and even subjective. Jain et al. (1995) say this: an *edge point* locates a pixel where there is "significant local intensity change"; an *edge fragment* is a collection of edge points; and an *edge detector* produces either a set of edge points or edge fragments.

What is implicitly crisp in these definitions of edges? That either a pixel is an edge point, or is not [and therefore, the intensity is either black or white]? Regardless of what you read into these definitions, it is clear that there are several opportunities to fuzzify the notion of an edge, because *two* variables are involved : spatial location and intensity. Our view is flexible: some edge detectors crisply locate pixels that are edge points; others use fuzzy sets to describe spatial locations of edges - sometimes these are called *fuzzy edge points*. Independently, we can regard the *strength* of an edge point (spatially located crisply) as crisp (black or white) or fuzzy (shades of gray); in the latter case, the interpretation of the non-crisp intensity is usually referred to as *fuzzy edge strength*. Some writers describe crisp pixel locations with fuzzy (or otherwise non-two-tone) intensities at "edges" as edge enhancement, reserving the term edge detection for crisp edge points with one of two intensities. In any case, there are many fuzzy models that attempt to locate regions in images that are related to edges, and this section discusses a very few of them.

Russo and Ramponi (1994b) describe an edge detector that is relatively immune to noise, based on the if-then-else FIRE paradigm discussed in Section 5.2. They use the gray level differences in a 3×3 neighborhood as inputs to the fuzzy rules. Let X_i denote the center pixel in the window, and let $x_j = I(X_i)-I(X_j)$, for j=1,...8. Figure 5.8 depicts Russo and Ramponi's numbering scheme for $X_i = 1,...,8$.

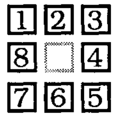


Figure 5.8 The window at center pixel X_i used by Russo and Ramponi

Russo and Ramponi use x_j, j=1,..., 8, as the variables in the antecedent clauses of the rule base in (5.2). As usual, firing strength α_i for rule i is given by (remember that in (5.2) p_i is the number of inputs to rule i)

$$R_i:\ \alpha_i(\mathbf{x}) = \otimes(\mathbf{m}^i(\mathbf{x})) = m^i_{1k_1}(x_1)\otimes\cdots\otimes m^i_{p_ik_{p_i}}(x_{p_i}) \ . \tag{5.17}$$

Russo and Ramponi use the mean operator for ⊗, which gives,

$$\alpha_i(\mathbf{x}) = \frac{1}{p_i}\left(\sum_{j=1}^{p_i} m^i_{jp_j}(x_j)\right) \quad . \tag{5.18}$$

Let $\alpha_T(\mathbf{x})$ denote the overall firing strength of the first M rules and $\alpha_E(\mathbf{x}) = \alpha_{M+1}(\mathbf{x})$ be the firing strength of the ELSE rule. From (5.4),

$$\alpha_T(\mathbf{x}) = \max_{m \in \{1,\ldots,M\}}(\alpha_m(\mathbf{x})) \quad ; \alpha_E(\mathbf{x}) = \alpha_{M+1}(\mathbf{x}) = 1 - \alpha_T(\mathbf{x}) \ . \tag{5.19}$$

The output $I(X_i)$ is obtained by adding the effects of the THEN and ELSE actions and then performing a suitable defuzzification - for example, the one shown in (5.5).

Example 5.3 R_8 is shown graphically in Figure 5.9. Russo and Ramponi's edge detector uses 4 rules; for k = 2, 4, 6 and 8 rule R_k is:

IF	x_k is zero
AND	$x_{k\text{mod } 8+2}$ is zero
THEN	$I(X_i)$ is white
ELSE	$I(X_i)$ is black.

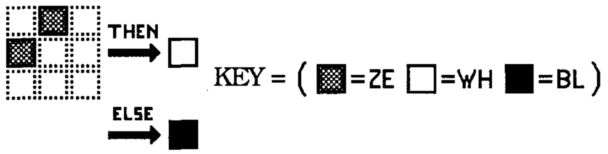


Figure 5.9 Representation of Russo-Ramponi rule R_8

The membership functions for *zero*, *white* and *black* are shown in Figure 5.10 for an image with intensity values between 0 and G - 1.

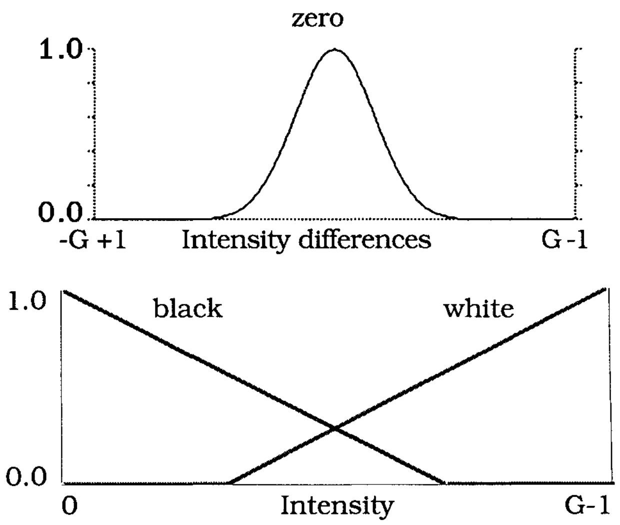


Figure 5.10 PMFs and CMFs for Russo and Ramponi's edge operator

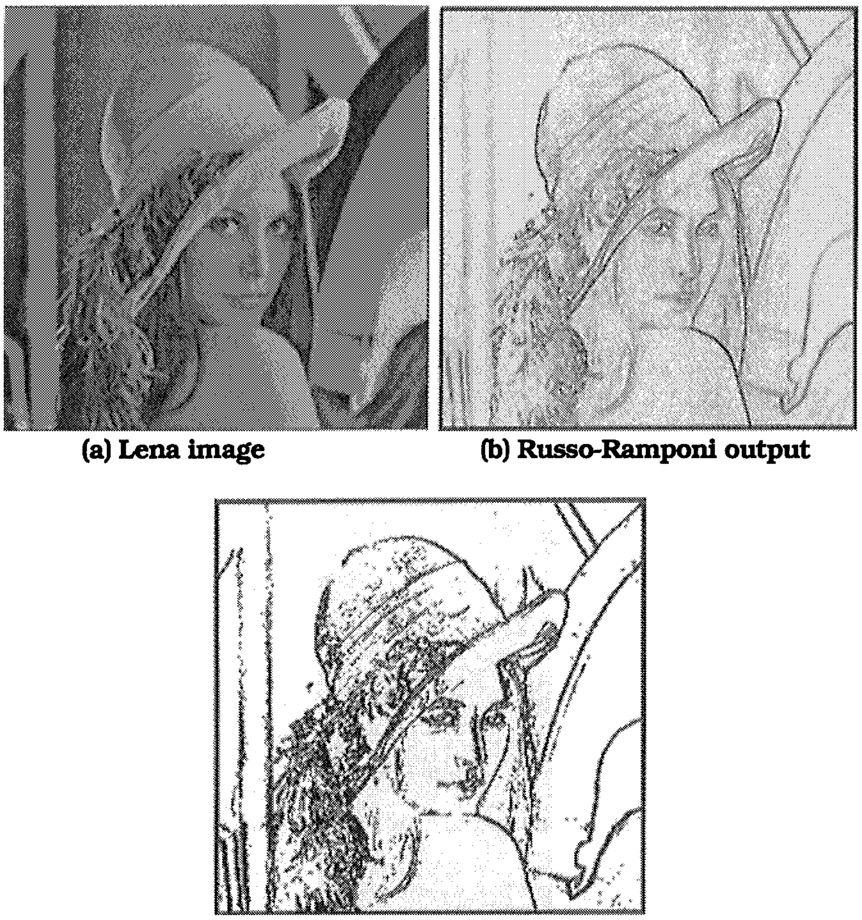


Figure 5.11 Fuzzy rule-based edge detection (Russo and Ramponi)

Figure 5.11 shows a typical result obtained by this edge detector. View (a) is the original input image. View (b) shows the output of the Russo-Ramponi edge detector, and view (c) is a thresholded version of the image in panel (b).

Bezdek et al. (1998a) also use a 4 rule fuzzy system for edge detection and enhancement, but their approach is based on the TS model. They describe edge detection as a composition of four operations. Specifically, they denote the edge image as $E_{IJ} = e[P_{IJ}]$ so that $e = s \circ b \circ \mathbf{f} \circ c$. The function $e: P_{IJ} \mapsto IJ \times Q$ is the *edge operator*. The four functions comprising e are :

$c: IJ \times Q \mapsto IJ \times \Re$: $c[P_{IJ}] = C_{IJ}$ which *conditions* (enhances) the raw data in P_{IJ};

$\mathbf{f}: IJ \times \Re \mapsto IJ \times \Re^p$: $\mathbf{f}[C_{IJ}] = \mathbf{F}_{IJ}$, which extracts geometrically relevant *feature vectors* in $\Re^p$ from C_{IJ};

$b: IJ \times \Re^p \mapsto IJ \times \Re$: $b[\mathbf{F}_{IJ}] = B_{IJ}$ which *blends* or aggregates the components of feature vectors in F_{IJ}; and

$s: IJ \times \Re \mapsto IJ \times Q$: $s[B_{IJ}] = E_{IJ}$, which *scales* blended image B_{IJ} to get gray levels in Q.

Figure 5.12 defines the correspondence used in Bezdek et al. between a 3×3 window W_i centered at spatial location X_i and a sequentially labeled *window vector* $\mathbf{w}_i = (I(X_1), \ldots, I(X_9))^T$ of the intensities at the locations in it. The center address in this window is X_i (instead of X_{ij}) for simplicity, and it occupies position 5.

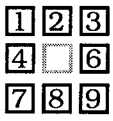


Figure 5.12 The neighborhood of center pixel X_i used by Bezdek et al.

Feature extraction functions (**f**) estimate indicators of geometric behavior at edges. The most common choice is an **f** that approximates the gradient of the picture function. The Sobel features (Gonzalez and Woods, 1992) do this for 3×3 window vectors $\mathbf{w}_i$. Letting h, v stand for the horizontal and vertical spatial directions, the (absolute value) of the Sobel features are :

$$\mathbf{f}_{|S|}(\mathbf{w}_i) = \left(\left|f_{Sh}(\mathbf{w}_i)\right|, \left|f_{Sv}(\mathbf{w}_i)\right|\right) \quad ; \text{where} \tag{5.20a}$$

$$f_{Sh}(\mathbf{w}_i) = (I(X_9) + 2I(X_8) + I(X_7)) - (I(X_1) + 2I(X_2) + I(X_3)); \tag{5.20b}$$

$$f_{Sv}(\mathbf{w}_i) = (I(X_3) + 2I(X_6) + I(X_9)) - (I(X_1) + 2I(X_4) + I(X_7)). \tag{5.20c}$$

The *range* (r) and *standard deviation* (s) of the intensities in $\mathbf{w}_i$ are surprisingly good edge features. The geometric relationship of r to edges is clear : it is an order statistic that measures the maximum distortion among the intensities in $\mathbf{w}_i$. The standard deviation measures how much variation occurs in intensities in $\mathbf{w}_i$. The formulae for (r, s) on a 3×3 window are:

$$\mathbf{f}_{rs}(\mathbf{w}_i) = (f_r(\mathbf{w}_i), f_s(\mathbf{w}_i)) \qquad \text{; where} \qquad (5.21a)$$

$$f_r(\mathbf{w}_i) = \underbrace{\max}_{i=1,\ldots,9}\{I(X_i)\} - \underbrace{\min}_{i=1,\ldots,9}\{I(X_i)\} \qquad \text{; and} \qquad (5.21b)$$

$$f_s(\mathbf{w}_i) = \sqrt{\left(\sum_{i=1}^{9} I(X_i)^2 / 9\right) - \left(\sum_{i=1}^{9} I(X_i) / 9\right)^2} \qquad . \qquad (5.21c)$$

Equation (5.21c) is a biased estimate of the variance because n is used in the denominator. Readers who prefer an unbiased estimate should use n-1 instead. Nice things about these features are that their functional forms are known for any size window, that they are invariant to changes in window indexing, and they are not orthogonal (like the Sobel features), so they are not biased towards finding only edges parallel to the axes of the spatial grid. Bezdek et al. give some examples of processing images for edges using all four of the features in (5.20) and (5.21).

Blending functions (b) aggregate information about edges possessed by the features. There are many types of blending functions. Of these, Bezdek et al. discuss three parametric families: (i) *norms*, of which the two most common families are the inner product and Minkowski norms; (ii) *generalized logistic functions*; and (iii) *computational learning models* such as neural networks and fuzzy systems. For convenience, let $\mathbf{x} = \mathbf{f}(\mathbf{w}_i)$ and $b_{\|*\|}(\mathbf{x}) = \|\mathbf{x}\|$, where $\|*\|$ is any norm on $\Re^p$. The Euclidean norm of $\mathbf{f}(\mathbf{w}_i)$ is often regarded as the standard blending function, but there is no reason *a priori* to prefer the Euclidean norm of, for example, the Sobel features $\mathbf{f}_S(\mathbf{w}_i)$ as the best way to make Sobel edge images. *ANY* norm can be used to combine the components of $\mathbf{f}(\mathbf{w}_i)$.

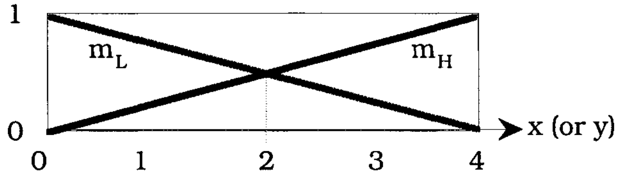


Figure 5.13 Membership functions for b_{TS4} (same for x and y)

The blending function discussed here is an analytic realization of a two input, one output TS model which Bezdek et al. defined subjectively (as opposed to training with IO data). Figure 5.13 shows membership functions for *low* (m_L) and *high* (m_H) for a 4-rule TS blending function they call b_{TS4}.

The numerical domain of the chosen features is normalized to [0, 4] here. This bizarre choice was a historical artifact of the evolution of this detector, and there is certainly no preternatural reason to prefer this scaling. The input features to the blending function are named $(\mathbf{x}) = (x, y)^T$. In example 5.4 $\mathbf{x} = \left(|f_{Sh}(\mathbf{w})|, |f_{Sv}(\mathbf{w})|\right)$, but there is no reason for the features to be limited to these, and Bezdek et al. show many edge images made with $\mathbf{f}_{rs}(\mathbf{w}_i) = (f_r(\mathbf{w}_i), f_s(\mathbf{w}_i))$ that are (visually) better than the absolute Sobel features. The output functions for b_{TS4} are specified to make it a four parameter family of models. Let $\tau, \chi, \gamma, \omega \in \Re^+$ and define the rules (see Bezdek et al., 1995 for more information about the choices for the consequent output functions) as :

$$\text{R1. If } x = L \text{ and } y = L \Rightarrow u_1(\mathbf{x}) = x^\tau + y^\tau \quad ; \qquad (5.22a)$$

$$\text{R2. If } x = L \text{ and } y = H \Rightarrow u_2(\mathbf{x}) = \chi \quad ; \qquad (5.22b)$$

$$\text{R3. If } x = H \text{ and } y = L \Rightarrow u_3(\mathbf{x}) = \gamma \quad ; \qquad (5.22c)$$

$$\text{R4. If } x = H \text{ and } y = H \Rightarrow u_4(\mathbf{x}) = \omega \quad . \qquad (5.22d)$$

Bezdek et al. chose the T_2 norm for aggregation of values of the premise membership functions, i.e., $T_2(a, b) = ab$. The functions in Figure 5.13 satisfy $m_L(x) + m_H(x) = 1$, so $m_H(x) = 1 - m_L(x)$ for x in [0, 4], and similarly for y. Since m_L and m_H are the same for x and y, the rules in (5.22) can be written in terms of a single membership function $m(z) = 1 - |z|/4$, where z can be x or y in [0, 4]. The firing strengths $\{\alpha_i(\mathbf{x}) = T(m(x), m(y))\}$ for the four rules in (5.22) are

$$R_1:\ \alpha_1(\mathbf{x}) = m(x) \cdot m(y) \quad ; \qquad (5.23a)$$

$$R_2:\ \alpha_2(\mathbf{x}) = m(x) \cdot (1 - m(y)) \quad ; \qquad (5.23b)$$

$$R_3:\ \alpha_3(\mathbf{x}) = (1 - m(x)) \cdot m(y) \quad ; \qquad (5.23c)$$

$$R_4:\ \alpha_4(\mathbf{x}) = (1 - m(x)) \cdot (1 - m(y)) \quad . \qquad (5.23d)$$

Substituting the right sides of (5.22) and (5.23) into equation (4.73) yields an explicit formula for b_{TS4}:

$$b_{TS4}(\mathbf{x}; \tau, \chi, \gamma, \omega) = \begin{array}{l} m(x)m(y)\left[x^\tau + y^\tau + \omega - \chi - \gamma\right] \\ +m(x)[\chi - \omega] + m(y)[\gamma - \omega] + \omega \end{array} \quad . \qquad (5.24)$$

This is a particularly simple fuzzy system as its output can be computed directly with (5.24). Since m(0)=1 and m(4)=0, boundary conditions can be computed from (5.24) at the four corners of the domain $[0, 4] \times [0, 4]$,

$$b_{TS4}((0,0);\tau,\lambda,\beta,\omega) = 0 \quad ; \quad (5.25a)$$
$$b_{TS4}((0,4);\tau,\chi,\gamma,\omega) = \chi \quad ; \quad (5.25b)$$
$$b_{TS4}((4,0);\tau,\chi,\gamma,\omega) = \gamma \quad ; \quad (5.25c)$$
$$b_{TS4}((4,4);\tau,\lambda,\beta,\omega) = \omega \quad . \quad (5.25d)$$

This shows that the constants on the right sides of R_2, R_3 and R_4 in (5.22) simply fix the values of b_{TS4} at the corresponding corners of its domain. It would be unusual not to specify $\chi = \gamma$, as this would destroy symmetry of the surface with respect to the plane $\{x = y\}$ in $\Re^3$ for features such as $f_{Sh}(\mathbf{w})$ and $f_{Sv}(\mathbf{w})$. Rule R_1 is the critical rule for b_{TS4} because $u_1(\mathbf{x}) = x^\tau + y^\tau$ controls the shape of its graph in the neighborhood of $\mathbf{0}$. Since m(x) and m(y) will both be close to 1 near 0, the value $x^\tau + y^\tau$, which is just the τ-th power of the Minkowski τ-norm of $\mathbf{x}$, will dominate (5.24) near $\mathbf{0}$.

Example 5.4 The blending function b_{TS4} was applied to the Lena image in Figure 5.4(a) with the following protocols: input features to the blending function were $\mathbf{x} = \left(\left|f_{Sh}(\mathbf{w})\right|, \left|f_{Sv}(\mathbf{w})\right|\right)$, scaled to the interval [0, 4]; $\tau = 4$, $\chi = \gamma = 2$ and $\omega = 3$. The output image was dynamically scaled (Bezdek et al., 1998a). Figure 5.14, right view, is the graph of the surface b_{TS4} over the input domain $[0, 4] \times [0, 4]$. For this τ the blending function is locally convex near $\mathbf{0}$, resulting in suppression of all but the brightest edges in the input image. The output is shown on the left side of Figure 5.14.

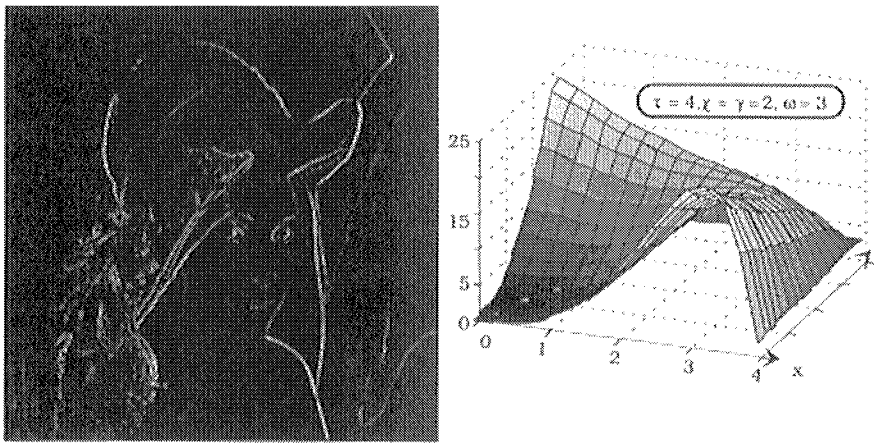


Figure 5.14 TS4 edge image and graph of b_{TS4} for $\tau = 4$

The left side of Figure 5.15 shows the output of the TS4 edge detector made with exactly the same computations as those discussed for Figure 5.14, except that for the right side of rule 1, $\tau = 1$. At this setting the blending function is locally linear near the origin (see the right view in Figure 5.15), resulting in an edge image that enhances structural details such as the feathers along the tail of the hat.

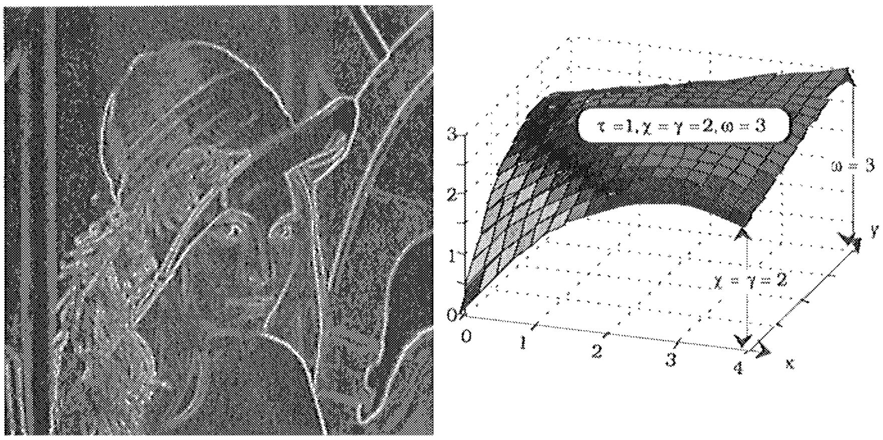


Figure 5.15 TS4 edge image and graph of the b_{TS4} for $\tau = 1$

As τ decreases, the shape of the blending surface defined by the function b_{TS4} continues to sharpen. The effect of this is seen in the left view of Figure 5.16, which depicts the TS4 edge image at $\tau = 0.25$.

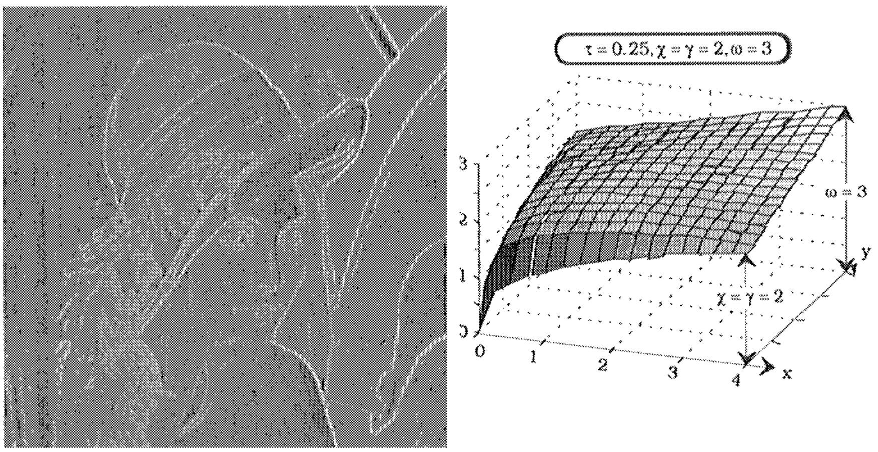


Figure 5.16 TS4 edge image and graph of the b_{TS4} for $\tau = 0.25$

These three views of Lena show that a very wide range of edge-enhanced images can be realized by simply adjusting a single parameter (τ) in the TS4 blending function. Bezdek et al. (1998a) discuss the utility of this feature for digital mammography, where an on-line viewing facility might enable practicing radiologists to view and tune enhanced edge images for optimal visual assessment in near real time.

Compare Figures 5.14-5.16 to previous and subsequent edge images of Lena. You will notice that the TS4 images have a "3D" like sheen to the edges. This is due in part to the fact that these three images are not thresholded. Instead, the pixel intensities are dynamically rescaled so that there is a full, 8 bit output range available for each pixel in these images. Applying any thresholding techniques discussed in Section 5.5.A to these three images would result in black and white edge images that have a more conventional appearance, such as those in Figure 5.20.

Summarizing, the key points made in Bezdek et al. are: (i) statistical features such as the range and standard deviation of window intensities can be as effective as more traditional features such as estimates of digital gradients; (ii) blending functions that are roughly concave near the origin of feature space can provide visually appealing edge images; (iii) geometric considerations can be used to specify the parameters of generalized logistic functions and TS systems that yield a rich variety of edge images; and (iv), understanding the geometry of the feature extraction and blending functions is the key to using models based on computational learning algorithms such as neural networks and fuzzy systems for edge detection.

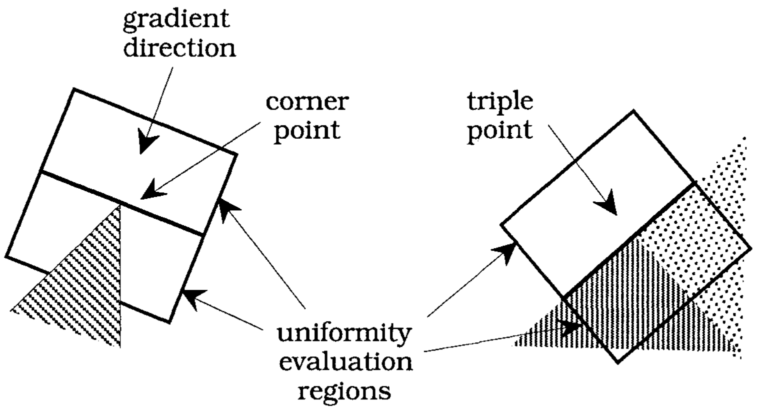


Figure 5.17 Corner and triple points in Law et al.'s edge detector

Law et al. (1996) propose a fuzzy logic based edge detector in which local features such as *gradient*, *symmetry* and *straightness* (see Section 5.2) are combined to determine *edgeness*, *cornerness*, and *tripleness*. They argue that the traditional definition for an edge point as the point of high gradient between two uniformly flat regions is not valid at corners (where a uniform region has a sharp corner) and junctions (where three regions meet). Figure 5.17 shows these situations.

The fuzzy rules used to compute memberships in the fuzzy sets *edgeness*, *cornerness*, and *tripleness* are summarized in Table 5.4. As in Tables 5.1 and 5.2, some compactification of these rules can be realized. For example, the first four rules can be replaced by the single rule : IF gradient is low THEN edgeness, cornerness and tripleness are low (and similarly for rules 5 and 6).

Table 5.4 Fuzzy rules for edgeness, cornerness and tripleness

gradient	symmetry	straight-ness	edge-ness	corner-ness	triple-ness
low	low	low	low	low	low
low	low	high	low	low	low
low	high	low	low	low	low
low	high	high	low	low	low
high	low	low	low	low	high
high	low	high	low	low	high
high	high	low	low	high	low
high	high	high	high	low	low

The memberships for *gradient*, *symmetry* and *straightness* are determined using gray-level values within the window, as explained in Section 5.2. The memberships in *edgeness*, *cornerness*, and *tripleness* are used to trace edges and join edge segments, as will be described in the next section.

5.4 Edge Linking

Section 5.3 discussed several fuzzy algorithms that identify pixels that may belong to an edge in an image. Many "edges" are (visually) fragments of larger edge structures. Edge linking techniques attempt to bind edge fragments, forming an image with better visual acuity than images made by edge detection.

Law et al. (1996) use *edgeness*, *cornerness*, and *tripleness* along with the edge direction perpendicular to the gradient direction to join edge fragments. They consider the four possible pairs of pixel neighbors (1,9), (2,8), (3,7) and (4,6) to a center pixel X_i as shown in Figure 5.18. If the edgeness of the central pixel exceeds that of the neighboring pixels, then the central pixel is marked as a crisp edge point. Then the edgeness image is thresholded to remove weak edge points, and T-, Y-, and X-shaped patterns are removed to simplify tracing. Law et al. do not specify how these patterns are detected. This removes junction points, but such points are taken into account by using *cornerness* and *tripleness* features.

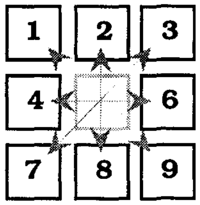


Figure 5.18 Four possible pairs of neighbors to X_i

To trace and join edge segments, Law et al. consider the four cases shown in Figure 5.19: joins between aligned edge segments (two end points), between two segments at a corner (two end points), between three segments that represent a "Y" junction (three end points), and between two segments at a triple point (one end point, one mid point).

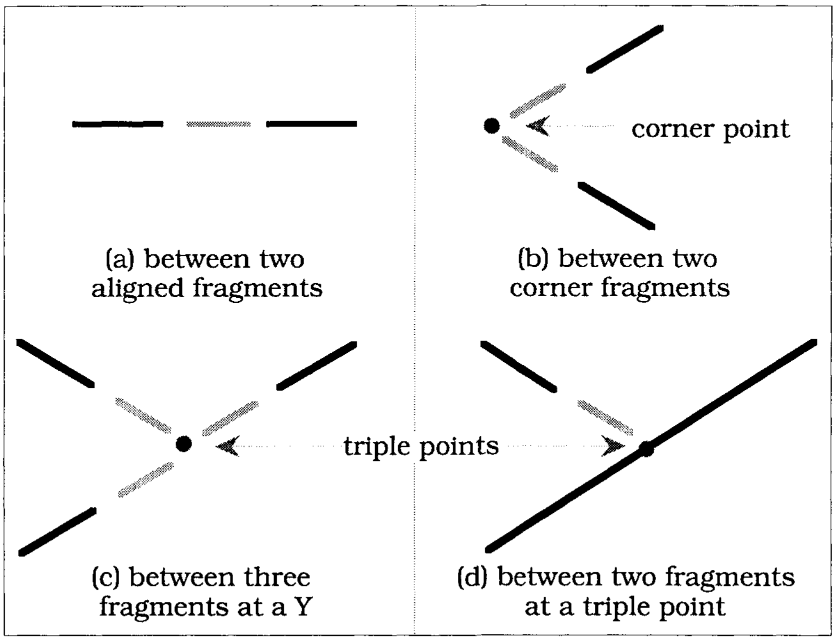

corner point
(a) between two aligned fragments
(b) between two corner fragments
triple points
(c) between three fragments at a Y
(d) between two fragments at a triple point

Figure 5.19 Types of joins considered by Law et al. (1996)

In addition to *edgeness*, *cornerness* and *tripleness*, fuzzy rules that govern the joining process are based on *alignment* and *proximity*. Alignment is the difference in angle between two edge fragments containing the end points, and proximity is the Euclidean distance between two end points. The fuzzy rules to compute joinness are listed in Tables 5.5-5.8. In Table 5.5, interim edgeness is computed by using the rules for edgeness in Table 5.4. The first rule from Table 5.5 reads:

IF	alignment is *low*
AND	proximity is *low*
AND	interim edgeness is *low*
THEN	joinness is *low*.

Table 5.5 Fuzzy rules for aligned-edge join (Figure 5.19(a))

alignment	proximity	interim edgeness	joinness
low	low	low	low
low	low	high	low
low	high	low	low
low	high	high	medium
high	low	low	low
high	low	high	medium
high	high	low	medium
high	high	high	high

Table 5.6 Fuzzy rules for corner join (Figure 5.19(b))

cornerness at intersection	proximity	joinness
low	low	low
low	high	low
high	low	low
high	high	high

Table 5.7 Fuzzy rules for 3-edge triple join (Figure 5.19(c))

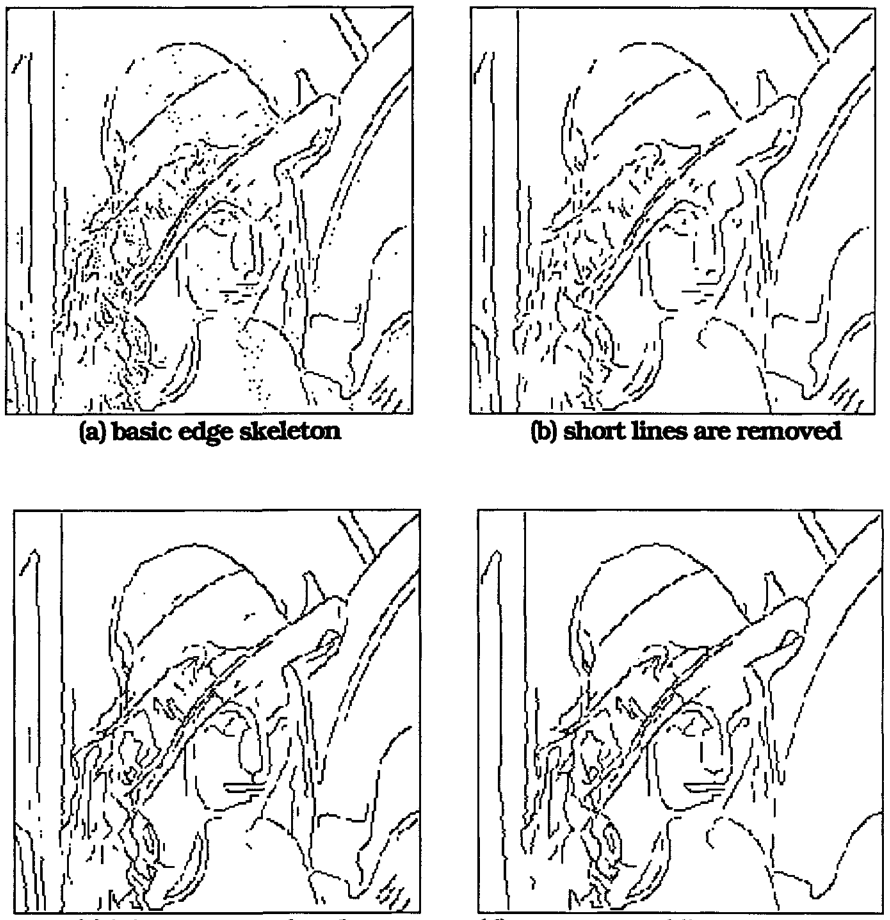
tripleness at intersection	proximity	joinness
low	low	low
low	high	low
high	low	low
high	high	high

Table 5.8 Fuzzy rules for 2-edge triple join (Figure 5.19(d))

tripleness near intersection	proximity	joinness
low	low	low
low	high	low
high	low	low
high	high	high

Membership functions for linguistic values such as *low* and *high* can be found in Law et al. (1996). As in several previous examples, the first two rows in each of Tables 5.6-5.8 can be combined into single rules, but Law et al. (1996) show them as above.

Example 5.5 Figure 5.20 illustrates the edge joining process proposed by Law et al. (1996). Figure 5.20(a) shows the basic skeleton of edge fragments. Figure 5.20(b) shows the result after short lines with less than three pixels are removed. Figure 5.20(c) is the result after the joins are completed. Figure 5.20(d) is the final result after unconnected lines are removed.

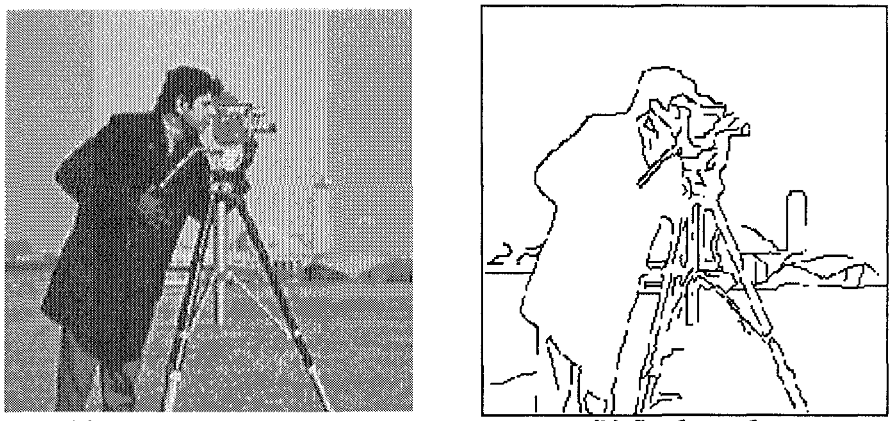


(a) basic edge skeleton **(b) short lines are removed**

(c) joins are completed **(d) unconnected lines are removed**

Figure 5.20 Edge joining as given in Law et al. (1996)

Figure 5.21(a) shows the original "camera man" image, and view 5.21(b) shows the result of Law et al.'s procedure after the steps of fuzzy filtering, edge detection, tracing and joining.



(a) "camera man" image **(b) final result**

Figure 5.21 Law et al.'s procedure illustrated on the camera man

Kim and Cho also (1994) describe a fuzzy reasoning method to perform edge linking. Their algorithm is based on the relaxation labeling approach proposed by Hanson and Riseman (1978). The basic idea is to increase or decrease the edge strength associated with a crack edge depending on its compatibility with the crack edge strengths in the neighborhood. Figure 5.22 illustrates the idea of a crack edge.

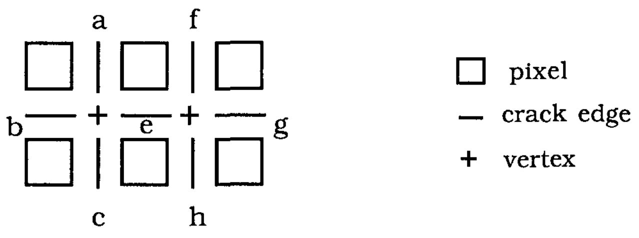


Figure 5.22 A neighborhood of crack edges

A crack edge occurs between a pair of adjacent pixels. A neighborhood consisting of the center crack edge e and six other crack edges labeled a, b, c, f, g, and h are shown in Figure 5.22. Kim and Cho heuristically select 10 compatibility relationships between an edge and its neighboring edges based on considerations such as linearity of edges. The edge strength associated with a compatible crack edge is increased, and the edge strength associated with an incompatible crack edge is decreased. This relaxation process is repeated until convergence. Figures 5.23(a)-(c) show examples where the edge strength of the central crack edge (denoted by a blank rectangle) should be decreased. Here, filled rectangles indicate strong crack edges and dotted lines indicate weak edges. Similarly, Figures 5.23(d)-(f) show examples where the strength of the central crack edge should be increased. Kim and Cho associate one fuzzy rule with each of 10 cases (6 of the 10 rules are shown).

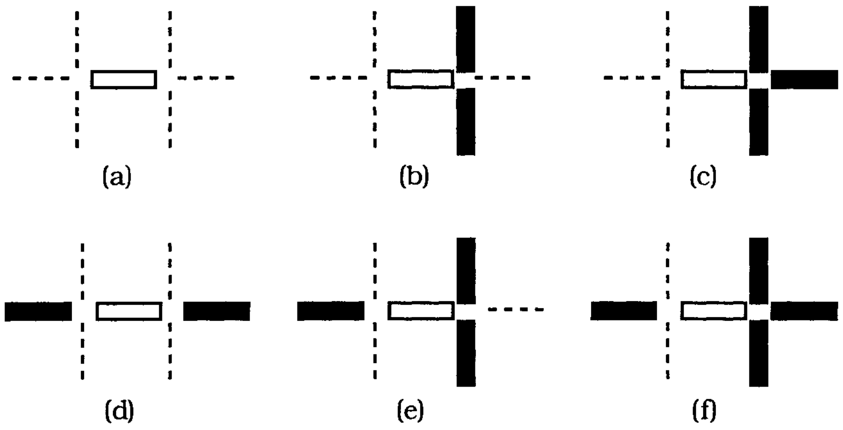


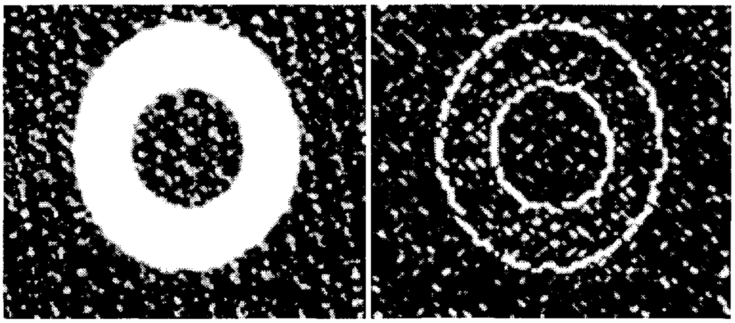
Figure 5.23 Incompatible (a-c) and compatible (d-f) crack edges

For example, the rule corresponding to Figure 5.23(d) is

IF	a is *small*
AND	b is *big*
AND	c is *small*
AND	f is *small*
AND	g is *big*
AND	h is *small)*
THEN	e is increased by *positive large.*

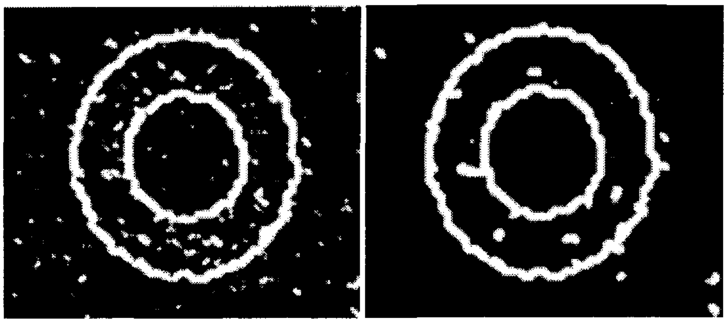
Kim and Cho use Gaussian-shaped membership functions for the antecedents *big* and *small* and tune the parameters of the membership functions by training a neural network with a set of synthetically generated crack edge data. The consequent membership functions that represent positive large and negative large are modeled by crisp singletons (±1 respectively), so this is another instance of the 0-th order TS model. Kim and Cho show that fuzzy edge relaxation is faster and gives better results when compared with traditional techniques.

Example 5.6 Figure 5.24(a) shows the 128×128 noisy image of a ring used by Kim and Cho (1994). Figure 5.24(b) shows the corresponding edge image obtained by applying the crack edge operator (see Rosenfeld and Kak, 1982) and then thinning the result by non-maximal suppression.



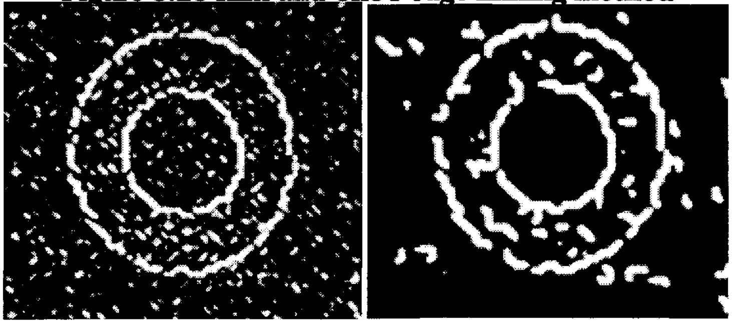
(a) Original noisy image **(b) Initial edge image**

Figure 5.24 Raw data for edge linking examples



(a) 1 iteration **(b) 5 iterations**

Figure 5.25 Kim and Cho's edge linking method



(a) 1 iteration **(b) 40 iterations**

Figure 5.26 Hanson-Riseman's edge linking method

The result after one iteration of applying Kim and Cho's fuzzy relaxation algorithm to the image in 5.24(b) is shown in Figure 5.25(a). The result after five iterations is shown in Figure 5.25(b).

The results of applying the Hanson-Riseman method to the edge image in Figure 5.24(b) after 1 and 40 iterations are shown in Figures 5.26(a) and (b) respectively. Visually, the Kim and Cho output at 5 iterations is quite superior to the Hanson-Riseman output after 40 iterations.

5.5 Segmentation

Image segmentation is an important step in many computer vision algorithms. The objective of segmentation is to divide an image into (meaningful) regions. Errors made in this stage will impact all higher level activities. Therefore, methods which incorporate the uncertainty of object and region definition and the faithfulness of the features to represent various objects (regions) are desirable. Pal and Pal (1993) and Bezdek and Sutton (1999) have contributed to the plethora of surveys on this topic.

In a segmented image, ideally each region should be homogenous with respect to some characteristics or features such as gray level or texture, and adjacent regions should have significantly different characteristics or features (Haralick and Shapiro, 1992). As in Section 5.1, let IJ denote the two-dimensional domain of the image $\mathbf{P}_{IJ}$. More formally (Fu, 1982), segmentation is the process of partitioning the entire image $\mathbf{P}_{IJ}$ into c crisp and maximally connected subregions such that each region R_i is homogeneous with respect to some predicate $\wp$, i. e.,

$$\begin{aligned}
&\bigcup_{i=1}^{c} R_i = \mathbf{P}_{IJ} \\
&R_i \cap R_j = \varnothing \ \ \forall\, i,j,\ \ i \neq j \\
&R_i, 1, \ldots, c \text{ are connected} \\
&\wp(R_i) = \text{TRUE} \ \ \forall\, i \\
&\wp(R_i \cup R_j) = \text{FALSE if } i \neq j \text{ and } R_i \text{ is adjacent to } R_j
\end{aligned} \quad . \qquad (5.26)$$

The crisp membership function $m_{R_i} : IJ \to \{0,1\}$ of a region R_i is

$$m_{R_i}(i,j) = \begin{Bmatrix} 1 & ; & (i,j) \in R_i \\ 0 & ; & (i,j) \notin R_i \end{Bmatrix} \quad . \qquad (5.27)$$

In many situations it is not easy to determine if a pixel should belong to a region or not. This is because the features used to determine homogeneity may not have sharp transitions at region boundaries. This is especially true when features are computed using, say, a local 3x3 or 5x5 window. To alleviate this situation, we can insert fuzzy sets concepts into the segmentation process. The first reference to fuzzy segmentation was made by Prewitt (1970), who suggested that the results of image segmentation should be fuzzy subsets rather than crisp subsets of the image plane. In a fuzzy segmentation, each pixel is assigned a membership value in each of the regions. If the memberships are taken into account while computing properties of regions, we often obtain more accurate estimates of region properties. This will be discussed further in Section 5.7.

The result of a fuzzy segmentation is a partition of $\mathbf{P}_{IJ}$ into c fuzzy subsets $\{R_i\}$. Each R_i is represented by its membership function $m_{R_i}: IJ \rightarrow [0,1]$, which replaces the membership function in (5.27). For $(i,j) \in \mathbf{P}_{IJ}$, $m_{R_i}(i,j)$ represents the degree to which (i,j) belongs to R_i. A fuzzy segmentation of an image into c regions is a fuzzy cxn partition matrix $U = [u_{ij}]$, where $u_{ij} = m_{R_i}(i,j)$. This construction loses the connectivity among regions that is presumably enforced by construction of the predicate $\wp$ in (5.26).

A. Segmentation via thresholding

Thresholding is one of the simplest methods to obtain a crisp segmentation a unispectral image. Thresholding generates a binary image in which the pixels belonging to objects have the value 1 and pixels belonging to the background have the value 0. Binary images are popular, but images are normally acquired as gray-scale images. Ideally, objects in the image should appear consistently brighter (or darker) than the background. Under such conditions, a binary image of the object can be obtained by thresholding the gray level image. Using X_i for location (i, j), the thresholded image is given by

$$I_T(X_i) = \begin{Bmatrix} 1 & ; \ I(X_i) \geq \tau \\ 0 & ; \ I(X_i) < \tau \end{Bmatrix} \quad . \tag{5.28}$$

There are several traditional thresholding techniques in the literature to determine the "correct" threshold τ (Sahoo et al. 1988, Pal and Pal, 1993). Two broad categories of fuzzy techniques are: (1) methods that search for a threshold τ which maximizes or minimizes an index of fuzziness based on the membership values assigned to pixels in the image; and (2) methods that cluster the gray values into two classes. Methods based on clustering will be discussed in the next section.

Typically it is assumed that the gray-level of a pixel is related to the degree of membership of the pixel in the object. If the object is lighter than the background, then it is reasonable to suppose that the higher the gray-level value, the higher the membership value of the pixel in the object region. Therefore, the membership function $m_{R_O}(X_i)$ is obtained by mapping gray levels into the interval $[0,1]$. This mapping is generally monotone increasing. Let $I(X) \in [0, G-1]$ denote the gray level of a pixel. Here we present two of the most frequently used mappings for generating membership functions. Let

$$m_{R_O}(I(X)) = \left\{ \begin{array}{ll} 0 & ;\ I(X) < a_1 \\ aI(X) + b & ;a_1 \le I(X) \le a_2 \\ 1 & ;\ I(X) > a_2 \end{array} \right\} \quad \text{, where} \tag{5.29}$$

$$a = \frac{1}{a_2 - a_1} \text{ and } b = \frac{a_1}{a_1 - a_2} \quad . \tag{5.30}$$

If a_1 is the minimum gray level in the image and a_2 is the maximum gray level, then we have a simple linear mapping. Equation (5.30) has the following advantages: it provides a reasonably smooth transition between background and object regions, it can be easily manipulated by fuzzy operators, and it lends itself to hardware implementations if speed is crucial. Another popular mapping is the so-called S-function

$$m_{R_O}(I(X)) = S(I(X)) = \left\{ \begin{array}{ll} 0 & ;I(X) \le a \\ \frac{1}{2}\left(\frac{I(X)-a}{b-a}\right)^2 & ;a < I(X) \le b \\ 1 - \frac{1}{2}\left(\frac{I(X)-c}{c-b}\right)^2 & ;b < I(X) \le c \\ 1 & ;I(X) > c \end{array} \right\} . \tag{5.31}$$

In (5.29) $(a_1+a_2)/2$ is the cross-over point; in (5.31), b is the cross-over point. Gray values above the cross-over point have memberships greater than 0.5, and gray values below the cross-over point have memberships less than 0.5. Therefore, a thresholded image can be obtained by choosing τ to be the cross-over point.

To find the optimum cross-over point or threshold, we can compute measures such as the linear/quadratic index of fuzziness (Pal et al. 1983a, Murthy and Pal 1990), fuzzy compactness (Pal and Rosenfeld 1988), or index of area coverage (Pal and Ghosh 1990). Usually a and c in (5.31) are fixed, and b is varied. For each b, the index of fuzziness is computed, and for any of the three indices just mentioned, we pick $\tau = b$ in (5.31), corresponding to either the global minimum or maximum, depending on which measure is used, as the threshold. The linear index of fuzziness (Kaufmann 1975) is

$$\gamma_\ell(P_{IJ}) = \frac{2}{n}\sum_{i=1}^{n}\min\left(m_{R_O}(X_i), 1 - m_{R_O}(X_i)\right) \quad . \tag{5.32}$$

The quadratic index of fuzziness (Kaufmann 1975) is

$$\gamma_q(P_{IJ}) = \frac{2}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}\left[\min\left(m_{R_O}(X_i), 1 - m_{R_O}(X_i)\right)\right]^2} \quad . \tag{5.33}$$

Definitions of compactness and index of area coverage are given in Section 5.7. We mention that any measure of fuzziness can be used for thresholding; see Pal and Bezdek (1994) for an extensive list of other indices. Fuzzy divergence and probability measures can also be used for object-background segmentation of images (Bhandari et al., 1992).

Object boundaries in gray-scale images are often blurred and distorted due to the imaging process. The thresholding operation does not preserve the uncertainty in the image, and could distort the shape and size of the object. An alternative approach is to preserve the uncertainty inherent in the image as long as possible until actual decisions have to be made. This will be discussed further in Section 5.7.

B. Segmentation via clustering

In general, pixel intensity is not directly related to membership degrees of the pixel in the objects/regions (for example, a textured region). Therefore, we need to generate membership functions for regions in the image. If we extract p features at pixel X_i, then X_i can be represented by a feature vector $\mathbf{x}_i$ in $\Re^p$. The components of $\mathbf{x}_i$ may be just intensities (in the case of multispectral images), or just functions of the intensities, or both. Moreover, if we select the features judiciously, then the feature vectors corresponding to each meaningful region may form clusters in $\Re^p$. Presumably, the object data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ corresponding to the n pixels in the image can be clustered into a required number c of clusters using a suitable hard/fuzzy/possibilistic/probabilistic clustering algorithm.

Clustering algorithms generate a c×n partition matrix $U = [u_{ik}]$, where u_{ik} is the membership of the k-th pixel X_k in the i-th region R_i. The i-th row of U contains values of the membership function m_{R_i} of R_i. Hardening the columns of U with (1.15) then gives us a crisp segmentation of the image. However, since clustering is unsupervised, it is not possible to predict what clusters will emerge from a perceptual standpoint. Moreover, there is no guarantee that the $\{R_i\}$ generated by hardening U will be connected regions in the image, or regions satisfying a predicate like those in (5.26). And as usual, you have to worry about how to specify or identify the number of classes c.

In terms of generating membership functions for later processing, the c-means clustering models have several advantages. They are unsupervised; they can be used with any number of features and classes; and they distribute membership values across the classes based on "natural" groupings in feature space.

When n is large, (e.g. n=65,536 for a 512×512 image) clustering methods are time consuming. However, some simplifications are possible if features have quantized values. For example, if the gray level of a pixel is a feature, then typically there are only 256 possible values for the feature. In a given image, there will usually be many pixels with identical feature vectors. Therefore, X can be stored in a more compact form as $X = \{(h_1, \mathbf{x}_1), \cdots, (h_k, \mathbf{x}_k), \cdots, (h_q, \mathbf{x}_q)\}$, where the feature vectors $\mathbf{x}_k$ are all distinct, h_k is the frequency of occurrence of $\mathbf{x}_k$, and q is the total number of distinct feature vectors. The values $\{h_i\}$ can be obtained from the p-dimensional histogram of the feature vectors. Since the membership of a feature vector in a cluster depends only on the values of the features, (2.7b) or (2.8b) can be written as

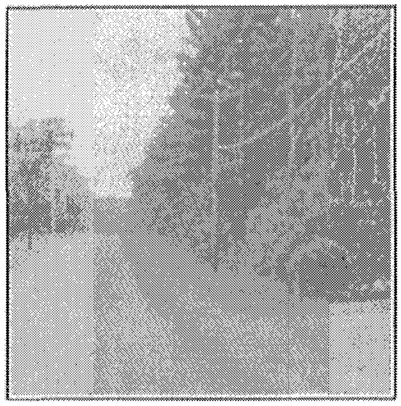
$$\mathbf{v}_i = \frac{\sum_{k=1}^{q} u_{ik}^m \cdot h_k \mathbf{x}_k}{\sum_{k=1}^{q} u_{ik}^m \cdot h_k},\ i = 1,\ldots,c \quad . \tag{5.34}$$

If q<<n, then (5.34) is considerably more efficient than the original formulation. Since an array of size c×q (rather than c×n) can be used to store all the memberships, there will be considerable savings in memory as well.

The segmentation obtained depends very much on the distance measure used in the c-means algorithm. Euclidean distance is effective only when the clusters are well separated, when they are expected to be hyperspherical, and when they are approximately equal in size. When this is not the case, other algorithms such as the Gustafson-Kessel (1978) algorithm (see Section 2.4) or the Gaussian mixture decomposition (GMD) algorithm (see Section 2.3) can be used.

Example 5.7 As an illustration we compare the performance of FCM with the Euclidean norm with that of the GK and GMD algorithms on a simple segmentation problem. In all cases the number of clusters specified was c = 3. FCM used the first three feature vectors as the initial prototypes. GK and GMD use the fuzzy partition generated by 5 iterations of FCM for their initializations. Termination in all cases occurred when the absolute change in every membership u_{ij} (or posterior probability p_{ij}) was less than 0.001. That is, the termination norm was $\|U_t - U_{t-1}\|_\infty \le 0.001$.

Figure 5.27(a) shows the original 256×256 image of an outdoor scene. The original image is a color image with red (r), green (g) and blue (b) components. We used two Ohta (1985) features, intensity =(r+g+b)/3 and excess green =(2g-r-b). For convenience we call this $\Re^2$(Ohta). Since the sky is very uniform, the sky feature vectors form a highly compact cluster (bottom left in Figures 5.27(b-d). Figures 5.27(b-d) show the clustering results in $\Re^2$(Ohta). The black dots are the estimated centers and the ellipses enclose points within a Mahalanobis distance of 2 of each center point. The third ellipse in Figure 5.27(d) - the sky - is really tiny, but it's there - see if you can find it. Equation (2.27) is used to compute the covariance matrix in the Mahalanobis distance after the algorithms terminate. We have repeated the original image, panel (a) on this page, in the lower right panel of the next page so you can compare the segmentations shown there to the input image without flipping back and forth to this page.


(a) An outdoor scene we've all seen

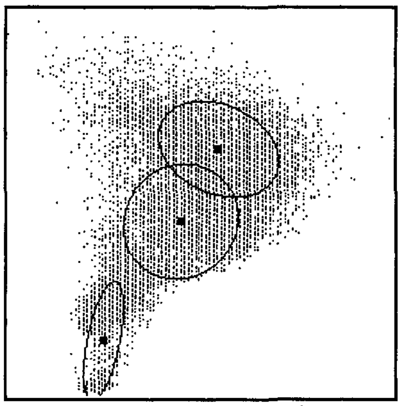
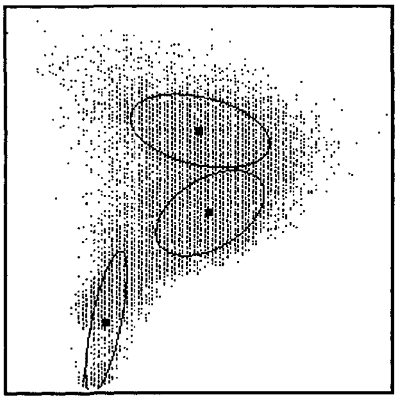
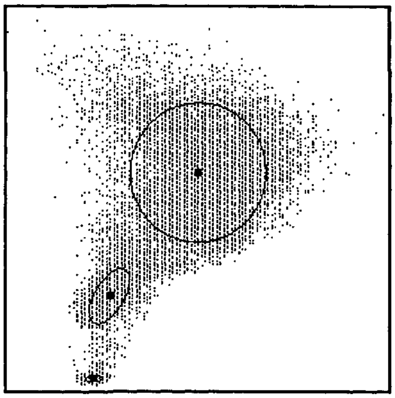
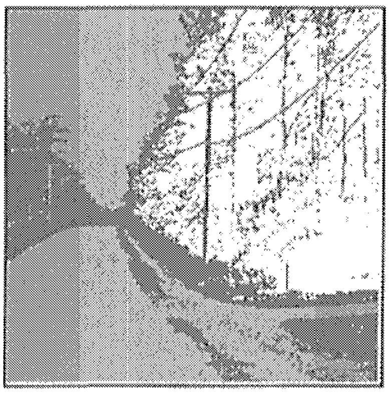
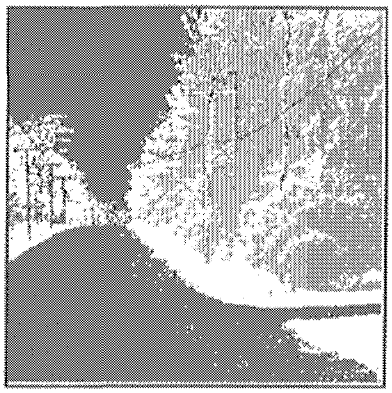

(b) FCM clusters in $\Re^2$(Ohta)

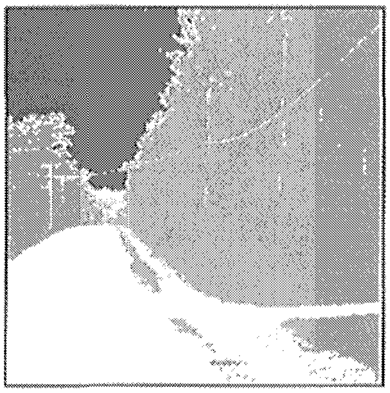
Figure 5.27 Segmentation via clustering


(c) GK clusters in $\Re^2$(Ohta)


(d) GMD clusters in $\Re^2$(Ohta)


(e) FCM segmentation


(f) GK segmentation


(g) GMD segmentation

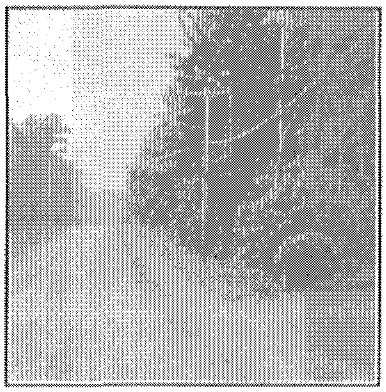
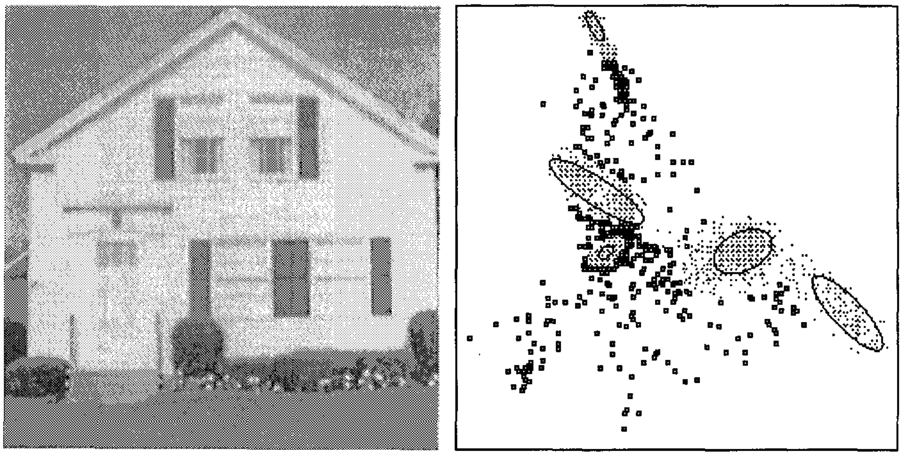

Raw image (repeated)

Figure 5.27 (con't.) Segmentation via clustering

Figures 5.27(e) and (f) show the segmentation results achieved by the FCM and GK algorithms respectively, where the feature points from the sky and road regions are lumped into one cluster and the feature points from the tree region are divided into two clusters. The segmentation result from the GMD algorithm in Figure 5.27(g) is quite good, giving a clear distinction between the road and sky.

In many segmentation applications (e.g., outdoor scenes), the number of clusters is not known in advance. In such cases, the methods discussed above cannot be applied without using, for example, an ancillary validity measure to determine the number of clusters (see Section 2.6). This approach can be computationally expensive. An alternative is to use an algorithm such as RCA (see Section 2.5), which determines the number of clusters dynamically during execution of the algorithm.

Example 5.8 Figure 5.28(a) shows the intensity representation of a color image of a house. The color image has red (r), green (g) and blue (b) components. The outputs shown are based on a reduced image that was made by using only every third pixel in the image in both the horizontal and vertical directions to reduce computation time. Figure 5.28(b) shows the five clusters identified by RCA in the 2D feature space extracted from the 3 intensities, where x_1 = red-blue difference (r-b) and x_2 = excess green (2g-r-b).



(a) Original image **(b) Clusters in feature space**

Figure 5.28 Segmentation by the RCA algorithm

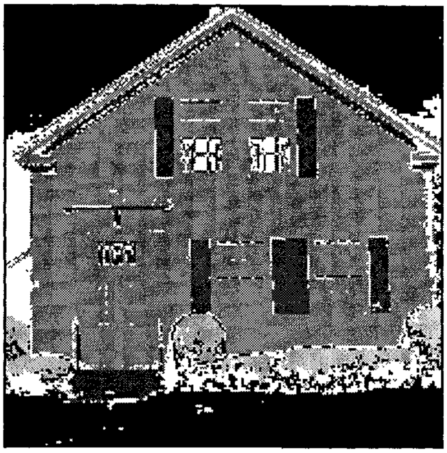

(c) Segmented image

Figure 5.28 (con't.) Segmentation by the RCA algorithm

The GK distance at (2.28) was used in this example. The ellipses enclose points within a Mahalanobis distance of 4. RCA was terminated when the prototypes did not change significantly between two successive iterations. Since each "prototype" consists of a point prototype $\mathbf{v}_i$ and a covariance matrix C_i, the termination condition was jointly applied to both sets of parameters, viz., $\underbrace{\max}_{1 \le i \le c}\left\{\left\|\mathbf{v}_{i,t} - \mathbf{v}_{i,t-1}\right\|_1\right\} < 0.01$ and $\underbrace{\max}_{1 \le i \le c}\left\{\left\|C_{i,t} - C_{i,t-1}\right\|_1\right\} < 0.1$, where t represents the iteration number.

Outlier points (i.e., points for which the possibilistic weights are equal or nearly equal to zero in all clusters) are shown as small squares in Figure 5.28(b). Figure 5.28(c) shows the segmentation corresponding to Figure 5.28(b), where each point is assigned to the cluster with the largest membership. Outlier points are shown in white in this figure, and they correspond to small and narrow classes such as the edge of the roof and the trees on the left and right sides of the house. The vertical region to the left of the house is segmented incorrectly in several places. This is probably due to small differences in color which are visible in the original image that are differenced out during feature extraction. An alternative explanation is that the sampling used made these classes have too few points in the 2D feature space to be identified as legitimate clusters.

Boujemaa et al. (1992a) present a segmentation algorithm that models the uncertainty in pixel information based on fuzzy clustering (e.g., FCM). They use the idea of a "gradually focusing decision". The algorithm proceeds in two steps. The first global step selects (in a non-sequential way) the most "ambiguous" pixels, which are close to regions boundaries. The ambiguity is measured by fuzzy memberships. The "strongest" (or least ambiguous) pixels represent the coarse information of the scene and locate the inner parts of regions. The second step performs fine segmentation of boundaries. It focuses exclusively on the ambiguous pixels and ignores all the others already classified in the previous global stage. Each ambiguous pixel is merged with one of the neighboring regions in the local neighborhood of the pixel. This way, "weak" and "vague" pixels, representing transition zones, are disambiguated by strong, contextual region membership information. This coarse to fine segmentation strategy provides finer boundary localization and smooth edge detection. Boujemaa et al. (1992b) apply this technique to images in several medical domains, including ventricular endocardium detection.

C. Supervised segmentation

If segmentation is unsupervised (the input data is not labeled), then either humans or an additional level of post-processing is needed to assign meaningful physical labels to algorithmically determined regions. Each object or region in the segmented image is labeled based on one or more properties of the region. (See Section 5.7 for a discussion of properties of fuzzy regions.)

However, in many situations, unsupervised methods do not give good segmentation results, because feature vectors belonging to different objects in the image may not be well separated (for example, see Figure 5.27(b)). To overcome this problem, segmentation and labeling can be performed simultaneously, by using the labels of some pixels extracted from the input data prior to the training phase of the segmentation process. The semi-supervised FCM algorithm discussed in Section 2.3 is one example of this approach (Bensaid et al. 1996a), and our Example 4.7 is an another illustration of supervised segmentation that uses the crisp k-nn rule as the classifier. We will comment on other supervised approaches to segmentation with fuzzy models in Section 5.11. In this subsection, we consider an approach based on *fuzzy aggregation networks* (FAN's) (Section 4.7.D). In this approach, labeled feature vectors are used to train a FAN, and the output of the FAN is used to segment and label the image simultaneously.

Example 5.9 Figure 5.30(a) shows a 256×256 image of a guy checking out the neighborhood - an outdoor scene we've all seen. The original image was in color, and from it we extracted feature vectors consisting of the three Ohta (1985) color features and a position feature. The color features are computed from the r, g and b (red, green and blue) components of the image as: excess green = (2g-r-b), red-blue difference = (r-b), and intensity = (r+b+g)/3. The position feature is simply the row number. Here we describe a situation where about 0.5% of the image data was used for training and the whole image was used in testing. Krishnapuram and Lee (1992a, 1992b) and Keller and Chen (1992a) show examples where the training and testing scenes are different. These experiments were conducted with several information fusion structures.

The features were normalized so that all values fell between 0 and 255. The training data consisted of 60 feature vectors for each type of object. In this example, 6 objects (classes) were chosen: *sky, tree, roof, wall, grass* and *road*. In the fuzzy aggregation network approach, we need to compute the membership of each feature in each of the classes. Therefore, we need to estimate the membership function for each feature for each object. The smoothed and normalized histogram of the values of feature i of pixels from class k was used as the membership function m_{ik}, k=1,...,6, i=1,...,4. Smoothing was achieved by averaging the histograms twice using a window of length 11. The overall training data consisted of 60×6 entries of 6×4=24 memberships. The training method adjusts the parameters of the aggregation operator used at the top nodes by a gradient descent technique.

We present the results of three kinds of aggregation operators: the multiplicative γ-model (Φ_M neurons), the additive γ-model (Φ_A neurons) and the additive γ-model with Yager's union and intersection operators (Φ_Y neurons). These operators behave like intersection operators when γ is close to 0 and like union operators when γ is close to 1. In addition, they all have parameters w_i that reflect the relative importances of the features (see Section 4.7). The Φ_Y model has an additional parameter π that regulates the severity of the union/intersection operators. This parameter may be chosen, or can be learned, as was done in this example, using the gradient descent procedure.

Figure 5.29 shows the FAN used. While training, the desired output of node k in the top layer was 0.99 if the 24-dimensional membership vector came from class k, and 0.01 otherwise. After training, the resulting network was used for segmentation of the image. The features corresponding to each pixel in the image were fed to the network and the memberships in the 6 classes generated by the top nodes were recorded. For display purposes, the fuzzy segmentation was hardened in the usual way, i.e., each pixel is given the label of the node that has the highest membership. Since the memberships generated by the network need not sum to 1, they can be considered as possibilistic memberships.

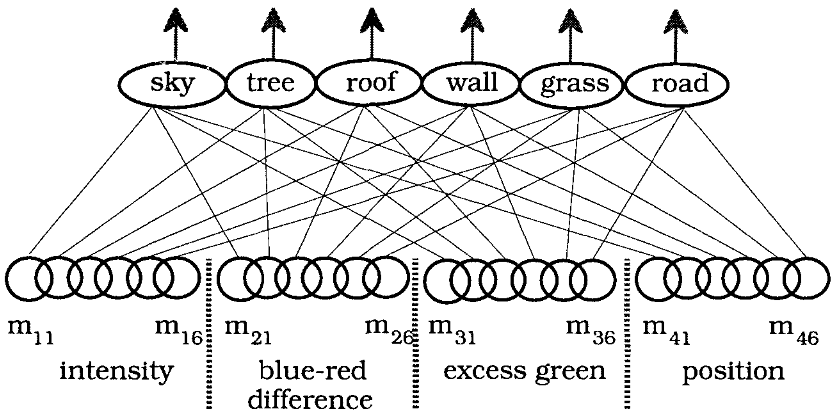


Figure 5.29 The FAN structure used for image segmentation

Table 5.9 The parameter values of FAN using the Φ_M model

Node	γ	w's			
sky	0.83	1.50	0.46	0.38	1.66
tree	0.68	2.35	0.21	0.52	0.93
roof	0.82	1.36	0.85	0.33	1.46
wall	0.84	1.57	0.00	0.00	2.42
grass	0.99	0.01	0.02	0.30	3.67
road	0.95	0.09	0.73	1.17	2.00

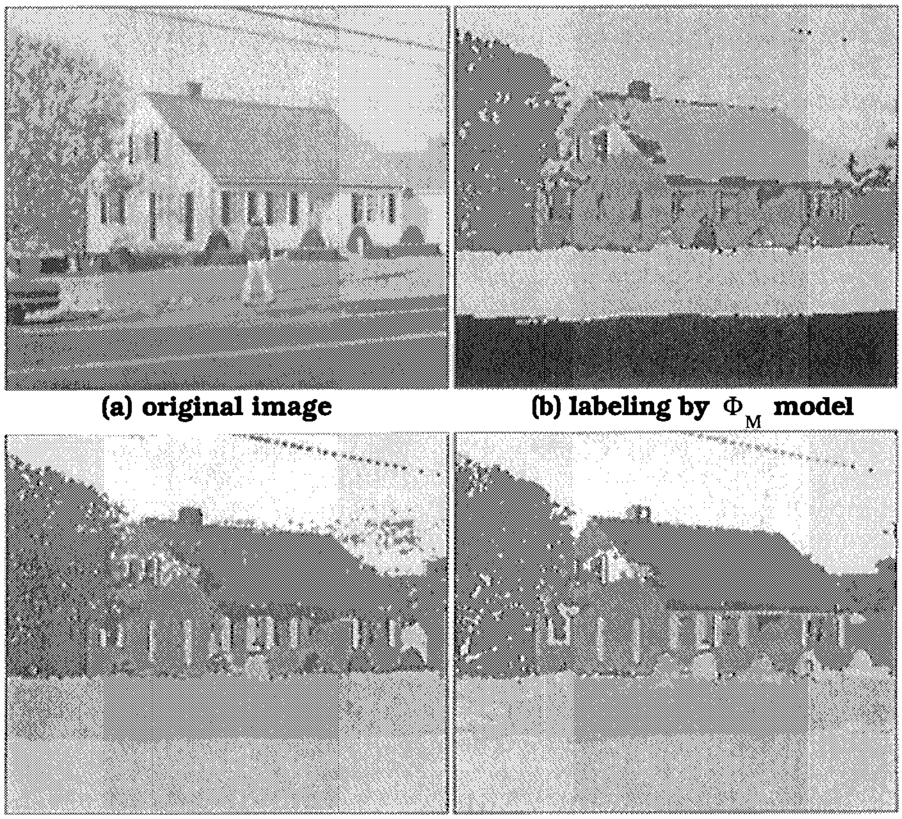
Table 5.10 The parameter values of FAN using the Φ_A model

Node	γ	w's			
sky	0.05	0.31	0.02	3.41	0.26
tree	0.15	3.35	0.10	0.10	0.45
roof	0.10	0.31	3.49	0.03	0.17
wall	0.09	0.32	0.00	3.14	0.54
grass	0.83	0.01	0.01	0.16	3.83
road	0.79	0.01	0.76	0.97	2.25

Table 5.11 The parameter values of FAN using the Φ_Y model

Node	γ	π	w's			
sky	0.02	2.93	0.36	0.38	0.57	0.30
tree	0.02	3.12	0.94	0.28	0.32	0.43
roof	0.02	2.99	0.30	0.42	0.27	0.33
wall	0.01	0.79	0.13	0.00	0.04	0.41
grass	0.50	3.24	1.26	1.00	0.93	0.39
road	0.75	2.08	2.79	0.94	0.80	0.73

The final parameters of each node after training are displayed in Tables 5.9, 5.10, and 5.11 respectively. For all models, the w's of the nodes for *grass* and *road* are union-like since they are close to 1. We can also conclude that the second feature "b-r" is redundant for the class *wall* since all models produce a value of w_i that is almost 0 for this feature.



(a) original image **(b) labeling by Φ_M model**

(c) labeling by Φ_A model **(d) labeling by Φ_Y model**

Figure 5.30 Image segmentation and labeling using a FAN

The segmented images corresponding to the Φ_M, Φ_A, and Φ_Y models are shown in Figures 5.30(b)-5.30(d) respectively. The Φ_M and Φ_Y neuron model results are similar to each other, and somewhat better than Φ_A , which is similar to other published segmentations of this scene. For those pixels belonging to the six defined classes, the segmentation results are quite good. Most misclassified pixels belong to the objects which are not defined as a class, i.e., the human, bushes and windows, or at the boundaries between regions.

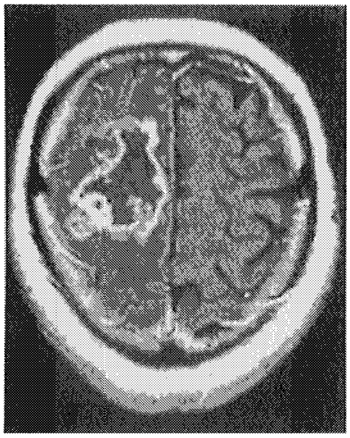
D. Rule-Based Segmentation

The most difficult aspect of automated segmentation is that no matter how good the training data are, unseen images will contain new objects, or objects that are sufficiently different from those in the training data, that the new image falls well outside the "experience" of the system. For example, different abnormal patients (with, say, brain tumors), simply have very different pathologies and anatomical structures from each other, so it is very hard to find a classifier that generalizes well across many abnormal patients. Most vision-aided assistance systems are aimed more towards separating images into normal and abnormal groups (perhaps with an additional "don't know" class), and then calling for help. In the context of automatic target recognition, for example, much progress has been made in the *detection* of targets in their background (where is an object?), but much less progress is evident in *recognition* (what is the object?). In this subsection we show how rules that attempt to capture human expertise can be used to augment low level segmentation - a step, we think, on the way towards truly automatic scene interpretation systems.

Bezdek et al. (1997a) summarize an ongoing body of work (Hall et al., 1992, Li et al., 1993, Clark et al., 1994, Vaidyanathan et al., 1995, Cheng et al., 1995, Bensaid et al., 1996b, Clark et al., 1998)) by comparing the results of unsupervised, supervised and rule-based segmentations of images in the medical domain obtained with a *knowledge-based* (KB) system. We repeat part of an example given in Bezdek et al. which ties together several algorithms discussed in Chapters 2, 4 and 5. In this discussion a *true positive* (true negative) is a *correctly identified tumor* (non-tumor) pixel; while false positives and false negatives correspond, respectively, to "false alarms" (pixels which are called tumor that are not), and "missed targets" (pixels classified as not tumor which are tumor pixels).

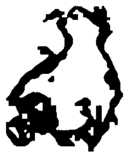
The image shown in Figure 5.31(a) is the T1 data of an MR image from a (T1, ρ, T2) MR slice of a patient with a tumor in the middle left section of the brain. The numerical features extracted from this image are the intensities of the pixels in these three dimensions. In this example we denote the 3D pixel data as X. The tumor is the white area that appears as an outline or boundary of a darker region within it. The white pixels are the tumor, and possibly blood vessels feeding it. The darker region within the tumor is (possibly) a combination of white matter, gray matter, and dead tissue.

Views (b)-(h) in Figure 5.31 are a set of black and white images made from the original (3D) image data by various techniques, each of which is an estimate of the tumor pixels in X. In these views "Seg" stands for "segmentation of". Figure 5.31(b) shows the ground truth image for the tumor, hand-labeled by an expert radiologist.

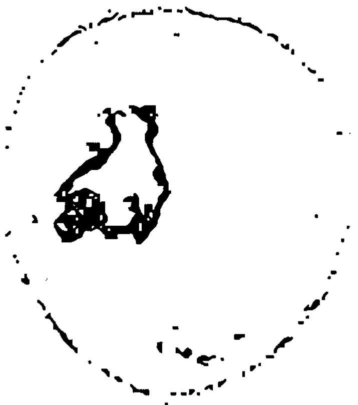


(a) original T1 MR image

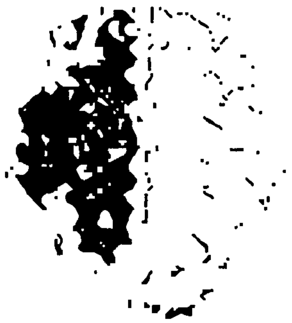
Black pixels in this view (which are white in the LHS of panel (a)) are the tumor pixels



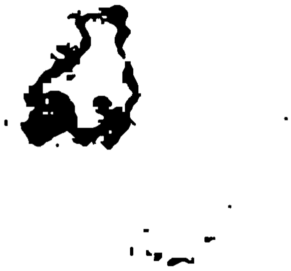
(b) Radiologist ground truth obtained by hand labeling



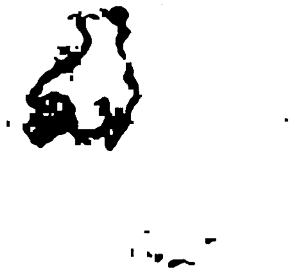
(c) tumor estimate from a k-nn segmentation



(d) X_m^0 = pathology mask created with crisp rules



(e) X_m^1 = Seg(X_m^0) by FCM



(f) X_m^2 = Seg(X_m^1) with VGC/FCM

Figure 5.31 Several approaches to image segmentation compared

Acquisition of the training data for the computational schemes was done as follows. First, the operator selected a subset of training pixels from each tissue class (approximately 50 pixels per class, by eye). These training data were used by the k-nn classifier to label the remaining unlabeled pixels in the source image. If visual evaluation of the result was accepted, the training data were fixed. If the segmentation was judged unsatisfactory, the procedure was repeated with new training data until an "optimized" k-nn segmentation was found, and the training data that produced it were taken as the training data for this image. Figure 5.31(c) shows the tumor region estimated by segmenting X into $c = 7$ regions (tissue classes) by this supervised, operator-optimized scheme using a crisp k-nn rule of the type given in Section 4.4.

Most of the remaining views in Figure 5.31 are based on the following steps. First, an initial segmentation of X is made by clustering it into c=7 classes with unsupervised FCM. In this, as well as all successive views that utilize the FCM algorithm, the basic parameters are $m = 2$, the Euclidean norm for both J_m and (successive prototypes termination norm) E_t, and $\varepsilon = 0.001$. The KB system (which has itself been trained with other input image data) removes the skull tissue and air classes using crisp rules. The rule base is divided into sub-blocks that have different rules for different parts of the human brain. Structure in the upper slices is represented by 40 rules, and 83 rules are used for the lower slices. Stage 5 is the final thresholding on the T1 image, and has 31 additional rules that pertain to all slices. Rules in this system are not fuzzy; they are crisp rules implemented in the CLIPS rule-based expert system shell (Giarratano and Riley, 1994). Rules for the intra-cranial mask (Figure 5.31(d)), for example, use histograms of the MR bands for pixels in the mask in crisp rules of the following form:

```
IF      T1(i,j) > T1 histogram peak
AND     ρ(i,j) > ρ histogram peak
THEN    keep (i,j) as possible tumor
ELSE    mark (i,j) as non-tumor
```

For another example, given an 8-connected component image of candidate tumor pixels and a known tumor region, a typical crisp rule looks like

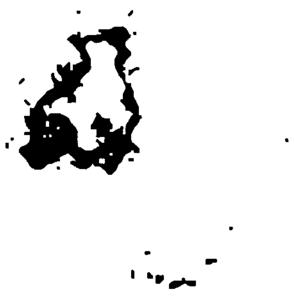
```
IF T1   mean value of region i is within one standard
        deviation of the mean of the known T1 tumor region
        in the T1 feature spectrum
THEN    keep region i as tumor
ELSE    remove region i from tumor list
```

Rules of this type provide an initial segmentation of the tumor pixels from remaining tissue classes that have already been isolated in previous stages.

At this stage the KB system identifies the patient as abnormal and removes what it believes to be the CSF, white matter and gray matter and extracranial pixels. This leaves the KB mask X_m^0 shown in Figure 5.31(d), which is the set of (mostly) pathological pixels in the image. The vectors associated with X_m^0 are a reduced image which is believed to contain the suspicious region which is now reclustered into c = 5 classes using various techniques.

View 5.31(e) shows the results of segmenting X_m^0 using unsupervised FCM at c = 5, followed by hand labeling of the pixels in the resultant segmentation by a human operator. You can see a number of islands in the southeastern quadrant of this output that are mistakes. Let X_m^1 denote the pixel vectors associated with the spatial locations in 5.31(e). View 5.31(f) shows the results of reclustering X_m^1 using *validity guided (re)-clustering* (VGC, see Bensaid et al., 1996b). It is pretty hard to see any improvement, but the number of false positives in 5.31(f) - as measured against the ground truth in 5.31(b) - is reduced slightly by VGC. However, the island mistakes persist.

Training data set X_m^0 is processed by the ssFCM algorithm to create view 5.31(g), the pixels of which we call X_m^3. Figure 5.31(h) is the output obtained by applying the KB to X_m^3. This view compares well with the ground truth in view 5.31(b). Most of the southeastern error islands are eliminated, but there are few new islands sprinkled around and in closer to the tumor mainland.


(g) X_m^3 = Seg(X_m^0) by ssFCM

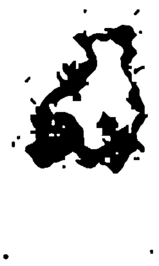

(h) Seg(X_m^3) by the KB system

Figure 5.31 (con't.) Several approaches to image segmentation

Clark et al.'s (1998) system comprises 6 stages. The first step is still unsupervised segmentation of MR slices with FCM. Initial FCM segmentations are used two ways. First, brain tissue regions are separated from extracranial clusters, and crisp morphological operations are used to clean up the initial tumor segmentation. The FCM output is also used to create the intracranial mask (Figure 5.31(d)). The remaining stages apply to the mask, and can be broadly lumped together as tumor recovery through region analysis of the tissues that have been retained, which consist of tumor and other tissue classes. Very roughly, initial tumor segmentation is done with adaptive histogram thresholding, which is then refined by "density screening", and then removal of the regions that do not contain tumor.

The system described by Clark et al. (1998) is completely automatic - no human intervention is needed on a per volume basis. The system, trained on 3 sets of MR slices, and tested on 13 new sets of unseen slices, almost replicates the radiologist ground truth in many of the test cases. Next we provide an example of simple fuzzy rule generation by returning to the fuzzy aggregation network discussed in Section 4.7. The following example using the FAN is adapted from Krishnapuram and Rhee (1993a, 1993b).

Example 5.10 We return to a 200×200 subset of the image shown in Figure 5.27(a), an outdoor scene consisting of three regions, "road", "sky", and "vegetation". Two texture features computed from 15×15 windows over the input image, homogeneity and entropy (Haralick et al., 1973), along with intensity (gray level) were used in this experiment. One hundred samples from each of the three regions were used to represent the classes in the training data.

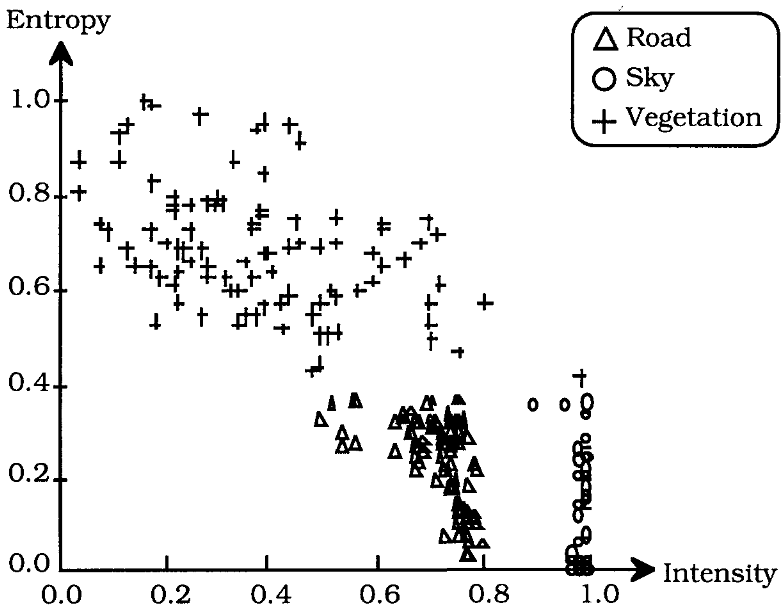


Figure 5.32 Views of pairwise features for the outdoor scene

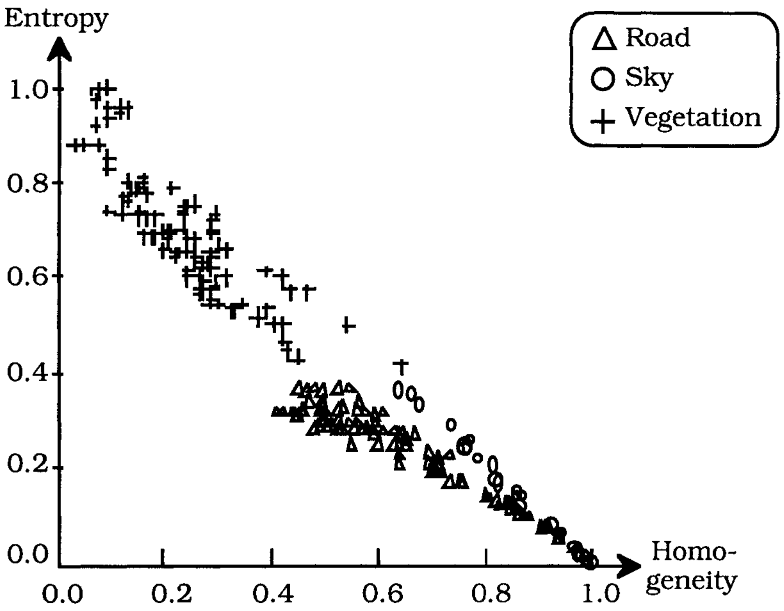


Figure 5.32 (con't.) Views of pairwise features for the outdoor scene

Figure 5.32 scatterplots two of the three sets of 2D features from the original 3 features for the outdoor scene. The top view in Figure 5.32 plots intensity against entropy, and in this view the three classes are fairly well separated (visually, perhaps even linearly). The bottom view in Figure 5.32 shows that the vegetation is still fairly separate from the other two classes, which seem somewhat more mixed in this pair of features. Bear in mind that these plots are for the training data, so good separability is not too surprising.

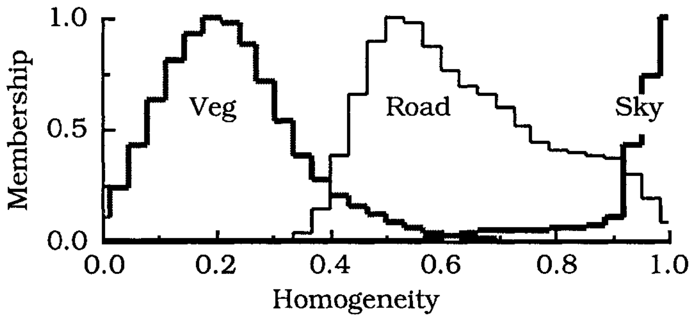
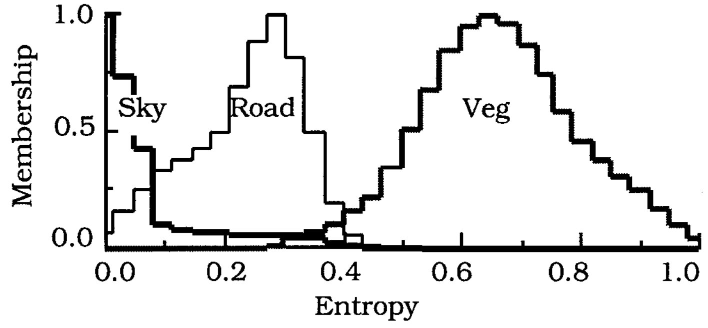


Figure 5.33 Smoothed histograms of the three features



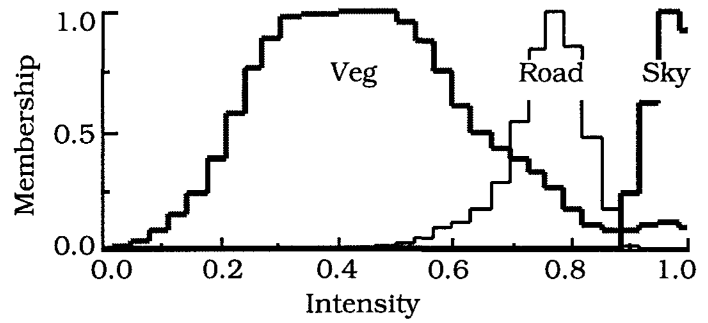


Figure 5.33 (con't.) Smoothed histograms of the three features

Figure 5.33 shows the smoothed histograms of the three features for the training data. These graphs certainly resemble mixtures of Gaussians with well separated means. The sky is the region of high brightness and homogeneity with low entropy, and in this example it is not hard to simply postulate reasonable rules without further processing.

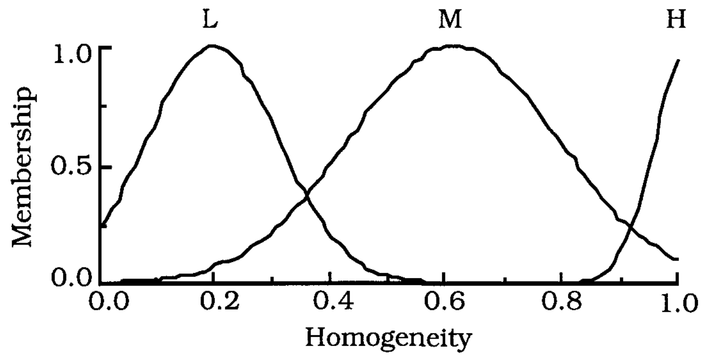


Figure 5.34 Linguistic labels for homogeneity

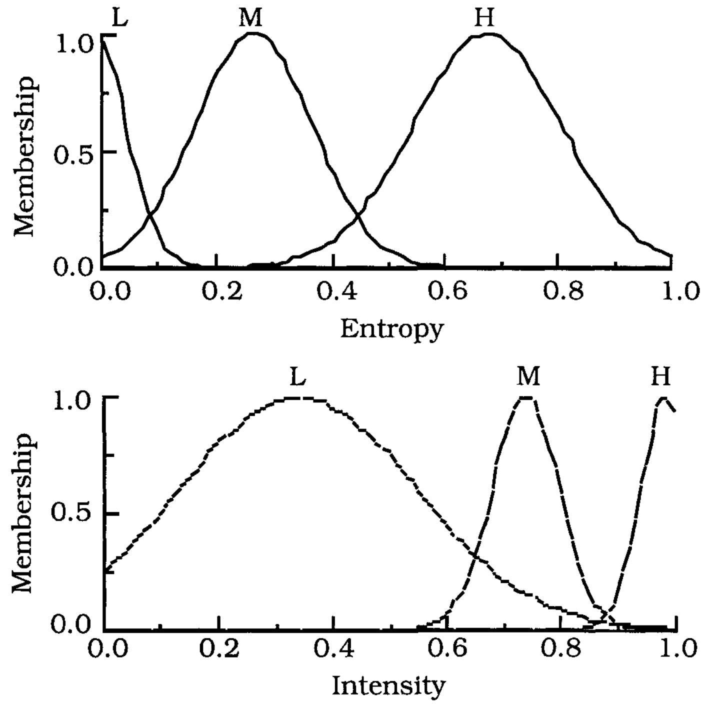
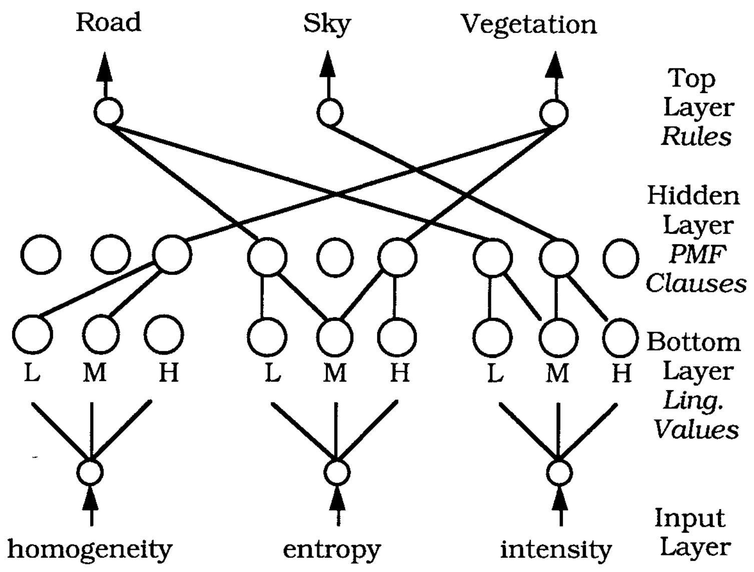


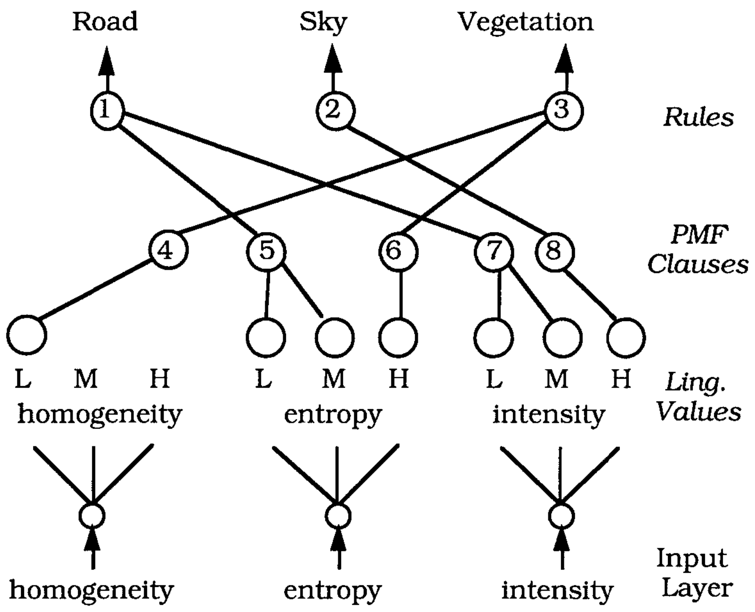
Figure 5.34 (con't.) Linguistic labels for entropy and intensity

Figure 5.34 shows the membership functions obtained by fitting each composite histogram (obtained by adding the individual histograms corresponding to road, sky and vegetation) in Figure 5.33 by a mixture of normal distributions for the linguistic labels that describe the three features. First, the histogram was approximated by a polynomial to determine the number of Gaussians: then, gradient descent was used to approximate the histogram by a Gaussian mixture. See Krishnapuram and Rhee (1993a) for more details on the fitting procedure.

Figure 5.35 shows the results of rule generation and redundancy detection using the method described in Section 4.7.E. View (a) shows the initial network, and note that it is not fully connected. This is due to the small α-cut criterion discussed in Section 4.7. Here, as there, α was 0.05. View (b) shows the network at the termination of training, where three of the initial connections between the input layer and the hidden layer have been pruned from the network because their connection weights were small (less than 0.01).



(a) initial network



(b) pruned network at termination

Figure 5.35 Rule generation and redundancy detection using a FAN

In this case the generalized mean in equation (4.103a) was used as the aggregation operator at the top and hidden layers. Values for the exponent q at nodes 1, 5 and 7 were -0.17, 7.52 and 7.51, respectively. Interpreting node 1 as conjunctive, and nodes 5 and 7 as disjunctive leads to the rule R_{road} below. The other two rules are inferred from values of the network parameters in a like manner. The rules obtained from the final network are similar to those an expert might elaborate. Here they are:

R_{road} :	IF *entropy* is (L OR M) AND *intensity* is (L OR M) THEN the class is road
R_{sky} :	IF intensity is H THEN the class is sky
R_{veg} :	IF *homogeneity* is L OR *entropy* is H THEN the class is vegetation

As will be discussed in Section 5.8, spatial relations between objects play an important role in computer vision. Krishnapuram and Rhee (1993b) show how rules involving the relations LEFT-OF, RIGHT-OF, ABOVE and BELOW can be generated using a similar approach.

5.6 Boundary Description and Surface Approximation

Boundary description can be viewed as an alternative approach to intensity image segmentation. In this approach, an edge operator (including those described in this chapter) is applied to the image to detect edge elements. The edge elements are considered to be parts of boundaries between various objects or regions in the image. The boundaries are then compactly described in terms of analytical curves such as straight lines, second-degree curves, and other more complex curves. For segmentation of range images, the edge detection step can be bypassed, and parametrized surfaces can be fitted directly to the raw range data. This process, known as surface approximation, generates a compact description of the objects present in the range image in terms of parametrized surfaces. The parametrized description of object boundaries or surfaces can be used at a higher level for view-independent object recognition and image understanding. There are many non-fuzzy methods that exploit the ideas of range image segmentation and surface approximation (cf. Hoffman and Jain, 1987, Besl and Jain, 1988, Yokoya and Levine, 1989).

The boundary and surface description problem can be stated as follows: fit parametrized curves/surfaces to an unsegmented data set. This problem is exacerbated by the following facts: (i) the number of segments (i. e., curves/surfaces) is usually unknown, and (ii) the edge or range data may be noisy and sparse. There is a plethora of techniques to fit parametrized curves such as conics to segmented edge pixels and to fit parametrized surfaces to segmented range data. However, segmentation of edge and range data is difficult in the case of jagged edges and noisy or sparse range data, since features such as gradients and curvatures cannot be computed reliably. A better approach in such situations would be to perform segmentation and boundary/surface fitting simultaneously on the data, without making use of features that assume continuity and smoothness of the edges and surfaces. Clustering based on non-point prototypes as discussed in Section 2.4 (lines, planes, hyperplanes, quadric shells, rectangles, etc.) is ideally suited to this approach, since it can perform segmentation and fitting simultaneously.

The shell clustering approach to boundary description and surface approximation has several advantages over traditional methods. It requires far less computation and memory compared to algorithms such as the generalized Hough transform (Hough, 1962, Ballard, 1981). Since it looks for global structures and does not use edge following or region growing, it is insensitive to local aberrations and deviations in shape. It does not use features such as gradients and curvatures and hence is not sensitive to noise and sharp discontinuities at the boundaries. Moreover, it is possible to robustify shell clustering algorithms by using a possibilistic approach, as discussed in Section 2.4.

Shell clustering algorithms have the drawback that the number of clusters present in a data set needs to be determined. Traditionally, the number of clusters is determined by evaluating a global validity measure of the c-partition for a range of c values, and then picking the value of c that optimizes the validity measure in some sense (Section 2.5). However, this is a very tedious and computationally expensive process, since the data must be clustered for each value of c. Moreover, in the case of shell clustering, the algorithms frequently converge to local minima, particularly when the data is complex. When the c-partition corresponds to a local minimum rather than a global one, the computed validity measure for the given value of c will not be correct. This can lead to a wrong choice of c. Sometimes these problems can be avoided with dynamic validity methods such as "compatible cluster merging" or "progressive clustering". However, even these methods have an internal measure of cluster validity that can be fooled, so don't expect validity miracles; instead, temper your judgment with suspicion and always look for satisfactory performance.

The compatible cluster merging approach begins clustering with a (presumably) overspecified number of clusters and then merges clusters that meet certain compatibility conditions. In the progressive clustering approach, after convergence of the clustering algorithm with an overspecified number of clusters, "spurious" clusters are eliminated, compatible clusters are merged, "good" clusters are identified, and points belonging to the good clusters are temporarily removed from the data set. Then clustering is performed again with the reduced number of clusters and data points. This procedure is repeated until no good clusters can be removed or until no data points are left. Unlike the traditional cluster validity approach which uses a global validity measure to evaluate the overall c-partition of the data set, the progressive approach uses individual cluster validity measures that evaluates the goodness or spuriousness of a particular cluster. In the remainder of this section we discuss boundary/surface description techniques based on these ideas.

A. Linear Boundaries and Surfaces

The early work in the area of linear/planar cluster detection was done by Bezdek et al. (1981a, 1981b, 1985). Anderson et al. (1982) and Davé (1989) extended this work. Krishnapuram and Freg (1992) showed that the Gustafson-Kessel (1978) algorithm can also be used to find linear and planar structures in data sets. They proposed a *compatible cluster merging* (CCM) algorithm to find the "optimal" number of line/plane segments in a data set. The CCM algorithm is applied after the GK algorithm (see Section 2.4) is run on the data set with a (hopefully) overspecified number c_{max} of clusters. The CCM algorithm merges compatible clusters among the c_{max} clusters to obtain the final result. This algorithm can be summarized as follows. Let $\mathbf{v}_i$ and $\mathbf{v}_j$ in $\Re^p$ be the point prototype centers of clusters i and j; let $\{\lambda_{i1},\ldots, \lambda_{ip}\}$ and $\{\lambda_{j1},\ldots, \lambda_{jp}\}$ be the eigenvalues of the fuzzy covariance matrices C_i and C_j at (2.27) of clusters i and j, arranged in descending order; and let $\{\mathbf{e}_{i1},\ldots, \mathbf{e}_{ip}\}$ and $\{\mathbf{e}_{j1},\ldots,\mathbf{e}_{jp}\}$ be their corresponding eigenvectors. Clusters i and j are said to be *compatible* if the following three conditions are all satisfied.

$$\left|\left\langle \mathbf{e}_{ip}, \mathbf{e}_{jp} \right\rangle\right| \ge c_1 \qquad ; \qquad (5.35a)$$

$$\left|\left\langle \frac{\mathbf{e}_{ip}+\mathbf{e}_{jp}}{2}, \frac{\mathbf{v}_i-\mathbf{v}_j}{\|\mathbf{v}_i-\mathbf{v}_j\|} \right\rangle\right| \le c_2 \qquad ; \text{ and} \qquad (5.35b)$$

$$\|\mathbf{v}_i-\mathbf{v}_j\| \le c_3\left(\sqrt{\lambda_{i1}}+\sqrt{\lambda_{j1}}\right) \qquad . \qquad (5.35c)$$

Equation (5.35a) ensures that the hyperplanes are parallel, and the constant c_1 should be chosen close to 1; (5.35b) ensures that the line joining the cluster centers is approximately orthogonal to the normals of the two lines (planes), and the positive constant c_2 should be chosen close to zero; and finally, (5.35c) verifies that the cluster centers touch each other if they are uniformly distributed. Krishnapuram and Freg suggest that the appropriate range for c_3 is [2, 4]. The three constants in (5.35) are user defined, and the utility of

CCM as a dynamic clustering algorithm - that is, its ability to terminate at a satisfactory number of clusters - is largely dependent on good choices for these parameters. Clustering and merging can be done either in one pass or iteratively. The iterative merging algorithm consists of running the GK algorithm followed by the CCM algorithm repeatedly, and stopping when no more clusters can be merged. A value of 1.5 is recommended for the fuzzifier m while running the GK algorithm.

Figure 5.36 illustrates the geometric conditions for merging two clusters that equations (5.35) attempt to enforce. Clusters A_1 and A_2 do not satisfy condition (5.35a). Clusters A_1 and A_5 do satisfy condition (5.35a), but not condition (5.35b). Clusters A_1 and A_3 satisfy conditions (5.35a) and (5.35b), but not condition (5.35c). A_3 and A_4 are the only clusters that can be merged in this case.

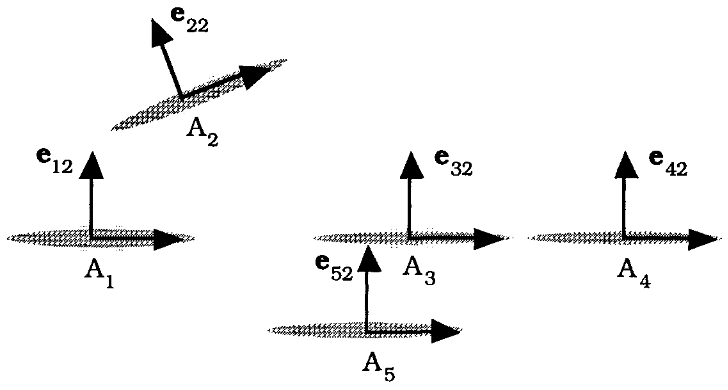
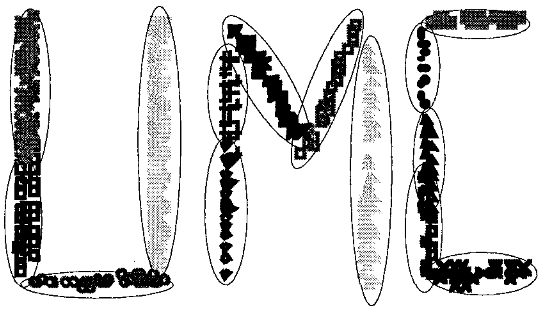


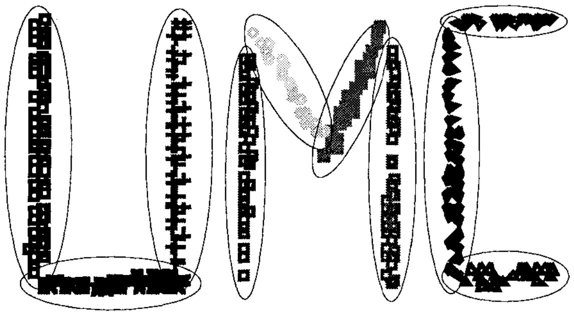
Figure 5.36 Conditions for merging two clusters

An approximate value for c_3 in (5.35c) can be derived for the 2-D case. Assume that the projections of points in two touching compatible linear clusters are uniformly distributed along intervals L_i and L_j that contain the projected points. Variances of the projected clusters are $L_i^2/12 = \lambda_{i1}$ and $L_j^2/12 = \lambda_{j1}$, and the distance between their cluster centers is $\|\mathbf{v}_i - \mathbf{v}_j\| = (L_i + L_j)/2 = \sqrt{3}\left(\sqrt{\lambda_{i1}} + \sqrt{\lambda_{j1}}\right)$. Hence c_3 should be at least $\sqrt{3}$, and preferably, a little larger.

Example 5.11 Figure 5.37 shows the results of using the GK algorithm with the CCM method on a data set containing samples from the characters "UMC". The GK algorithm was started with c_{max}=14. There are 14 clusters in Figure 5.37(a) before merging. In a color display, it is very easy to see the 14 clusters, but in this black and white display it is difficult to see them, so we have manually added hand-drawn ellipses that roughly capture the 14 clusters (after hardening). These are *not* the ellipses you could draw using the eigenstructure of the fuzzy GK covariance matrices.



(a) linear GK clusters



(b) after merging with CCM

Figure 5.37 Description of linear clusters by CCM

The results for the letters "UMC" after merging with CCM are shown in Figure 5.37(b). Again, we have enhanced the c =10 clusters in this final result by bounding them with manually inserted ellipses so you can see them. As you can see, CCM merges the two collinear clusters in the left vertical stroke of the "U", the two collinear clusters in the left vertical stroke of the "M", and the three collinear clusters in the vertical stroke of the "C". The values of c_1, c_2 and c_3 used for the outputs shown in Figure 5.37(b) are 0.95, 0.05 and 3.0 respectively; these values were chosen by trial and error. Results similar to those shown in Figure 5.37 can also be obtained with linkage-type clustering algorithms (Section 3.3); and by the boundary-hunter algorithm discussed in Bezdek and Anderson (1985).

Figure 5.38 shows the results of using the GK algorithm with the CCM method on a range image of a block obtained from the *Environmental Research Institute of Michigan* (ERIM). In this case, $c_{max} = 9$, $c_1=0.9$, $c_2=0.1$ and $c_3=4.0$.

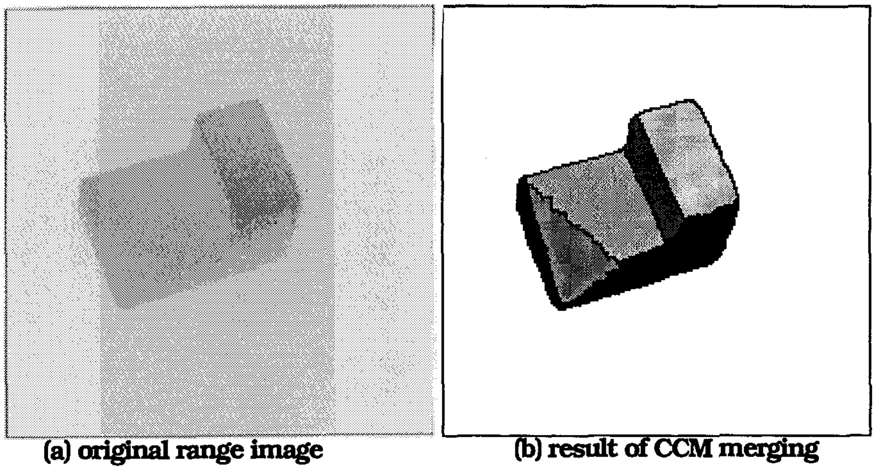


Figure 5.38 Approximation of linear surfaces by CCM

Comparing the processing described for Figures 5.37 and 5.38, we see that like most algorithms, judicious selection of $\{c_i\}$, the parameters of CCM, is needed to obtain satisfactory results for a particular data set. In both of these examples, all three parameters do satisfy the general recommendations that c_1 be close to 1, c_2 be close to zero, and c_3 be in [2, 4], but their individual values in these two examples are all different. There is a short dark line on the front face of the block in panel 5.38(b): this is a small cluster, that appears here due to many "noise" points in this region in the range image.

Hoeppner (1997) proposed a *fuzzy c-rectangular shells* (FCRS) algorithm to detect rectangles and lines in digital images that is very similar to the NISP clustering algorithm in Section 2.4. Since the history of shell clustering algorithms has shown that Euclidean distance is useful in many instances, Hoeppner's FCRS model uses a (nearly) Euclidean distance measure that still allows the direct computation of prototypes for use in an AO algorithm.

The contour of a rectangle can be assembled by four lines, each described by a normal equation $\langle \mathbf{x}-\mathbf{p},\mathbf{n}\rangle = 0$, in which $\mathbf{p}$ is a point on the considered line in $\Re^2$ and $\mathbf{n}$ is a vector perpendicular to the line. If $\mathbf{n}$ is a unit normal vector, the expression $|\langle \mathbf{x}-\mathbf{p},\mathbf{n}\rangle|$ yields the

Euclidean distance of a point **x** to the line (see Figure 4.4). This is illustrated in Figure 5.39 for the line L_1 through the point $\mathbf{p}_1$ with unit normal $\mathbf{n}_1$. Let the center of the rectangle be **v** and let $\mathbf{r} = (r_0, r_1)$, where the edge lengths of the rectangle are $2r_0$ and $2r_1$ as in Figure 5.39. You should compare Figure 5.39 to Figure 2.12; for the appropriate choices of parameters, these two figures depict the same rectangle.

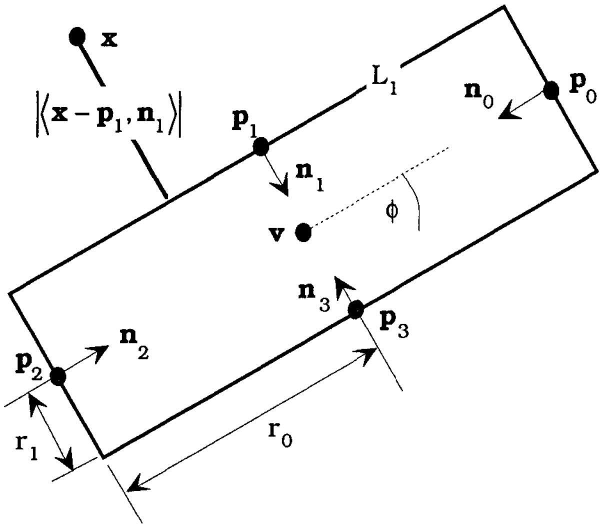


Figure 5.39 Hoeppner's rectangular shell prototype

Let ϕ be the angle between the positive x-axis and the first side of the rectangle encountered by counterclockwise rotation of the positive x axis (see Figure 5.39). The triple $(\mathbf{v}, \mathbf{r}, \phi)$ characterizes the rectangle completely; points on the rectangle will be denoted by rect($\mathbf{v}, \mathbf{r}, \phi$). The lines that form the edges of the rectangle are enumerated counterclockwise, beginning with zero at the right line (ϕ = 0 assumed). The points $\{\mathbf{p}_i\}$ and normal vectors $\{\mathbf{n}_i\}$ of the lines are numbered in the same way. For the unit normal $\mathbf{n}_k = (-\cos(\phi+k\pi/2), -\sin(\phi+k\pi/2))^T$, we require $\langle \mathbf{v}-\mathbf{p}_k, \mathbf{n}_k \rangle > 0$. In this way, all four normal vectors are directed towards the center of the rectangle.
The point $\mathbf{p}_i$ can be replaced by $\mathbf{p}_i=\mathbf{v}-r_{i \bmod 2}\,\mathbf{n}_i$, where $r_i = r_0$ or r_1. Since the normal vectors point towards the center of the rectangle **v**, the orthogonal distance from any point **x** to side i of the rectangle is

$$\Delta_i(\mathbf{x}, \mathrm{rect}(\mathbf{v}, \mathbf{r}, \phi)) \equiv (\mathbf{x} - \mathbf{p}_i)^T \mathbf{n}_i = (\mathbf{x}-(\mathbf{v}-r_{i \bmod 2}\mathbf{n}_i))^T \mathbf{n}_i = (\mathbf{x}-\mathbf{v})^T \mathbf{n}_i + r_{i \bmod 2}.$$

Δ_i yields a positive value for i = 0, 1, 2, 3 only if the vector **x** lies within the rectangle. Outside the rectangle Δ_i is negative for at least one i.

Given a vector **x**, the minimum of the four orthogonal distances $\{(\Delta_i(\mathbf{x};\ \text{rect}(\mathbf{v}, \mathbf{r}, \varphi))\}$ is positive/zero/negative according as **x** lies inside/on/outside the rectangle. The absolute value of the minimum of these four distances is the Euclidean distance between **x** and the nearest edge of the rectangle. Therefore, Hoeppner defines the FCRS distance measure as $\delta_{FCRS}(\mathbf{x}, \text{rect}(\mathbf{v}, \mathbf{r}, \varphi)) = \left| \underbrace{\min}_{i=0,1,2,3} \left\{\Delta_i(\mathbf{x}, \text{rect}(\mathbf{v}, \mathbf{r}, \varphi)\right\}\right|$.
Note that the lines are clipped (to the true edges) by the min-function, so they do not have infinite extents like they do in the FCV model (see Section 2.4). This distance is substituted for D_{ik} in equation (2.24a), and the weight vector **w** in (2.24a) is the zero vector. Thus, Hoeppner seeks minima of the function

$$J_m^{FCRS}(U, \mathbf{B}) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik}^m \delta_{FCRS}^2\left(\mathbf{x}, \text{rect}(\mathbf{v}_i, \mathbf{r}_i, \varphi_i)\right),$$

where $\boldsymbol{\beta}_i = (\mathbf{v}_i, \mathbf{r}_i, \phi_i)$ are the parameters of the i-th rectangular prototype. The use of the minimum function by δ_{FCRS} prevents us from finding the prototype parameters explicitly because they are arguments of the minimum function. In the case of rectangles it is insufficient to identify only the rectangle (cluster) to which a point **x** belongs with a certain membership degree. We also need to associate one of the four edges of the identified rectangle with **x**. An initial fuzzy partition of the data can be arbitrarily defined. The second (possibly hard) partition is produced by associating each **x** in the data with the line that is responsible for the minimum value of $\Delta_i(\mathbf{x}, \text{rect}(\mathbf{v}, \mathbf{r}, \varphi))$. By generating the second partition in this way, we actually rewrite the minimum function in another form. Using the Kronecker delta function $_K\delta$ ($_K\delta_{i,j}$=1 if i=j, $_K\delta_{i,j}$=0 otherwise) we define for i=0, ..., 3 four functions called $\text{minh}_i : \Re^4 \mapsto \{0, 1\}$. The action of minh_i is

$$\text{minh}_i(a_0, a_1, a_2, a_3) \equiv {}_K\delta_{i,s} \ni a_s = \min\{a_0, a_1, a_2, a_3\}.$$

For example $\text{minh}_2(7,8,4,6) = 1$ and $\text{minh}_s(7,8,4,6) = 0$ for s = 0, 1, 3, because the minimum of a_0, a_1, a_2 and a_3 is a_2 in this example. If multiple a_i are minimal, we can randomly choose only one s in { j : a_j=min{a0, a1, a2, a3} }, i.e., the minimum function must satisfy the constraint $\sum_{s=0}^{3} \text{minh}_s(a_0, a_1, a_2, a_3) = 1$. This constraint leads to the equality $\min\{a_0, a_1, a_2, a_3\} = \sum_{i=0}^{3} a_i \text{minh}_i(a_0, a_1, a_2, a_3)$. Hoeppner interprets $\text{minh}_s(a0, a1, a2, a3)$ as the grade of minimality of a_s with respect to a_0, a_1, a_2 and a_3.

For s = 0, 1, 2, 3 let $\Delta_s(\mathbf{x}_j, \boldsymbol{\beta}_i)$ be the (directed) distance of the j-th data point $\mathbf{x}_j$ to the s-th side of the i-th FCRS prototype $\boldsymbol{\beta}_i = (\mathbf{v}_i, \mathbf{r}_i, \varphi_i)$, and define,

$$u_{i,j,s} = \text{minh}_s(\Delta_0(x_j, \boldsymbol{\beta}_i), \Delta_1(x_j, \boldsymbol{\beta}_i), \Delta_2(x_j, \boldsymbol{\beta}_i), \Delta_3(x_j, \boldsymbol{\beta}_i)). \tag{5.36}$$

Then $u_{i,j,s}$ denotes the crisp membership of $\mathbf{x}_j$ in edge s of cluster i. The matrices $U_s = [u_{i,j,s}]$, s = 0, 1, 2, 3, are four crisp c - partitions of the data that assign data vectors to the four rectangle edges.

With this notation, we can find closed-form equations for use in an AO algorithm to minimize J_m^{FCRS}. Unfortunately the use of crisp grades of minimality leads to a convergence problem, as a data vector might be assigned to different edges in an alternating fashion. To overcome this problem, Hoeppner (1997) replaces the crisp minimum functions $\{\text{minh}_i\}$ by fuzzy membership functions $\text{minf}_i: \Re^4 \to [0,1]$, which still satisfy $\sum_{s=0}^{3} \text{minf}_s(a_0, a_1, a_2, a_3) = 1$. (This constraint is required to avoid the trivial solution.) Hoeppner proposes two possible fuzzy minimum functions, and shows that the modified distance measures lead to only slight changes in the objective function. Therefore, the same prototype update rules as for the hard case can be used.

The chance of terminating at a local minimum with an algorithm to detect rectangles is quite large because if a depicted rectangle consists only of some of its edges, there are many possible rectangles that approximate the data vectors. Furthermore, if some edges are parallel they can easily be exchanged between different rectangle clusters, which leads to strong local minima (i.e., deep local minima that cannot be easily escaped).

Example 5.12 Figure 5.40 shows an example of FCRS clustering of a data set with 5 rectangles. The number of clusters c was specified to be 5. Although the data set in Figure 5.40 is pretty complicated, the FCRS algorithm discovers all five rectangles (of course, it is told to look for five). The fact that edges of different rectangles never lie parallel to each other makes the detection easier.

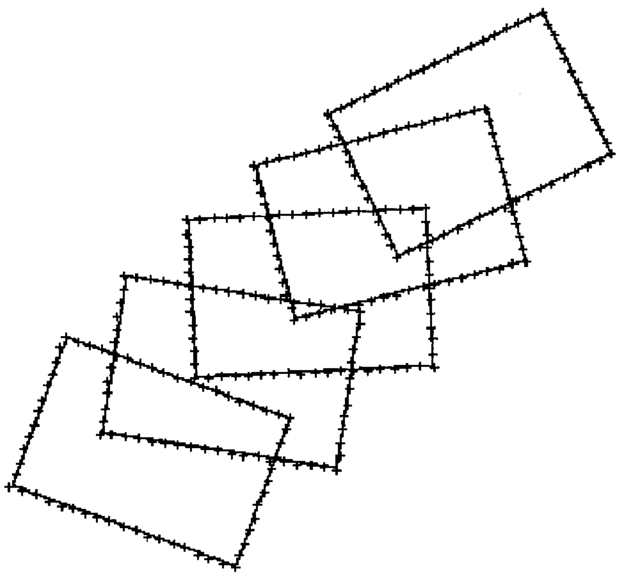


Figure 5.40 FCRS detects rectangles at different angles of rotation

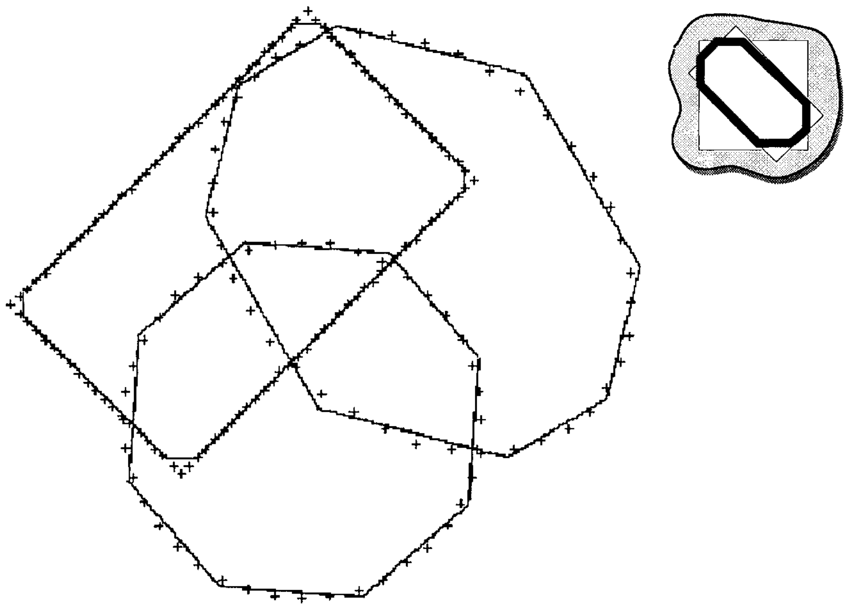


Figure 5.41 FC2RS approximates rectangles and complex shapes

In most applications images don't have a nice sequence of equal area rectangles such as those used in Figure 5.40. So, the ability to detect more complex shapes is useful, and often necessary. FCRS can be easily generalized to polygons other than rectangles. In the derivation of the FCRS algorithm, the normal vectors of the edges vary in steps of 90 degrees. By using another angle that is a divisor of 360, more complex shapes can be realized. This is illustrated in the *inset* of Figure 5.41, which shows an octagonal shell as the superposition of two rectangular shells. Hoeppner calls this the *fuzzy c two rectangular shapes* (FC2RS) model. (The name FC2RS originates from the visualization of the cluster's shape with the help of two rectangles.)

Complex polygons are appropriate for approximating circles or ellipses, so FC2RS might handle complicated scenes with rectangles, circles and ellipses correctly, as shown in Figure 5.41. Here an angle of 45 degrees is used to obtain a polygonal approximation of a data set containing a rectangle, a circle, and an ellipse.

Compare the data in Figure 5.40 to the data in Figure 2.13, which has a pair of overlapping rectangles. The difference between the two data sets shows you the main difference between the NISP and FCRS clustering algorithms. Once the norm is chosen for NISP, all c of the rectangles (diamonds for the 1-norm in Figure 2.13) have the same fixed orientation and side lengths; but in FCRS, each rectangle can have different orientations and side lengths. In this sense Hoeppner's FCRS clustering scheme stands to NISP as the GK clustering model stands to FCM; FCRS and GK are, at least in principle, able to adjust their prototypes to individual cluster structures, whereas NISP and FCM impose the topological structure of the norm used on all c clusters.

B. Circular Boundaries

Description of circular boundaries based on the *Fuzzy c-Shells* (FCS) and *Fuzzy c-Spherical Shells* (FCSS) algorithms (Section 2.4) can be found in (Davé, 1990b) and (Krishnapuram et al. 1992). Man and Gath (1994) contains examples involving the detection of ring-shaped clusters mixed with regular (cloud) clusters. Section 2.6 discusses validity issues related to circular as well as more general shell clusters. Here we present a technique called the *divide and conquer* (D&C) algorithm proposed by Davé and Fu (1994) for the detection of circular boundaries. The overall technique of which this is a part, called the *divide and conquer NFCS* (D&C-NFCS) algorithm, is summarized in Table 5.12. The D&C technique combines good features of the Hough transform with fuzzy shell clustering.

Table 5.12 The divide and conquer NFCS algorithm

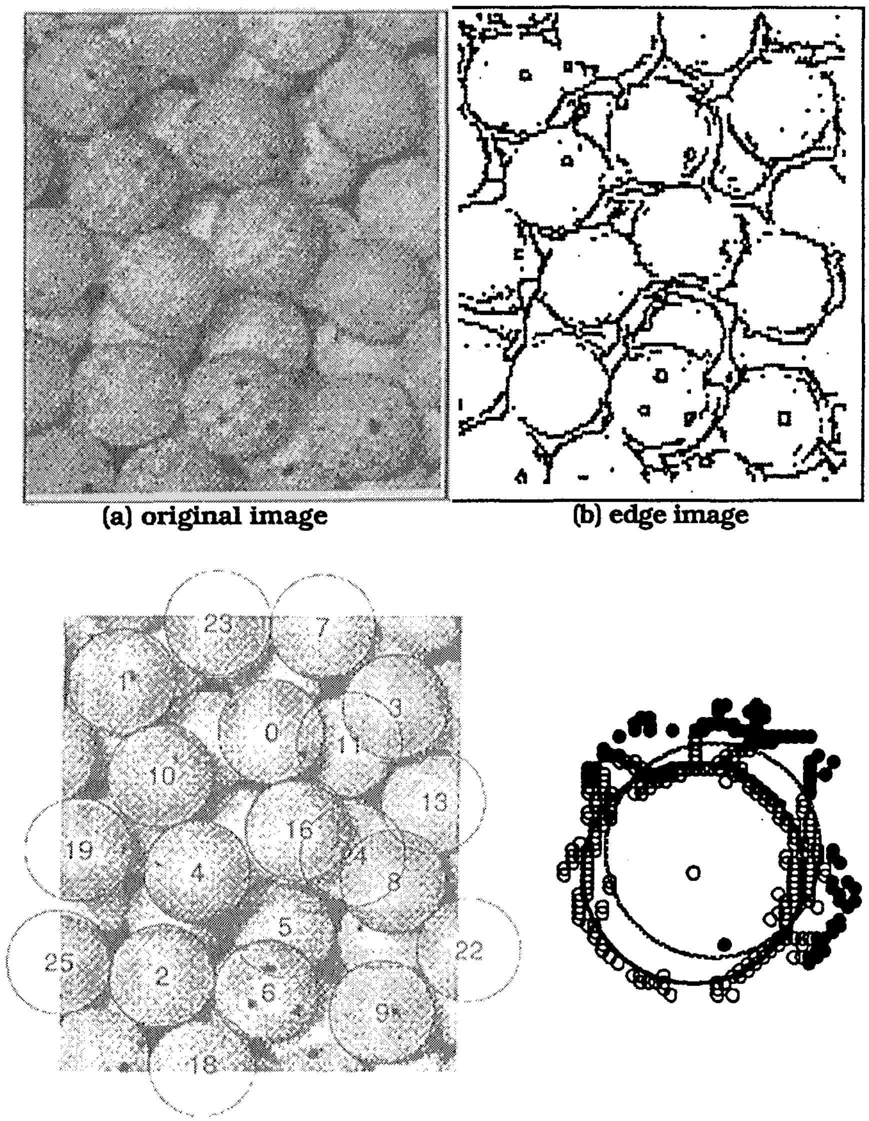
```
Store   Unlabeled object data $X \subset \Re^p$
Pick    ☛ Maximum number of clusters $c_{max}$
        ☛ Merger thresholds $\varepsilon_1$ and $\varepsilon_2$
        ☛ $ACC_{min}$ = smallest Hough transform accumulator
          array value acceptable as a peak
Do      Set peak_parameters[i], i=1,...$c_{max}$, to zero
        Set DS_array[i], i=1,...$c_{max}$, to zero
        Fill accumulator array, ACC, using the Hough transform
        Set peak counter c = 0,
        Set PV = highest peak in ACC
        REPEAT UNTIL (PV < $ACC_{min}$ or c =$c_{max}$)
           increment c
           Record peak_parameters(c)
           Zero out the peak location in ACC and a small
              neighborhood of it
           Put points belonging to a small neighborhood of
              circle corresponding to peak c in DS_array[c]
           Assign value of the highest peak in ACC to PV
        END UNTIL
        FOR (i = 1 to c)
           Run NFCS (with # of clusters = 1) on DS_array[i] with
                  peak_parameters(i) as init. prototypes
           Record center $\mathbf{v}_i$ and radius $r_i$ of cluster i
           Compute a set of validity criteria, val(i)
        END FOR
        Initialize removal counter, $c_{rem}$ = 0;
        FOR (i = 1 to c)
           if ( val(i) not acceptable) remove cluster i, and
              increment $c_{rem}$
        END FOR
        c = c - $c_{rem}$; Re-initialize $c_{rem}$ = 0
        FOR (each pair ( i, j ) of clusters)
           if ( $\left(\|\mathbf{v}_i - \mathbf{v}_j\| < \varepsilon_1 \text{ and } \|r_i - r_j\| < \varepsilon_2\right)$
              Remove the cluster with lesser number of points;
              increment $c_{rem}$
        END FOR
        c = c - $c_{rem}$
```

The Hough transform approach is popular in computer vision because of its implementational simplicity and robustness in the presence of noise. However, if the objective is to find the location and size of circles with high accuracy, then the cost of the Hough transform is very high due to increased memory requirements and computations. It also suffers from other disadvantages such as bin splitting and occurrence of false peaks (Davé and Bhaswan, 1992; Davé and Fu, 1994). On the other hand, shell clustering methods (see

Section 2.4) provide fast and accurate parameter estimates with less memory, provided a good initialization for the prototypes is available. Hence, it is advantageous to apply the Hough transform with a coarse resolution to obtain rough estimates of the prototype parameters, and then apply a shell clustering algorithm such as FCS based on the initialization provided by the Hough transform to find more accurate parameter estimates.

In the shell clustering stage, the data points in the image can be divided into subsets such that each subset contains only those points that are in a small neighborhood of the circle corresponding to each peak in the Hough transform. A robust version of FCS is applied with c=1 to each subset. A robust version is required because each data subset can contain many extraneous noise points. Davé and Fu use the Noise Clustering (NC) approach (see Section 2.5) to robustify FCS. They call the resulting algorithm NFCS. Circle detection by NFCS is fast due to (1) the use of a good initialization, and (2) the use of only a small fraction of the whole data set. After the NFCS stage, clusters with similar parameters are merged and spurious clusters are removed to produce the final results.

Example 5.13 Davé and Fu (1994) applied the above algorithm to a problem of detecting spheres in a random packing. Figure 5.42(a) shows an image of spheres in a random packing. The edge map, after clean-up, is shown in Figure 5.42 (b). In Figure 5.42(b) there are 2596 points, and it is noisy due to shadow artifacts and poor image quality. There are also partial shapes due to hidden geometry. The final result of the above algorithm after applying cluster merging and removal is shown in Figure 5.42(c), where the found circles are superimposed on the original data set. It can be seen that all the spheres which are in the front (i.e. not hidden) are detected by this approach. Figure 5.42 (d) illustrates an individual cluster detection step. The dashed circle is the initialization from Hough transform, and the solid circle is the final result of NFCS. It's hard to see these two circles in Figure 5.42(d); the dashed circle is the one that is northeast (above and to the right) of the solid circle. The points plotted are the points in the subset used by NFCS. Small open circles are points classified as good points, and small filled circles are points classified as noise points. As can be observed in the cases shown, the visible fit obtained by NFCS is very good. Although the fit obtained by the Hough transform alone is not accurate, it is close enough to each correct circle so that the its neighborhood contains most of the good boundary points.



(a) original image **(b) edge image**

(c) circle superposition **(d) Hough (initialization and final result)**

Figure 5.42 Detection of circular boundaries using D&C-NFCS

NFCS does have the ability to handle noisy data, and its breakdown point is comparable to that of a robust M-estimator (Davé and Krishnapuram, 1997). D&C-NFCS may be modified for ellipses as well, by roughly estimating the parameters of the ellipses by a circle detecting Hough transform, and then using a robust version of an elliptical shell clustering algorithm such as AFCS or FCQS (see Section 2.4). Thus, this technique, which is computationally efficient and handles the problem of unknown clusters, has good potential for solving practical problems.

C. Quadric Boundaries/Surfaces

Examples of boundary description in terms of elliptical shell clusters based on the AFCS algorithm (see Section 2.4) can be found in (Davé and Bhaswan, 1992). However, the issue of unknown number of clusters is not addressed in that paper.

Krishnapuram et al. (1995a, b) describe an unsupervised clustering algorithm called *unsupervised boundary description* (UBD) for describing edges with quadric curves. This algorithm is based on the fuzzy c-quadric shells (FCQS) and possibilistic c-quadric shells (PCQS) algorithms described in Section 2.4. Before we describe the UBD algorithm, we briefly summarize a line detection algorithm which is part of the UBD algorithm.

Images often contain linear boundaries in addition to quadric ones, and this can pose a problem. However, the FCQS (or PCQS) algorithm can be used to find linear clusters, even though the constraint in (2.48) forces all prototypes to be of second degree. This is because FCQS can fit a pair of coincident lines for a single line, a hyperbola for two intersecting lines, and a very "flat " hyperbola, or an elongated ellipse, or a pair of lines, for two parallel lines. Hyperbolas and extremely elongated ellipses occur rarely in practice.

When the data set contains many linear clusters, the FCQS algorithm characterizes them variously as hyperbolas, extremely elongated ellipses, etc. In this case, we can group all the points belonging to such pathological clusters into a data set and then run a line finding algorithm such as the GK algorithm (see previous subsection) on this data set with an appropriate initialization. The parameters of the lines can be determined from the centers and the covariance matrices of the GK clusters.

The various conditions that need to be checked to determine the nature of β_i as well as the initialization procedures required by the line detection algorithm can be found in Krishnapuram et al. (1995a, b). Since the initialization is usually good, the GK algorithm often terminates in a couple of iterations and yields the parameters of the lines. The line detection algorithm is summarized in Table 5.13.

Table 5.13 The line detection algorithm

```
Store   Unlabeled Object Data $X \subset \Re^p$
-----------------------------------------------------------------
Init.   ☛ Set $\chi$, the set of data points in linear clusters to $\emptyset$
        ☛ Set number of lines c = 0
        ☛ Start with FCQS or PCQS output as initialization
-----------------------------------------------------------------
Do      FOR each cluster i with prototype parameters $\beta_i$
          IF $\beta_i$ is a pair of coincident lines THEN
            Add all points assigned to cluster i to $\chi$
            c=c+1
            Initialize new linear prototype as the one of
                two coincident lines
          IF $\beta_i$ is a non-flat hyperbola OR a pair of intersecting
            lines OR a pair of parallel lines THEN
              Add all points assigned to $\beta_i$ to $\chi$
              c=c+2
              Initialize new linear prototypes as asymptotes
              of the hyperbola or as individual lines making
              up the pair of lines;
          IF $\beta_i$ is an ellipse with large major to minor axis ratio
            THEN
              Add all points assigned to $\beta_i$ to $\chi$
              c=c+2
              Initialize new linear prototypes as two
              tangents to the ellipse at the two ends of the
              minor axis
          IF $\beta_i$ is a hyperbola with a very large conjugate axis to
            transverse axis ratio THEN
              Add all points assigned to $\beta_i$ to $\chi$
              c=c+2
              Initialize new linear prototypes as tangents to
              the hyperbola at its two vertices
        END FOR
        Run the GK algorithm on $\chi$ with c clusters using the
        above initializations for the prototypes
```

The UBD algorithm automatically determines the number of curves to be used in the description by progressively clustering the data starting with an overspecified number c_{max} of clusters. Initially, a possibilistic version of the FCQS algorithm, known as PCQS (see Section 2.4) is run with $c = c_{max}$. The weights w_i in the objective function (see (2.24a)) are all set to the square of the estimated thickness of the curves. In boundary description applications, this value is typically equal to 2. At this stage, spurious clusters are eliminated, compatible clusters are merged, good clusters are identified, and points with high memberships in good clusters are temporarily removed from the data set to reduce the complexity of the remaining data set. The PCQS algorithm is invoked again with the remaining feature points. This procedure is repeated until no more elimination, merging, or removing occurs.

Shell cluster validity measures such as shell thickness $\mathcal{V}_{ST}(U,\boldsymbol{\beta}_i)$, shell hypervolume $\mathcal{V}_{SHV}(C_{S_i})$, and surface density $\mathcal{V}_{SSD_{i2}}(U,C_i)$ are used to determine candidates for elimination, merger, or removal. These are validity measures for individual shell clusters rather than for the partition. Surface density measure $\mathcal{V}_{SSD_{i2}}(U,C_i)$ is defined at (2.124) in Section 2.6. The definition of shell thickness $\mathcal{V}_{ST}(U,\boldsymbol{\beta}_i)$ is

$$\mathcal{V}_{ST}(U,\boldsymbol{\beta}_i)=\frac{\sum_{k=1}^{n}u_{ik}^{m}\left\|\mathbf{t}_{ik}\right\|^2}{\sum_{k=1}^{n}u_{ik}^{m}} \quad , \qquad (5.37)$$

where $\boldsymbol{\beta}_i$ represents the prototype parameters of shell cluster i, and $\mathbf{t}_{ik}$ is the vector from $\mathbf{x}_k$ to the closest point on the shell prototype. Recall that the *shell hypervolume* $\mathcal{V}_{SHV}(C_{S_i})$ for shell cluster i is

$$\mathcal{V}_{SHV}(C_{S_i})=\sqrt{\det(C_{S_i})} \quad , \qquad (5.38)$$

where C_{S_i} is defined in (2.117). The conditions for elimination, merger, or removal are described below.

Cluster i is considered spurious if the sum of the memberships of all feature points in that cluster (cardinality) is very low or if the sum is low and the surface density is also low, i.e.,

$$\sum_{k=1}^{n}u_{ik}<n_{VL}, \quad \text{or} \quad \sum_{k=1}^{n}u_{ik}<n_L \text{ AND } \mathcal{V}_{SSD_{i2}}(U,C_i)<SD_L. \qquad (5.39)$$

The suggested values for the constants are: $n_{VL} \approx 2\%$ of the total number of data points n, $n_L \approx 4\%$ of the total number of points, $SD_L \approx 0.15$. To determine if two clusters are compatible so that they can be merged, the error of fit and the validity for the merged cluster are computed. To do this, all points having a membership greater than an α-cut in either of the two clusters i and j are collected. Let this data set be denoted by X_{ij}. In practice, a value of about $\alpha = 0.25$ works best, independent of the data set. Then the PCQS algorithm is run with c=1 on this data set. If the fit and surface density for the resulting cluster is good, the two clusters are merged. In other words, the condition for merging is

$$\mathcal{V}_{ST}(U,\beta_i) < ST_L \text{ AND } \mathcal{V}_{SSD_{12}}(U,C_i) > SD_H \quad , \tag{5.40}$$

where $\mathcal{V}_{ST}(U,\beta_i)$ and $\mathcal{V}_{SSD_{12}}(U,C_i)$ are the shell thickness and the surface density respectively of the cluster formed by X_{ij}. Suitable values for ST_L and SD_H for this application are about 2.0 and 0.7.

Cluster i is characterized as "good" if

$$\mathcal{V}_{SSD_{12}}(U,C_i) > SD_{VH} \quad , \text{ or} \tag{5.41a}$$
$$\mathcal{V}_{SSD_{12}}(U,C_i) > SD_H \text{ AND } \mathcal{V}_{SHV}(C_{S_i}) < SHV_L \quad , \tag{5.41b}$$

where SD_{VH} is a very high threshold for surface density, SD_H is the same value that was used for merging, and SHV_L is a low value for the fuzzy hypervolume. The second condition is designed to handle cases in which the surface density has borderline values. Suitable values for SD_{VH} and SHV_L are about 0.85 and 0.5 in this application. Points are temporarily removed from the data set if their membership in one of the "good" clusters is greater than $u_H = 0.5$.

In addition to the above steps, we need to identify noise points and temporarily remove them from the data set. Noise points are identified as those which have low memberships in all clusters, i.e., feature point $\mathbf{x}_k$ is removed if

$$\max_{1 \le i \le c}\{u_{ik}\} < u_L \quad . \tag{5.42}$$

Noise points have to be removed at the end of each run of the PCQS algorithm, because as the number of clusters decreases and points assigned to good clusters are removed, the number of noise points relative to the good points becomes high, making it difficult to detect the few good clusters that are left. A good choice for u_L is about 0.1. The condition for noise point removal applies only when possibilistic memberships are used, and not when fuzzy memberships are used. In the case of fuzzy memberships, (5.42) can be true even for good points if they are shared among many clusters. A similar comment applies to the removal of good points.

Several comments about the UBD algorithm are in order. First, spurious clusters are clusters that have a low validity measure, but not necessarily low cardinality. Next, the second pass in Table 5.14 is needed to improve the reliability of UBD - using one pass often terminates at an unattractive solution. Finally, there are a lot of thresholds to choose when you implement this algorithm. Conditions (5.39), (5.40) and (5.41) are used to decide whether to eliminate, merge or temporarily remove a cluster. All of these conditions are based on multiple validity measures. Although a single validity measure could be used to design each condition, the resultant algorithm would not be as reliable as it is when more than one measure is used (see Section 2.6 for a discussion on the reliability of a single validity measure). The UBD algorithm is summarized in Table 5.14.

Table 5.14 The unsupervised boundary detection (UBD) algorithm

Store	Unlabeled Object Data $X \subset \Re^p$
Pick	☛ Maximum number of clusters, c_{max}
Do	REPEAT UNTIL (No elimination, merging or removal takes place) Perform clustering using the PCQS algorithm (use FCQS for initialization) Run the Line Detection algorithm (Table 5.13) Eliminate spurious clusters and update c Merge compatible prototypes and update c Detect good clusters, save their prototypes in a list, remove points with high memberships in them and update c; Remove noise points END UNTIL Add removed feature points to X Append remaining clusters' prototypes from the last iteration in the above repeat loop to the list of removed clusters' prototypes and update c <u>Second Pass</u> REPEAT UNTIL (No more merging or elimination takes place) Perform the PCQS algorithm using the prototype list as initialization; Merge compatible prototypes and update c accordingly; Eliminate clusters with small cardinality, and update c accordingly END UNTIL

The use of multiple validity measures imposes a higher burden on the user when picking thresholds. However, in the boundary description case, most of the thresholds are (more or less) fixed by the nature of the edge images. For example, ST_L and SH_{VL} are determined by the expected thickness of the edges. Also, since the surface density $V_{SSD_{12}}$ is always between 0 and 1 it is fairly easy to pick SD_L, SD_H and SD_{VH}. In the experiments we know of, the UBD algorithm seems pretty insensitive to changes in these thresholds; most of them can tolerate as much as a 20% change without adverse effects on the results. Although changes in these thresholds may affect the sequence in which clusters are eliminated, merged, or temporarily removed, the final results are usually the same.

Example 5.14 Figure 5.43 (a) shows a 200×200 image of objects whose boundaries can be described by linear and second-degree curves.

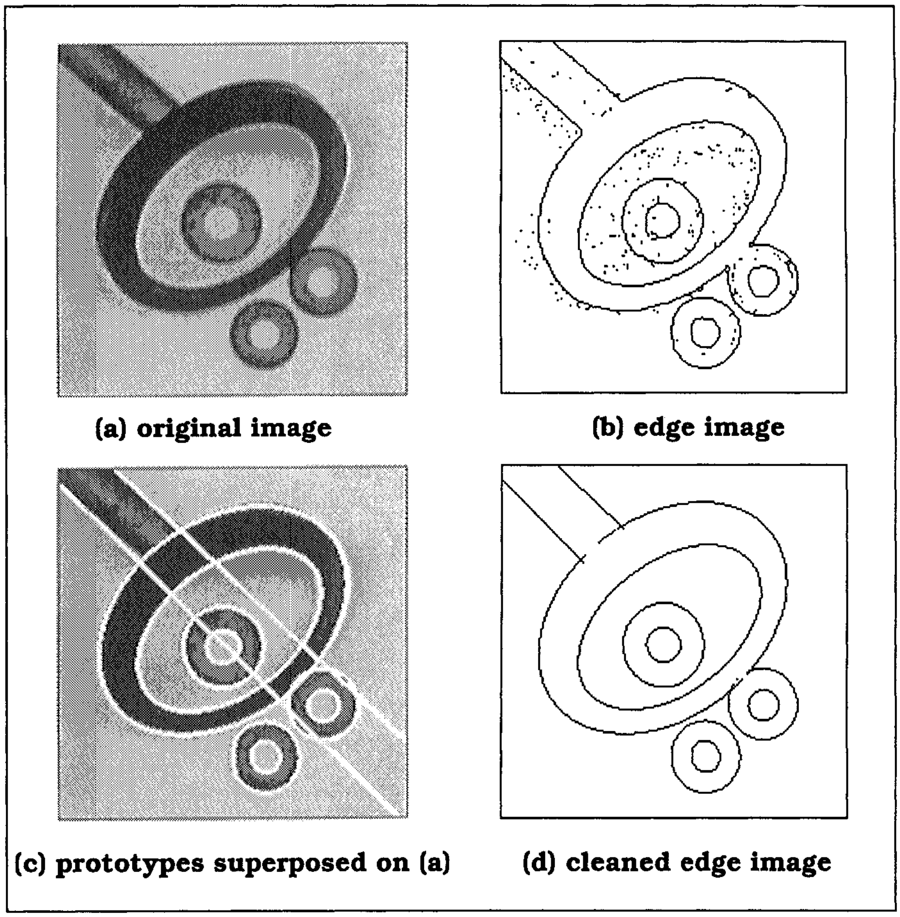


Figure 5.43 Description of quadric boundaries in edge images

Uniformly distributed noise on the interval [-15, 15] was added to the image intensity values. The object edges were then obtained by applying the Sobel operator and thresholding. The edge images were then thinned using a neural net thinning algorithm (Krishnapuram and Chen 1993). The thinning procedure is important because it makes all edges one-pixel thick, making it easier to pick the various validity thresholds in UBD. It also reduces the number of pixels to be processed. Figure 5.43 (b) shows the thinned image which is used as input to UBD. It can be seen that the boundaries are not always clean and there are many noise points. The image has about 2,000 edge points.

The boundary description algorithm was applied with the initial number of clusters c_{max} = 25. Figure 5.43(c) shows the final prototypes superimposed on the original image. The prototypes are shown three-pixels thick for emphasis. The "cleaned edge image" in Figure 5.43(d) is obtained by plotting the prototypes only in those regions where there were at least 2 edge pixels within a 3×3 neighborhood.

D. Quadric surface approximation in range images

Krishnapuram et al. (1995a, b) describe a *quadric compatible cluster merging* (QCCM) algorithm for surface approximation in range images. This algorithm starts with the initial planar approximation of a range image obtained by applying the CCM algorithm (see (5.35)). At this point, the following condition is checked for each pair of clusters.

$$\left\|\mathbf{v}_i - \mathbf{v}_j\right\| \le C_3\left(\sqrt{\lambda_{i1}}\cdot\left|\left\langle \mathbf{e}_{i1}, \hat{\mathbf{v}}_{ij}\right\rangle\right| + \ldots + \sqrt{\lambda_{jp}}\cdot\left|\left\langle \mathbf{e}_{jp}, \hat{\mathbf{v}}_{ij}\right\rangle\right|\right). \tag{5.43}$$

In (5.43) $\hat{\mathbf{v}}_{ij} = (\mathbf{v}_i - \mathbf{v}_j)/\left\|\mathbf{v}_i - \mathbf{v}_j\right\|$ is the unit vector in the direction of $\mathbf{v}_i - \mathbf{v}_j$. A suitable value for C_3 is $\sqrt{3}$. A pair of clusters satisfying (5.43) is considered "close".

The points belonging to each pair of "close" clusters are used as the input data set to the *possibilistic c-plano-quadric shells* (PCPQS) algorithm (see Section 2.4) with c=1. If the fit of the resulting quadric cluster is "good", then the two clusters are merged. The fit is considered "good" if the shell thickness measure $V_{ST}(U,\boldsymbol{\beta}_i)$ in (5.37) is less than a threshold ST_L. A suitable value for ST_L in this application is 0.1. Since the exact distance from a feature point to a surface is difficult to compute in the 3D case, the approximate distance at (2.55) is used to compute $\|\mathbf{t}_{ik}\|$ in (5.37). The QCCM algorithm is summarized in Table 5.15.

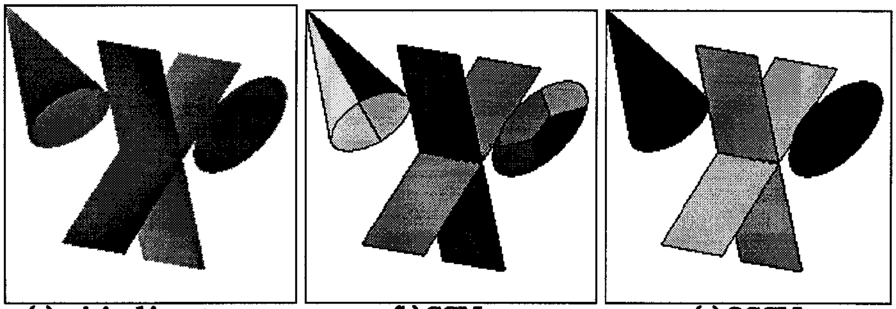
Table 5.15 The quadratic compatible cluster merging (QCCM) algorithm

Store	Unlabeled Object Data $X \subset \Re^p$
Pick	☛ Max. # of clusters, c_{max}; α-cut level α
Init.	☛ Use the CCM algorithm
Do	see below

Do

```
merge = TRUE
WHILE (merge = TRUE) DO
    merge = FALSE ;
    FOR each pair of clusters i and j DO
        IF clusters i and j satisfy condition (5.43) THEN
            $X_{ij} = \{\mathbf{x}_k : u_{ik} \wedge u_{jk} > \alpha\}$
            Run the PCPQS algorithm on $X_{ij}$ with c=1
            Estimate error of fit $\mathcal{V}_{ST}(U,\boldsymbol{\beta}_i)$ using (5.37)
            IF ($\mathcal{V}_{ST}(U,\boldsymbol{\beta}_i) < ST_U$) THEN
                $u_{ik} = \max(u_{ik}, u_{jk}) \quad \forall k$
                Eliminate cluster j and replace parameters of
                cluster i with parameters of combined cluster
                $c \leftarrow c-1$
                merge = TRUE
            END IF
        END IF
    END FOR
END WHILE
```

Example 5.15 Figure 5.44 (a) shows a 200×200 synthetic range image consisting of 2 planes, a right circular cone, and an ellipsoid.



(a) original image (b) CCM (c) QCCM

Figure 5.44 Approximation of quadric surfaces by QCCM

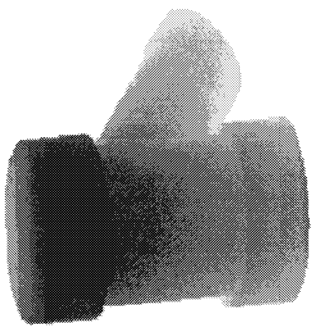
Every third pixel was chosen from the original image in both the horizontal and vertical directions to reduce the computational burden. This also makes the data sparse, illustrating the fact that the QCCM algorithm can work for sparse data. The number of feature points after sampling is about 3,000.

In the CCM algorithm, the GK algorithm was applied with fuzzifier m=1.5. The initial number of clusters c_{max} was 15. Figure 5.44 (b) displays the CCM planar approximation for this image. The final results of the QCCM algorithm consisting of the correctly identified surfaces is shown in Figure 5.44 (c). It is very difficult to see the surface of the cone in view (c) of Figure 5.44 correctly because all pixels on both sides of the surface belong to the same segment, so no matter what color is chosen for this segment, the visually apparent ellipse in view (a) which tells you the cone is not solid cannot be seen. But it's there.

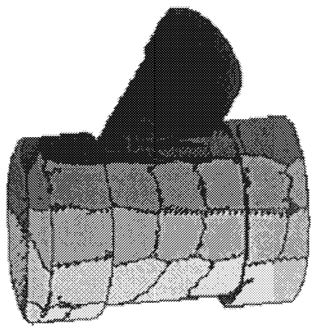
Frigui and Krishnapuram (1996a) propose an algorithm for quadric surface approximation based on the RCA algorithm (see Section 2.5). The algorithm starts by dividing the image into a large number of non-overlapping windows. Then RCA is applied on each window with c=1. This generates a large number (say c_{max}) of initial prototypes. The approximate distance in (2.55) is used in RCA. At this point RCA is applied on the whole data set with $c=c_{max}$. Due to the competitive and agglomerative nature of RCA, when the algorithm converges, only the clusters corresponding to legitimate surfaces survive. Since RCA is robust, it can also handle noisy range data. Unlike QCCM, this approach does not require an initial planar approximation.

Example 5.16 Figure 5.45(a) shows a 240×240 synthetically created range image that presumably mimics the range image that would be obtained by illuminating a real plastic pipe. Every third pixel in the horizontal and vertical directions was used to reduce computation. The image was divided into non-overlapping windows of size 30×30, and initial prototypes for each patch were generated using only the points in the window. Figure 5.45(b) shows the initial surface patches, where each point in the entire image is assigned to the nearest prototype.

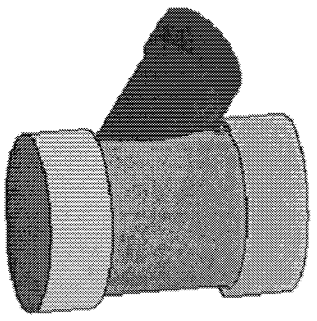
Figure 5.45(c) shows the final result, after RCA is applied with the initialization shown in Figure 5.45(b). Each surface is shown in a different gray value and the boundaries are black. The termination condition used for RCA was $\underbrace{\max}_{1 \le i \le c}\left\{\left\|\mathbf{p}_{i,t} - \mathbf{p}_{i,t-1}\right\|_1\right\} < 0.1$, where the subscript t represents the iteration number. If the distance criterion was not met, the algorithm was terminated at the maximum number of iterations, which was set at 50 in this example.



(a) "range image" of pipe junction



(b) initial approximation



(c) result of CA method

Figure 5.45 Boundary description by RCA

5.7 Representation of Image Objects as Fuzzy Regions

Geometric and non-geometric properties of objects from images play an important role in image understanding. Properties computed from regions are typically used for object description and shape analysis. By geometric properties, we mean those properties that deal with the shape of the silhouette of the object. Some of the commonly-encountered geometric properties of objects are area, perimeter, height, length, extrinsic diameter, intrinsic diameter, elongatedness, and roundness. By non-geometric properties, we mean properties that depend on the actual gray-level patterns within the object boundary. Examples of non-geometric properties are intensity, color and texture. Properties of objects are useful for object description, discrimination and shape analysis.

If objects in the image appear consistently brighter (or darker) than the background, a binary image of the objects can be obtained by thresholding the gray level image (see Section 5.5). However, object boundaries in gray-scale images are often blurred and distorted due to the imaging process. The thresholding operation does not preserve the uncertainty in the image, and could distort the shape and size of the object. Hence, the computed geometric properties may not be accurate. An alternative approach is to preserve the uncertainty inherent in the image as long as possible until actual decisions have to be made. In this approach, each object in the image is treated as a fuzzy region represented by a fuzzy set. A fuzzy approach would assign lower weights (memberships) to property values near the boundaries, thus leading to more accurate estimates. The methods discussed in Section 5.5 can be used to generate the membership function for the object regions in the image. In this section, we discuss various methods to compute properties of fuzzy regions.

A. Fuzzy Geometry and Properties of Fuzzy Regions

It is possible to define properties such as perimeter, height, etc. for multispectral images, but most of the work we are aware of is for the unispectral case. Consequently, in this section images are understood to be unispectral. Ideally, each region in a segmented image corresponds to an object or object class. In a fuzzy representation, fuzzy region F is represented by a membership function $m_F : IJ \to [0,1]$. When F is finite we use the variables (i,j) as arguments for m_F; otherwise, we use the variables (u, v), and remind you of our convention about the domain of support for integrals and derivatives of functions of (u, v). This notation is designed for images with two spatial dimensions, but many of the ideas in this section generalize to higher dimensions.

In gray-scale images it is convenient to represent the object or region in terms of crisp α-cuts of m_F, where $\alpha \in [0,1]$ and

$$F_\alpha = \{(i,j) \in IJ : m_F(i,j) \geq \alpha\} \quad . \qquad (5.44)$$

Since the gray values of images are quantized in practice, if the α-cut values are ordered as $1 = \alpha_1 > \alpha_2 > \cdots > \alpha_n > \alpha_{n+1} = 0$, the level sets from (5.44) are nested, i.e., $F_{\alpha_i} \subseteq F_{\alpha_j}$ for $\alpha_i > \alpha_j$.

Rosenfeld (1979, 1984, 1992) defined many terms in fuzzy geometry that can be used in the analysis of fuzzy regions of objects. Pal and Rosenfeld (1988), Pal and Ghosh (1990) have defined similar geometric attributes. Pal's (1992a) book chapter contains a review of this topic. Dubois and Jaulent (1987) generalized Rosenfeld's definitions using both fuzzy set and evidence theories. Here we briefly summarize the definitions of some geometric properties of fuzzy regions.

The *area of a fuzzy region* F is defined as the volume under m_F,

$$a(F) = \int\int m_F(u,v)dudv \quad . \tag{5.45}$$

In the discrete case, the area of fuzzy region F is

$$a(F) = \sum_{(i,j)\in IJ} \sum m_F(i,j) \quad . \tag{5.46}$$

Assuming that m_F is piece-wise constant, and that the finite number of constant values that m_F takes are given by $\alpha_1, \alpha_2, \cdots, \alpha_n$, the *perimeter* per(F) of F defined by (Rosenfeld and Haber 1985) is

$$\text{per}(F) = \sum_{i<j\in IJ} \sum \left(\sum_k |\alpha_i - \alpha_j| \ell_{ijk} \right) \quad , \tag{5.47}$$

where $\{\ell_{ijk}\}$ are the lengths of the arcs along which the discontinuity between regions F_{α_i} and F_{α_j} occurs. Assuming that m_F is smooth, Krishnapuram and Medasani (1995) define the *perimeter of a fuzzy region* with respect to the Euclidean norm as

$$\text{per}_{f2}(F) = \int\int \left[\left(\frac{\partial m_F(u,v)}{\partial u} \right)^2 + \left(\frac{\partial m_F(u,v)}{\partial v} \right)^2 \right]^{1/2} dudv. \tag{5.48}$$

The perimeter of a fuzzy region can be defined with respect to any norm on $\Re^2$, and Krishnapuram and Medasani also consider the 1-norm distance to measure the lengths of the arcs,

$$\text{per}_{f1}(F) = \int\int \left[\left| \frac{\partial m_F(u,v)}{\partial u} \right| + \left| \frac{\partial m_F(u,v)}{\partial v} \right| \right] dudv \quad . \tag{5.49}$$

The *height of a fuzzy region* F along the direction u may be defined as

$$h_u(F) = \int_v \max_u \{m_F(u,v)\} dv \quad , \tag{5.50}$$

where v denotes a direction orthogonal to u. In the discrete case, we have (Rosenfeld 1979)

$$h_i(F) = \sum_j \max_i \{m_F(i,j)\} \quad , \tag{5.51}$$

The *length* ℓ of a fuzzy region F along the direction u is defined as

$$\ell_u(F) = \max_u\{\int_v m_F(u, v)dv\} \quad , \qquad (5.52)$$

where the integration is performed over the region of the image P_{IJ} inside which $m_F(u, v) > 0$. In the discrete case, we have

$$\ell_i(F) = \max_i\{\sum_j m_F(i, j)\} \quad . \qquad (5.53)$$

In a digital picture where (u, v) takes only discrete values (i, j), since $m_F(i, j) = 0$ outside a bounded region, the max operation is performed over a finite set. By default, the *height* of an object or region F in a digital image is taken as the sum of the maximum memberships among the rows of positive memberships. This corresponds to using the horizontal direction for u in (5.50) and for i in (5.51).

The *width* w(F) of a fuzzy region F is the sum of the maximum membership values of the columns of positive memberships. This corresponds to using the vertical direction for u in (5.50) and for i in (5.51). Similarly, the *length* ℓ(F) and *breadth* b(F) of a fuzzy region F correspond to using only the horizontal (vertical) direction for u in (5.52) and for i (5.53). Thus, the length of a fuzzy region F in an image gives its largest expansion in the column direction, while the breadth gives its largest expansion in the row direction.

Example 5.17 The non-zero values of a membership function m_F of a region F are shown in Figure 5.46(a). The corresponding crisp membership function of the same region, obtained by thresholding m_F at 0.5, is shown in Figure 5.46 (b). Pixels that are not zeroed by hardening are shown with bold values and shaded cells in view (b).

Table 5.16 Fuzzy and crisp properties of the region F

(a) m_F for fuzzy region F

			0.1					
		0.1	0.2	0.1	0.1			
		0.1	**0.5**	**0.5**	0.2	0.1	0.1	
	0.1	0.2	**0.6**	**0.8**	**0.6**	**0.5**	0.2	0.1
0.1	0.2	**0.5**	**0.6**	**1.0**	**0.6**	0.2	0.1	
	0.1	0.1	**0.5**	0.4	**0.5**	0.1		
			0.1	0.2	0.2	0.1		
					0.1			

→

(b) m_F of crisp region

	1.0	**1.0**		
	1.0	**1.0**	**1.0**	**1.0**
1.0	**1.0**	**1.0**	**1.0**	
	1.0		**1.0**	

The crisp and fuzzy areas, heights, lengths, widths and breadths of F computed from the membership functions in Figure 5.46 are shown in Table 5.16.

Figure 5.46 Fuzzy and crisp geometry of image regions

property	crisp geometry	fuzzy geometry
area	12	10.9
height	1+1+1+1 = 4	0.1+0.2+0.5+0.8+1.0+ 0.5+0.2+0.1 = 3.4
length	max (1,4,3,3,1) = 4	max (0.1, 0. 4, 1.0, 2.6, 3.0, 2.3, 1.0, 0.4,0.1) = 3.0
width	1+1+1+1+1 = 5	0.1+0.2+0.5+0.6+1.0+0.6 +0.5+0.2+0.1 = 3.8
breadth	max (2,4,4,2) = 4	max (0.1, 0.5, 1.5, 3.1, 3.3, 1.7, 0.6, 0.1) = 3.3

Rosenfeld (1984) defined the *extrinsic diameter* of a fuzzy region F as

$$e(F) = \max_u\{h_u(F)\} \quad , \tag{5.54}$$

where h_u is defined in (5.50). The geometric property *elongatedness* is defined in terms of the ratio of the minor extrinsic diameter and the major extrinsic diameter, i.e.,

$$m_{EL}(F) = 1 - \frac{\min_u\{h_u(F)\}}{e(F)} \quad . \tag{5.55}$$

The *compactness* of a fuzzy region is defined by (Rosenfeld 1984) as

$$\text{comp}(F) = \frac{a(F)}{(p(F))^2} \quad . \tag{5.56}$$

The *index of area coverage* (IOAC) defined by Pal and Ghosh (1990) is

$$\text{IOAC}(F) = \frac{a(F)}{\ell(F) \cdot b(F)} \quad . \tag{5.57}$$

It is possible to give unified definitions for both geometric and non-geometric properties of fuzzy regions (Dubois and Jaulent, 1987, Krishnapuram et al. 1993a). In general, each non-geometric property has a feature associated with it. For example, associated with the greenness property is the green component of an RGB image. If we let $P(F_{\alpha_i})$ denote the value of a property in the crisp region F_{α_i}, then we may compute the expected value $\overline{P}(\mathbf{F})$ of the property for a fuzzy region with n α-cuts as

$$\overline{P}(\mathbf{F}) = \sum_{i=1}^{n} bp(F_{\alpha_i}) \cdot P(F_{\alpha_i}) \quad . \tag{5.58}$$

In (5.58), $bp(F_{\alpha_i})$ denotes a weighting function associated with F_{α_i}, which is sometimes called a *basic probability assignment* (bpa) (Shafer, 1976, Dubois and Jaulent, 1987), and it must satisfy the conditions

$$\sum_{i=1}^{n} bp(F_{\alpha_i}) = 1 \ ; \ bp(F_{\alpha_i}) \geq 0 \ \forall i \quad . \tag{5.59}$$

Dubois and Jaulent (1987) suggested the following definition of the bpa,

$$bp(F_{\alpha_i}) = \alpha_i - \alpha_{i+1} \quad , \tag{5.60}$$

where it is assumed that $1 = \alpha_1 > \alpha_2 > \cdots > \alpha_n > \alpha_{n+1} = 0$. Since F_{α_i} is a crisp set, traditional techniques can be used to compute $P(F_{\alpha_i})$. Using the definition of $bp(F_{\alpha_i})$ in (5.60), Dubois and Jaulent (1987) proved that: the expected area $\overline{a}(F)$ is equal to a(F); the expected height $\overline{h}(F)$ along the v-axis (i.e., in the default direction) of F is equal to the height h(F) of F along the v-axis; and the expected perimeter $\overline{p}(F)$ is equal to p(F). For the expected extrinsic diameter, the following inequality is true:

$$\overline{e}(F) \geq e(F) \quad . \tag{5.61}$$

Other definitions for the basic probability assignment (bpa) are possible. For example, we can define

$$bp(F_{\alpha_i}) = \frac{\left|F_{\alpha_i}\right|}{\sum_{i=1}^{n}\left|F_{\alpha_i}\right|} \quad . \tag{5.62}$$

This is the normalized α-cut cardinality. Using (5.62) for bp(F_{α_i}) we can compute the expected value of the property as

$$\bar{P}(F) = \frac{\sum_{i=1}^{n} |F_{\alpha_i}| P_i}{\sum_{i=1}^{n} |F_{\alpha_i}|} = \frac{\sum_{i=1}^{n} N_i P_i}{\sum_{i=1}^{n} N_i} , \qquad (5.63)$$

where P_i denotes the property value of pixels in F_{α_i} and N_i is the cardinality of F_{α_i}. In (5.63) the property values of each crisp region F_{α_i} are weighted by the cardinalities of the corresponding α-cut regions. Alternatively, we can use weighted α-cut cardinalities as the bpa, i.e.,

$$\mathrm{bp}(F_{\alpha_i}) = \frac{\alpha_i |F_{\alpha_i}|}{\sum_{i=1}^{n} \alpha_i |F_{\alpha_i}|} = \frac{\alpha_i N_i}{\sum_{i=1}^{n} \alpha_i N_i} . \qquad (5.64)$$

When (5.64) is used in (5.58), the properties of each of the α-cut regions are weighted by the cardinalities of the α-cut as well as the α-values.

B. Geometric properties of original and blurred objects

Krishnapuram and Medasani (1995) show that the definitions given in the previous section for properties of fuzzy regions can be used directly on a blurred binary image to obtain accurate estimates of the geometric properties of the original object. This is important because thresholding can change the size and shape of the object, and hence the property values computed from a thresholded image can be misleading. Here we only consider the properties area, perimeter, height, and length, and show the relations between the values of these properties computed from a blurred image and the values of the same properties computed from the original binary image. Similar relations can be derived for other commonly-used properties.

Objects may appear blurred in images either due to the imaging process or due to diffuse boundaries. In most instances, the blurring can be modeled by a convolution operation. In this section we will be integrating and differentiating several functions of two real variables, so we let x and y denote the horizontal and vertical directions in a unispectral image; let f(x, y) denote the original picture function of an object in a binary image; and let g(x, y) denote the blurred image of the object. We write

$$g(x,y) = \int\int f(u,v) \cdot b(x-u, y-v)dudv \quad , \tag{5.65}$$

where b(x, y) is the blurring function, and integration extends over the object region on which f(x, y) is greater than zero. In practice, f and b have finite supports, and if the image is sufficiently large, aliasing does not occur. If the membership function m_F of the object region F is obtained by applying a linear mapping to pixels in the image such as the one in (5.29), ignoring the scale factor, we can treat the picture function f(x, y) as the membership function of the original object and g(x, y) as the membership function of the blurred object.

Let a(f) denote the original area of an object in the image. From the definition in (5.45), the area of the blurred object represented by g(x, y) is given by

$$\begin{aligned} a(g) &= \iint g(x,y)dxdy \\ &= \iint \left\{ \iint f(u,v) \cdot b(x-u, y-v)dudv \right\} dxdy \\ &= \iint \left\{ \iint b(x-u, y-v)dxdy \right\} f(u,v)dudv \\ &= \iint a(b) f(u,v)dudv \end{aligned} \quad . \tag{5.66}$$

This shows that

$$a(g) = a(f) \cdot a(b) = a(f) \quad . \tag{5.67}$$

The last equality follows from the fact that we assume that the area of the blurring object b(x, y) is unity. In other words, the fuzzy area of the blurred function is equal to the original area. This result is quite general in that the original object need not be binary. If the area of the blurring function b(x, y) is not unity, we need to normalize a(g) by the area of the blurring function. This can be *roughly* achieved by calibrating the imaging system by imaging an object with a known area (while the actual object area may be known, we cannot know its area in the digital image, since it is blurred). In what follows, we assume that the area of the blurring function b(x, y) is unity, in which case, we have the general result, where "*" stands for the convolution operator.

$$\iint f(x,y)dxdy = \iint \{f(u,v) * b(u,v)\} dxdy \quad . \tag{5.68}$$

Let $\text{per}_{f2}(f)$ denote the fuzzy perimeter of the original binary object with respect to the 2-norm, and let $\text{per}_{f2}(g)$ denote the same fuzzy perimeter of the blurred object. From (5.48) we have

$$\text{per}_{f2}(g) = \iint \left[\left(\frac{\partial g(x,y)}{\partial x}\right)^2 + \left(\frac{\partial g(x,y)}{\partial y}\right)^2\right]^{1/2} dxdy$$
$$= \iint \left[\left(\frac{\partial (f*b)(x,y)}{\partial x}\right)^2 + \left(\frac{\partial (f*b)(x,y)}{\partial y}\right)^2\right]^{1/2} dxdy \cdot$$

Since

$$\frac{\partial (f*b)(x,y)}{\partial x} = \frac{\partial f(x,y)}{\partial x} * b(x,y) \quad \text{, and}$$
$$\frac{\partial (f*b)(x,y)}{\partial y} = \frac{\partial f(x,y)}{\partial y} * b(x,y)$$

we may write

$$\text{per}_{f2}(g) = \iint \left[\left(\frac{\partial f(x,y)}{\partial x} * b(x,y)\right)^2 + \left(\frac{\partial f(x,y)}{\partial y} * b(x,y)\right)^2\right]^{1/2} dxdy$$
$$\le \iint \left[\left(\left|\frac{\partial f(x,y)}{\partial x}\right| * b(x,y)\right)^2 + \left(\left|\frac{\partial f(x,y)}{\partial y}\right| * b(x,y)\right)^2\right]^{1/2} dxdy \cdot$$

It can be shown that

$$\left([a(x,y)*c(x,y)]^2 + [b(x,y)*c(x,y]^2\right)^{1/2} \le \left(a^2(x,y) + b^2(x,y)\right)^{1/2} * c(x,y)$$

when a(x, y), b(x, y) and c(x, y) are always positive. Therefore, we have

$$\text{per}_{f2}(g) \le \iint \left\{\left[\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2\right]^{1/2} * b(x,y)\right\} dxdy.$$

Using the general result in (5.68) we write the fuzzy perimeter as

$$\mathrm{per}_{f2}(f) = \iint \left\{ \left[\left(\frac{\partial f(x,y)}{\partial x} \right)^2 + \left(\frac{\partial f(x,y)}{\partial y} \right)^2 \right]^{1/2} * b(x,y) \right\} dxdy.$$

Hence

$$\mathrm{per}_{f2}(g) \le \mathrm{per}_{f2}(f) \qquad . \tag{5.69}$$

In practice, if the object does not have narrow intrusions or protrusions, and if the support of the blurring function is small compared to the size of the object, per(g) is a good approximation to per(f). A similar result can be shown for the case of the fuzzy perimeter in the city-block norm, i.e., $p_{f1}(g) \le p_{f1}(f)$.

In general, the original height and the fuzzy height of the blurred object are not equal. However, the fuzzy height of the blurred object is a good approximation of the original height, provided that the support of the blurring function is small compared to the support of the original function. To give an intuitive idea as to why this is so, we now derive a result assuming that the original object is rectangular.

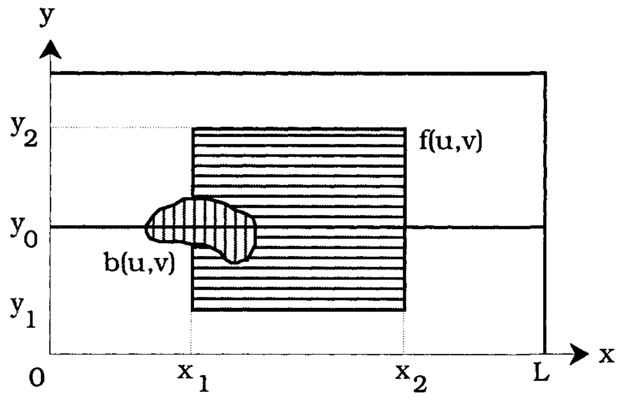


Figure 5.47 A rectangular object convolved with a blurring function

Let f(x, y) denote the image of the original (unblurred) object. It follows that f(x, y) can be treated as a binary function whose value is either 0 or 1 for all (x, y). Let g(x, y) denote the blurred image. Let h(f) denote the height of the original object in the direction of the x-axis, and let h(g) denote the height of the blurred object. If the object extends from x_1 to x_2 in the x-direction and y_1 to y_2 in the y direction, as shown in Figure 5.47, we can write

$$h(g) = \int_0^L \max_y g(x,y)dx = \int_0^L \max_y \left(\int_{u=x_1}^{x_2} \int_{v=y_1}^{y_2} f(u,v)b(x-u,y-v)dudv \right) dx.$$

If the support of the blurring function is smaller than the support of the rectangle, then there exists at least one $y = y_0$ such that all the maximum values of g(x, y) occur at $y = y_0$ (see Figure 5.47). Therefore,

$$h(g) = \int_0^L \left(\int_{u=x_1}^{x_2} \int_{v=y_1}^{y_2} f(u,v)b(x-u,y_0-v)dudv \right) dx$$

$$= \int_{u=x_1}^{x_2} \left(\int_{x=0}^{L} \int_{v=y_1}^{y_2} f(u,v)b(x-u,y_0-v)dxdv \right) du \ .$$

If we assume that the object is rectangular, then we can write f(x, y) = $f_1(x)f_2(y)$, where $f_1(x)$ and $f_2(y)$ are one-dimensional square waves. Since $f_2(v) = 1$ within the limits of integration,

$$h(g) = \int_{u=x_1}^{x_2} f_1(u) \left(\int_{x=0}^{L} \int_{v=y_1}^{y_2} f_2(v)b(x-u,y_0-v)dxdv \right) du$$

$$= \int_{u=x_1}^{x_2} f_1(u) \left(\int_{x=0}^{L} \int_{v=y_1}^{y_2} b(x-u,y_0-v)dxdv \right) du \quad .$$

The value of the double integral is always 1 for all values of u within the limits of integration because it is the area of a reflected version of the blurring function. Hence,

$$h(g) = \int_{u=x_1}^{x_2} f_1(u)\,du = x_2 - x_1, \text{ from which there follows,}$$

$$h(g) = h(f) \quad . \qquad (5.70)$$

A similar result holds for the length, i.e., assuming that f(x, y) is rectangular,

$$\ell(g) = \ell(f) \quad . \qquad (5.71)$$

We conjecture that the above result is true for non-rectangular objects as well as long as the support of the blurring function is small compared with the intrusions and protrusions in the contour of the object. Formalization and proof of this conjecture (or a counterexample that shows the conjecture is false) would be very interesting.

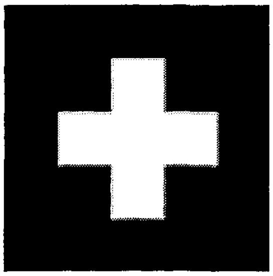
Let the extrinsic diameter of the blurred object be denoted by e(g), and let e(f) denote the extrinsic diameter of the original object. Since the extrinsic diameter is defined as the maximum value of the height measured in all possible directions (see (5.54)), if the height of the blurred object is the same as the height of the original object in all directions, then the extrinsic diameters of the original and blurred objects will also be the same, i.e.,

$$e(f) = \max_u \{h_u(f)\} \quad , \text{ and} \qquad (5.72a)$$

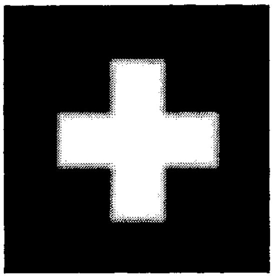
$$e(g) = \max_u \{h_u(g)\} = e(f) \quad , \qquad (5.72b)$$

where the max is performed over all possible directions u.

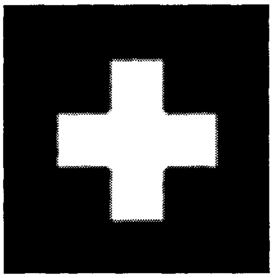
Example 5.18 Figure 5.48 shows a 200×200 synthetic image of the character (+). The image was blurred by convolving it with a truncated two-dimensional Gaussian, $b(x,y) = n(\mathbf{0}, \sigma I)$, notation as in (2.18). Different degrees of blurring were produced by varying the standard deviation of the Gaussian function. Here we show the results for σ = 2, 3, and 4 pixels. The size of the Gaussian mask was 33×33 pixels. The Otsu (1979) algorithm, which finds an optimal threshold by minimizing the intra-class variance, was used to threshold the image in all cases.



(a) original (+) image

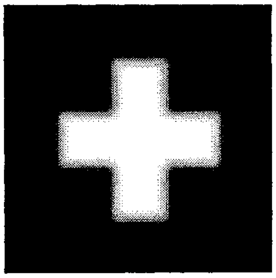


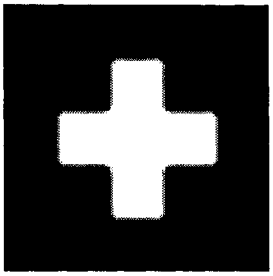
(b) (+) blurred with $b(x,y) = n(0, 2I)$

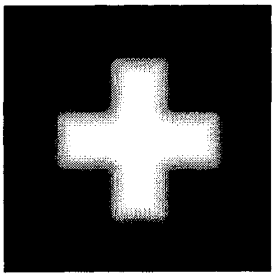


(c) image (b) thresholded

Figure 5.48 Images for fuzzy geometric properties


(d) (+) blurred with b(x, y) = $n(0, 3I)$


(e) image (d) thresholded


(f) (+) blurred with b(x, y) = $n(0, 4I)$

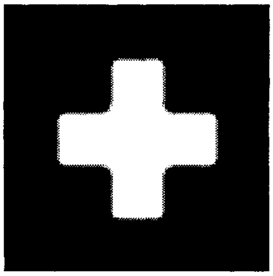

(g) image (f) thresholded

Figure 5.48 (con't.) Images for fuzzy geometric properties

Table 5.17 summarizes the values of the geometric properties: area, perimeter (using the city-block or 1-norm distance), height and length for the synthetic binary image (+). The table shows that the fuzzy area is always equal to the area of the original binary image. The value of the area computed by thresholding the image usually becomes progressively worse as the variance of the blurring function increases, but this is not the case with the fuzzy area. In accordance with (5.69), the fuzzy perimeter is always less than or equal to the perimeter of the original binary image. The fuzzy method gives a better estimate than the thresholding method in each case. In the case of height and length, both methods give similar results, although the fuzzy method is slightly better.

Table 5.17 Crisp and fuzzy geometric properties

		$\sigma=2$		$\sigma=3$		$\sigma=4$	
	Orig.	Fuzzy	Thresh.	Fuzzy	Thresh.	Fuzzy	Thresh.
Area	8241	8241	8012	8241	7988	8241	7992
Perim.	484	484	480	484	480	484	480
Height	121	121	120	121	120	121	120
Length	121	121	120	121	120	121	120

Next we give an example that illustrates the measurement of a non-geometric property - average gray level - of regions within an image. An alternative formulation of the PCM algorithm (Krishnapuram and Keller 1996, Davé and Krishnapuram 1997) is used in this application. In this formulation the objective function is

$$\underbrace{\min}_{(U,\,\mathbf{B})}\left\{J_m(U,\mathbf{B};\mathbf{w})=\sum_{i=1}^{c}\sum_{k=1}^{n}u_{ik}^m D_{ik}(\mathbf{x}_k,\boldsymbol{\beta}_i)+\sum_{i=1}^{c}w_i\sum_{k=1}^{n}\left(u_{ik}\log u_{ik}-u_{ik}\right)\right\}, \tag{5.73}$$

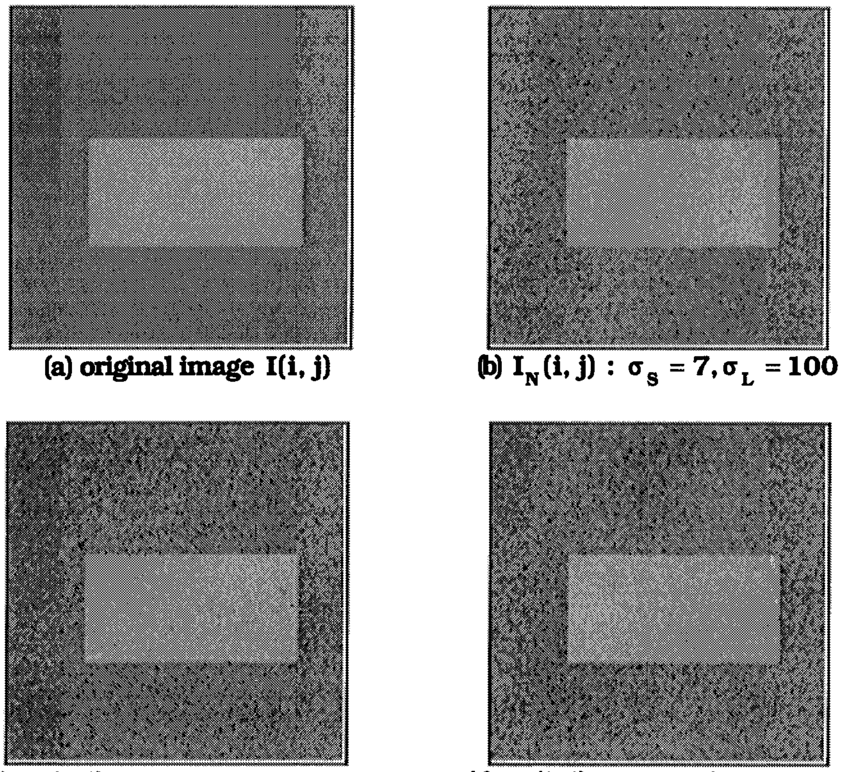
and the membership update equation is

$$u_{ik}=\exp\left\{-\frac{D_{ik}(\mathbf{x}_k,\boldsymbol{\beta}_i)}{w_i}\right\} \quad . \tag{5.74}$$

The center $\mathbf{v}_i$ is updated as usual. Since the exponential membership function in (5.74) decays faster than the one in (2.8a), this formulation exhibits better characteristics when the clusters are expected to be close to one another. In Example 5.19 we use the Euclidean norm $D_{ik}(\mathbf{x}_k,\boldsymbol{\beta}_i)=\|\mathbf{x}_k-\mathbf{v}_i\|^2$, and $w_i=2\sigma_i^2$, where σ_i^2 is the variance of the cluster estimated after initially running the FCM algorithm until termination. After the PCM algorithm terminates with this initial estimate of w_i, the variance of each cluster is re-estimated by considering only the most typical points i.e., those that have a membership value greater than 0.5. With these updated w_i, the PCM algorithm is run once again on each cluster separately in order to fine-tune the centers and memberships.

Example 5.19 We compare estimates of the property P(F) = *average gray level of a region* measured by crisp, fuzzy and possibilistic c-means. The gray level feature values of pixels are clustered to generate membership functions for regions. These membership functions are then used to compute P(F) of the regions using two different methods. The first method uses the Dubois-Jaulent bpa in (5.60), and the second method employs the normalized α-cut cardinalities in (5.62). In both cases the membership values are first quantized to 10 levels. The property $P(F_{\alpha_i})$ is generated for each of the ten α-cuts. $\bar{P}(F)$ is then computed by equation (5.58), which combines the crisp estimates with the bpa.

There are two regions in the original 256×256 image shown in Figure 5.49(a) with intensities I(i,j) having one of two uniform gray levels. The original image was corrupted by adding two kinds of Gaussian noise, one with a small standard deviation σ_S (to simulate non-impulse noise) and the other with a large standard deviation σ_L (to simulate impulse noise).



(a) original image $I(i, j)$ **(b) $I_N(i, j)$: $\sigma_S = 7, \sigma_L = 100$**

(c) $I_N(i, j)$: $\sigma_S = 10, \sigma_L = 100$ **(d) $I_N(i, j)$: $\sigma_S = 15, \sigma_L = 100$**

Figure 5.49 Computing non-geometric properties of fuzzy regions

Gray levels $I_N(i, j)$ of pixels in the noisy image can be represented as $I_N(i, j) = I(i,j) + 0.95n(0, \sigma_S) + 0.05n(0, \sigma_L)$. Here we show the results for $\sigma_S = 7, 10, \text{and} 15$ with $\sigma_L = 100$. The original image and the images resulting after adding the three degrees of noise are shown in Figure 5.49.

Table 5.18 summarizes the values of the average gray level of the two regions: object = O and background = B. Membership functions were generated by running the HCM, FCM and PCM clustering algorithms on the noisy images with c = 2 using the unlabeled sets of corrupted intensities, $X_N = \{I_N(i, j)\}$, as inputs. From Table 5.18 the average intensity value for the object measured using the membership function generated by the PCM algorithm is slightly more accurate than the other two methods. This might be because in the PCM algorithm the membership values represent typicalities.

Table 5.18 Crisp and soft computation of average region intensity

Image in Figure (5,49)	Class	Orig.	HCM	FCM with (5.60)	FCM with (5.62)	PCM with (5.60)	PCM with (5.62)
5.49 (b)	O	180	176	173	175	180	180
σ_s=7	B	120	120	120	121	120	120
5.49 (c)	O	180	176	170	173	179	179
σ_s=10	B	120	120	121	121	120	120
5.49 (d)	O	180	172	164	168	178	178
σ_s=15	B	120	121	122	121	120	120

The measured value of P(F) for the object region deviates more from the actual value as the variance σ_s is increased. As expected, the same trend applies to the value of the average intensity property of the background region. The two methods for computing bpa, i.e., Dubois-Jaulent bpa and normalized α-cut cardinalities, gave similar results; Table 5.18 is based on the Dubois-Jaulent bpa.

In Examples 5.18 and 5.19 membership functions for the object regions were generated using a feature (gray level) that is related to the property we are trying to measure. Our recommendation is to adhere to this as a general rule - generate membership functions with features that are related to the property you want to measure. This seems obvious, but it is worth saying - if you don't follow this rule of thumb, the results can be very disappointing (Medasani et al., 1999).

5.8 Spatial Relations

As explained in Section 5.1, properties of objects and spatial relations between objects play an important role in rule-based approaches for high-level vision. The partial presence or absence of such properties and relationships can supply both positive and negative evidence for region labeling hypotheses. In this section, we briefly review some fuzzy methods to represent spatial relations between image regions.

In some situations, spatial relations between objects are quite crisp. For example, in Figure 5.50 (a), "A is ABOVE B", and in panel 5.50(b), "A is to the LEFT OF B". Humans are able to describe spatial relationships between objects even when the relations are not so crisp. For example, in Figure 5.50(c) "A is *somewhat above* B" and in Figure 5.50(d) "A is *somewhat left of* B". However, this task has turned out to be a rather elusive for automation. When the objects in a scene are represented by fuzzy sets, rather than crisp sets, the problem of generating such relational descriptions becomes even more complex.

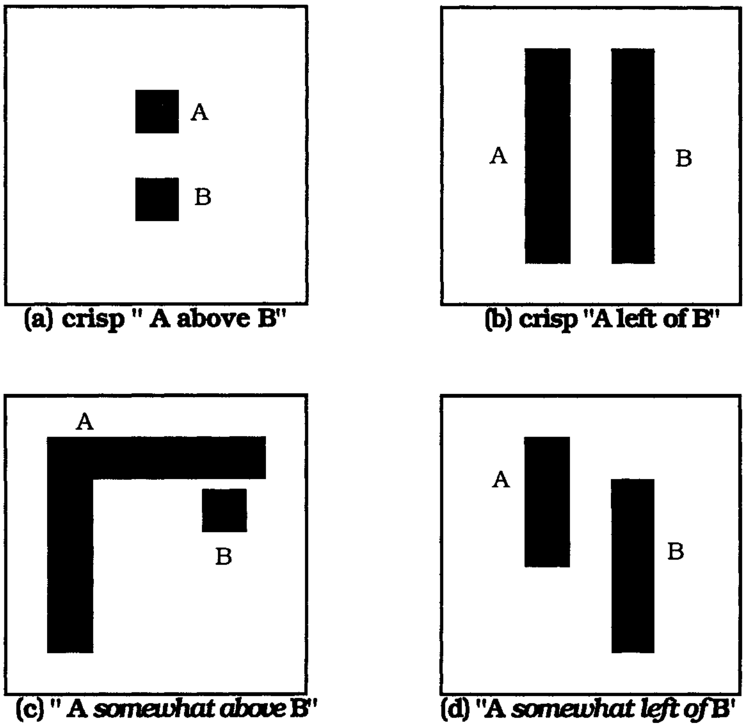


Figure 5.50 Spatial relations between objects

Approximate spatial relation analysis has also attracted the attention of many researchers in the past several years. In many situations, precise description of relations among objects may be too complex and computationally too expensive. Approximate spatial relation analysis provides a natural way to solve real world problems with a reasonable cost.

Freeman (1975) was among the first to recognize that the nature of spatial relations among objects requires that they be described in an approximate (fuzzy) framework. Rosenfeld and Klette (1985) defined the relations "adjacency" and "surroundedness" between image regions. Retz (1988) examined the intrinsic, deictic, and extrinsic use of spatial prepositions and designed a system called CITYTOUR that answers natural language questions about spatial relations between objects in a scene and about the movement of objects. Dutta (1991) applied fuzzy inference and used a generalization of Warshall's algorithm to reason about object spatial positions and motion. However, modeling spatial relations among image objects is not addressed in the last two papers.

Cultural aside "Deictic" is a term from logic that means "proving by direct argument" as opposed to elenctic, which means "refuting an argument by proving the falsehood of its conclusion". In grammar, "deictic" represents a word whose reference is determined by the context of its utterance. Examples are "here" and "I". Basically, deictic refers to the "relative nature" of the term. Spatial relations are relative as well.

Keller and Sztandera (1990) considered the problem of defining spatial relationships between fuzzy subsets of an image using dominance relations of projections of the regions onto coordinate axes. Krishnapuram *et al.* (1993a) presented three methods that can be used to characterize both properties and spatial relationships of object regions in a digital image. These methods are referred to as (1) the centroid method, (2) the aggregation method, and (3) the average-angle method. Other older versions of the centroid method can be found in Winston (1975) and Rosenfeld and Kak (1982). Miyajima and Ralescu (1993, 1994) proposed a method to evaluate spatial relations, which we will refer to as the compatibility method. Keller and Wang (1995) present a comparison of these methods.

Kundu (1995) defined the fuzzy spatial relation LEFT(A, B) between two objects A and B based on a set of seemingly desirable postulates such as object independence, translation invariance, etc. Bloch (1996a, 1996b) and Gader (1997) both present definitions based on mathematical morphology which are computationally efficient and give reasonable results.

Ideally, the automated system should yield spatial relation measures that are consistent with human intuition. However, most researchers have not paid attention to this issue. Marks and Egenhofer (1994) discuss how people think about spatial relations in an attempt to find a realistic basis for defining spatial relations. More recently, Keller and Wang (1996) have proposed a method based on multilayer perceptrons to mimic spatial relation values indicated by human subjects.

The primitive spatial relations between two objects are (Freeman 1975): (1) LEFT OF, (2) RIGHT OF, (3) ABOVE, (4) BELOW, (5) BEHIND, (6) IN FRONT OF, (7) NEAR, (8) FAR, (9) INSIDE, (10) OUTSIDE, and (11) SURROUND. Both Winston (1975) and Rosenfeld and Kak (1982) have discussed the difficulties in defining such relations for crisp subsets of the plane.

The first aggregation method we discuss is actually defined for fuzzy regions of the plane (Krishnapuram et al. 1993b). We first consider the case of a crisp region, which is equivalent to looking at a specific level set of a fuzzy region. This method is based on the observation that human perception of spatial positions between objects is closely related to angular information. Consider two points **a** and **b**.

in $\Re^2$. Let $\overline{\mathbf{ab}}$ denote the line connecting **a** and **b**, and let θ_{ab} be the *counterclockwise* angle between $\overline{\mathbf{ab}}$ and the horizontal axis as shown in Figure 5.51.

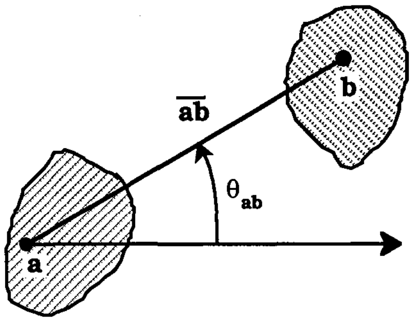


Figure 5.51 Using θ to define the spatial relation RIGHT OF

The membership function for RIGHT OF is defined as

$$m_{RIGHT}(\theta_{ab}) = \begin{cases} 1 & ; \quad |\theta_{ab}| < \frac{k\pi}{2} \\ \frac{(\pi/2) - |\theta_{ab}|}{(\pi/2)(1-k)} & ; \quad \frac{k\pi}{2} \le |\theta_{ab}| < \frac{\pi}{2} \\ 0 & ; \quad |\theta_{ab}| > \frac{k\pi}{2} \end{cases} \quad . \tag{5.75}$$

The value k > 0 in (5.75) is a constant such that $k\pi/2$ is the half width of the top (where it takes the value 1) of the trapezoidal membership function defined by (5.75). Figure 5.52 shows similar membership functions for LEFT OF, ABOVE, and BELOW.

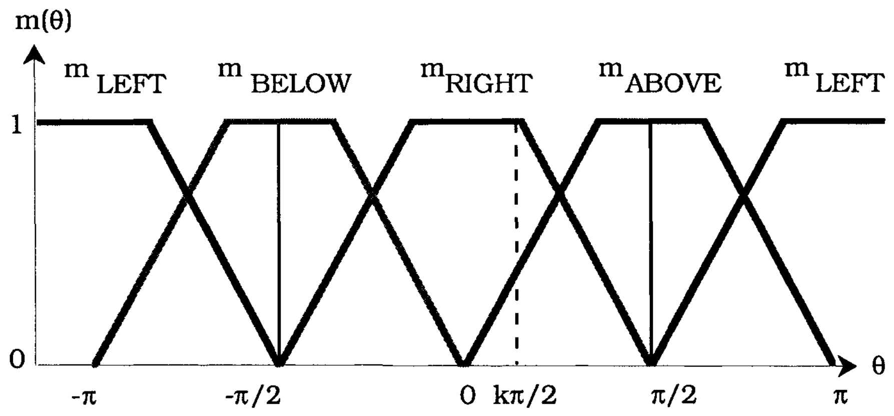


Figure 5.52 Membership functions for five spatial relations

If A and B are two image regions, to compute the membership for "A RELATION B" (e.g. "A is LEFT OF B"), Krishnapuram et al. compute the angle $\theta_{\mathbf{a}_i\mathbf{b}_j}$ for each pair of elements $\mathbf{a}_i \in$ A and $\mathbf{b}_j \in$ B, and evaluate the memberships $\{m_{\text{RIGHT}}(\theta_{\mathbf{a}_i\mathbf{b}_j})\}$. Finally, the membership $m_{\text{A\_REL\_B}}$ for "A RELATION B" is computed by aggregating the memberships $\{m_{ij}=m_{\text{RIGHT}}(\theta_{\mathbf{a}_i\mathbf{b}_j})\}$. Several fuzzy aggregation operators may be used for aggregation (Krishnapuram and Lee 1992a, 1992b). One choice is the *weighted mean of order q*,

$$M_q(\mathbf{m},\mathbf{w}) = \left(\sum_{j=1}^{n} \sum_{i=1}^{m} w_{ij} m_{ij}^{q} \right)^{1/q} \quad \text{, where} \tag{5.76a}$$

$$\mathbf{m} = (m_{11},\ldots,m_{nm})^T ; \mathbf{w} = (w_{11},\ldots,w_{nm})^T ; \text{and} \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} = 1. \tag{5.76b}$$

The weight w_{ij} denotes the relative importance of the corresponding membership value m_{ij}. The weights $\{w_{ij}\}$ may be (usually *are*) chosen to be equal to $(1/(n \cdot m))$, and then $M_q(\mathbf{m},[\mathbf{1}/(n \cdot m)]) \propto \|\mathbf{m}\|_q$. The parameter q is chosen to suit the required (or desired) degree of optimism or pessimism.

Krishnapuram et al. (1993a) also define membership functions for relations NEAR, FAR, INSIDE, OUTSIDE, and SURROUND. Figure 5.53 illustrates how the relation "A SURROUNDS B" can be handled.

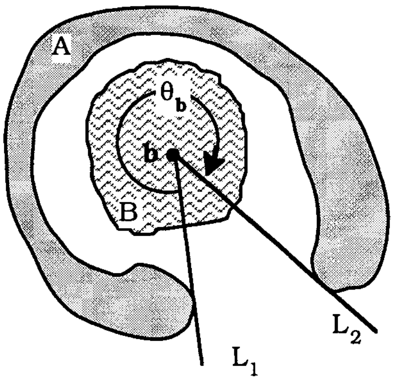


Figure 5.53 The angle that defines the spatial relation SURROUND

For each point $\mathbf{b} \in$ B two lines L_1 and L_2 that touch A are found. Then the *clockwise* angle θ_b between the lines is measured, as shown in Figure 5.53 (if no tangents exist, then $\theta_b = 2\pi$). Rosenfeld and Klette

(1985) discuss one membership function for "A SURROUNDS **b**", and Krishnapuram et al. (1993a) suggested a similar one, namely

$$m_{A\_SURROUND\_\mathbf{b}}(\theta_{\mathbf{b}}) = \begin{Bmatrix} 1 & ; & \theta_{\mathbf{b}} > (2-k)\pi \\ \dfrac{\theta_{\mathbf{b}} - \pi}{\pi(1-k)} & ; & \pi \le \theta_{\mathbf{b}} \le (2-k)\pi \\ 0 & ; & \theta_{\mathbf{b}} < \pi \end{Bmatrix}, \tag{5.77}$$

where the constant k in (5.77) has the same meaning as in (5.75). To find $m_{A\_SURROUND\_B}$, we find memberships $m_{A\_SURROUND\_\mathbf{b}}(\theta(\mathbf{b}))$ for all $\mathbf{b} \in B$ and aggregate them with, for example, equation (5.76).

Now we consider the case when A and B are fuzzy regions. Let the α-cuts of the two regions be denoted by $A_{\alpha_1}, A_{\alpha_2}, \cdots, A_{\alpha_n}$ and $B_{\alpha_1}, B_{\alpha_2}, \cdots, B_{\alpha_n}$. The membership value for A_{α_i} RELATION B_{α_i}, denoted by $m_{A\_REL\_B}(\alpha_i)$, is first computed for each corresponding pair of α-cuts A_{α_i} and B_{α_i}. The overall membership value for "A RELATION B" is then computed using an equation similar to (5.58),

$$m_{A\_REL\_B} = \sum_{i=1}^{n} bp(\alpha_i) m_{A\_REL\_B}(\alpha_i) \quad , \tag{5.78}$$

where $bp(\alpha_i)$ is the probability mass associated with level α_i.

In the centroid method used by Krishnapuram et al. (1993a) for fuzzy regions, the centroids of the two crisp regions A_{α_i} and B_{α_i} are first computed. Then the membership of the spatial relations in terms of the angular measurement of the centroids are generated from the same functions as in the aggregation method. Equation (5.78) is used to obtain the final memberships. The centroid method is computationally simple, but the results are not generally satisfactory.

Miyajima and Ralescu (1993, 1994) also present a method for computing spatial relations between pairs of points **a** and **b** based on the angle $\theta_{\mathbf{ab}}$ as shown in Figure 5.51. In their definition, however, positive angles are measured *clockwise*. The membership function they use for "**a** is RIGHT OF **b**" is given by

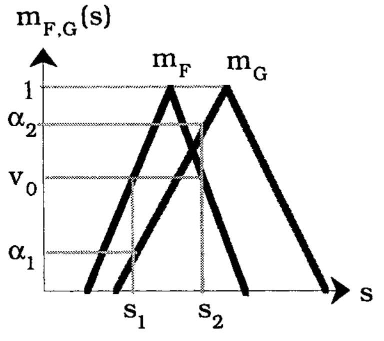
$$m_{RIGHT}(\theta_{\mathbf{ab}}) = \begin{Bmatrix} \cos^2\theta_{\mathbf{ab}} & ; -\frac{\pi}{2} \le \theta_{\mathbf{ab}} \le \frac{\pi}{2} \\ 0 & ; \text{otherwise} \end{Bmatrix} \quad . \tag{5.79}$$

The membership functions for the remaining relationships are defined similarly. As with the aggregation method, the case when A and B are both crisp regions which are sets of points, i. e., A = {$\mathbf{a}_1,\ldots,\mathbf{a}_n$} and B = {$\mathbf{b}_1,\ldots,\mathbf{b}_m$}, is reduced to considering nm pairs of points ($\mathbf{a}_i, \mathbf{b}_j$), i = 1,...,n; j = 1,...,m.

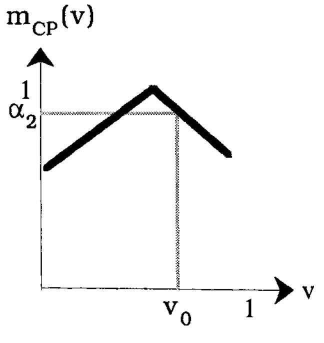
Let Θ denote the collection of angles $\{\theta_{\mathbf{a}_i\mathbf{b}_j}\}$ where $\mathbf{a}_i \in$ A and $\mathbf{b}_j \in$ B. Since different pairs of points may result in the same angle, Θ is a multiset. For each $\theta_{\mathbf{a}_i\mathbf{b}_j} \in \Theta$, let $n_\theta = \text{card}\left\{(\mathbf{a}_i, \mathbf{b}_j): \theta_{\mathbf{a}_i\mathbf{b}_j} = \theta\right\}$. Let $H_\Theta(A,B) = \left\{(\theta, n_\theta)\right\}$ denote the histogram associated with Θ. The frequency n_θ is divided by the largest frequency to get the normalized frequency. The histogram H_Θ is treated as an unlabeled fuzzy set that captures the spatial relations between A and B. Given the membership functions for the fuzzy sets RIGHT OF, LEFT OF, ABOVE, and BELOW, the degree to which H_Θ matches these spatial relations is obtained by measuring the compatibility between the H_Θ distribution and the fuzzy set that represents the relation. The compatibility of a distribution F to a fuzzy set G is a fuzzy set CP(F;G) with membership function

$$m_{CP(F;G)}(v) = \begin{cases} \sup\limits_{s:\, v=m_F(s)} \{m_G(s)\} & ; m_F^{-1}(v) \neq \varnothing \\ 0 & ; m_F^{-1}(v) = \varnothing \end{cases} \quad . \tag{5.80}$$

In our context F is the histogram H_Θ and G is a fuzzy set for the relation, such as the one in (5.79). After the compatibility is computed, the final degree to which a spatial relation holds is obtained as the center of gravity of the compatibility fuzzy set.



(a) MFs of fuzzy sets G and F

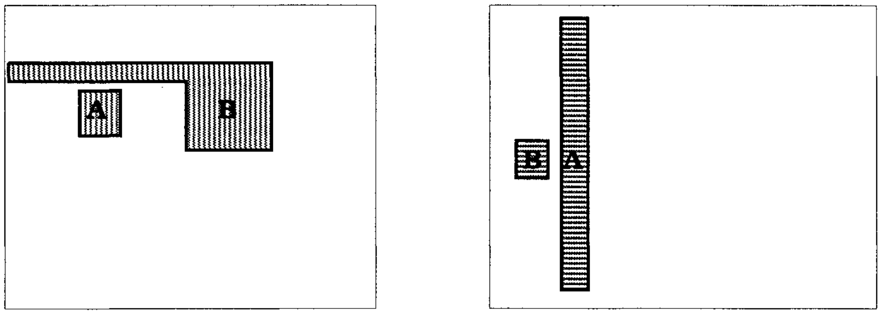


(b) compatibility set CP (F, G)

Figure 5.54 Compatibility between fuzzy sets

For the fuzzy sets m_F and m_G shown in Figure 5.54(a), given v_0, we have $v_0 = m_F(s_1) = m_F(s_2)$ for two points s_1 and s_2. We find $m_G(s_1) = \alpha_1$ and $m_G(s_2) = \alpha_2$. The compatibility value at v_0 is then $m_{CP}(v_0) = \max\{\alpha_1, \alpha_2\}$. By considering all values of v, we generate the compatibility fuzzy set shown in Figure 5.54(b). The center of gravity of m_{CP} is the degree to which a spatial relation holds.

Example 5.20 Figure 5.55 shows two images containing crisp regions named A and B. These images will be used to discuss "ABOVE", "BELOW", "RIGHT" and "LEFT".



(a) Test image (a) **(b) Test image (b)**

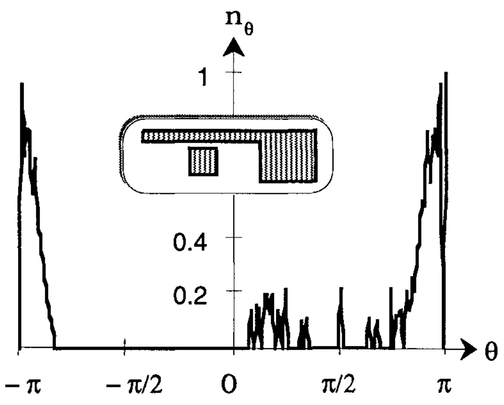
Figure 5.55 Images of crisp regions used in Example 5.20

Table 5.19 summarizes the results for the spatial relations "A RELATION B" using the aggregation, centroid, and compatibility methods. From Table 5.19, we see that for Figure 5.55(a), the centroid method tends to give results which are very close to crisp values. We think the results are slightly better for the aggregation method than for the compatibility method.

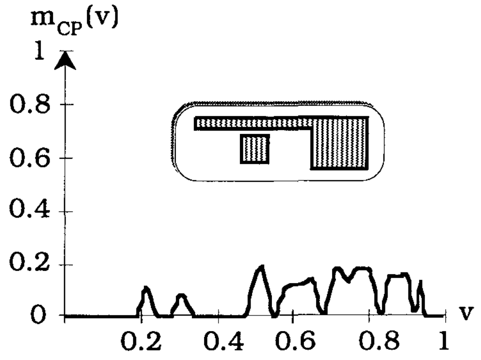
Table 5.19 Spatial relation results for images in Figure 5.55

	Figure 5.55(a)				Figure 5.55(b)			
Method	Left	Above	Right	Below	Left	Above	Right	Below
aggreg.	0.77	0.08	0.24	0.30	0.00	0.46	0.46	0.46
centroid	0.94	0.00	0.00	0.07	0.00	0.00	1.00	0.00
compat.	0.81	0.07	0.68	0.22	0.00	0.66	0.36	0.66

Compatibility gives an unreasonably high membership value (0.68) for "A is RIGHT of B" in Figure 5.55(a) because of the shape of the histogram of angles shown in Figure 5.56(a). Continuing with the processing associated with Figure 5.55(a), the compatibility fuzzy set of the membership function for "RIGHT", as well as the histogram computed in the region $[-\pi/2, \pi/2]$ are shown in Figure 5.56.



(a) histogram of angles between objects for Figure 5.55(a)

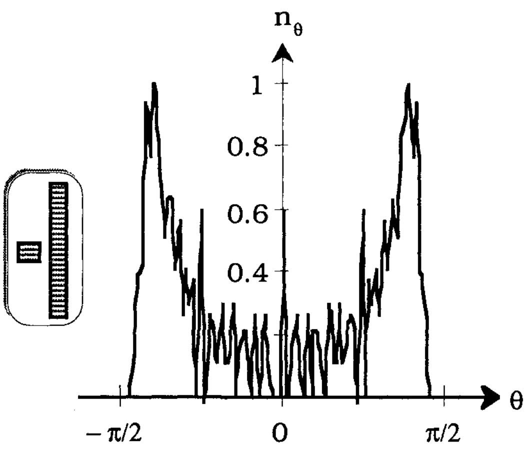


(b) compatibility fuzzy set for "A is RIGHT OF B" for Figure 5.55(a)

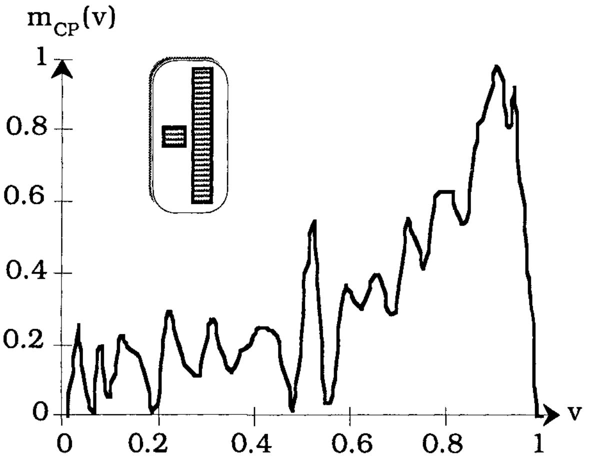
Figure 5.56 Histogram and compatibility fuzzy set for Figure 5.55(a)

When the compatibility fuzzy set has low membership for most values, the centroid is usually fairly large. For the image in Figure 5.55(b), the histogram of angles is shown in Figure 5.57(a), and its graph is much more regular with respect to θ than the graph of histogram angles in Figure 5.56(a). This agrees with a visual assessment of the relationship between A and B in right hand image in Figure 5.55(b).

The compatibility fuzzy sets for "A is ABOVE B", "A is RIGHT of B", and "A is BELOW B" are shown in Figure 5.57(b)-(d). After computing the centers of gravity of these sets, we can see that the degrees for "ABOVE" and "BELOW" are much higher than those for "RIGHT". The aggregation method gives almost the same value for "ABOVE", "BELOW", and "RIGHT". In the centroid method, the value for "RIGHT" is dominant (Table 5.19).



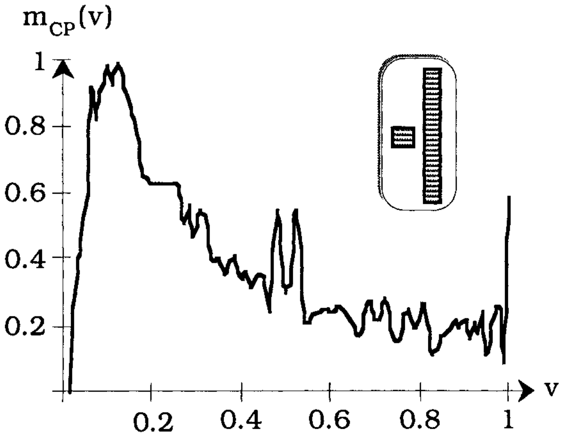
(a) histogram of angles between objects for Figure 5.55(b)



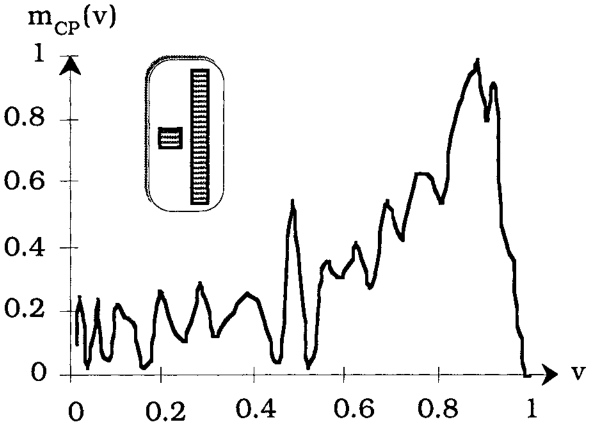
(b) compatibility fuzzy set for "A is ABOVE B" for Figure 5.55(b)

Figure 5.57 Histogram and compatibility fuzzy sets for Figure 5.55(b)

Humans probably would not assign the same degree to the three relations that was found by the aggregation method. However, it also is not reasonable to have "ABOVE" and "BELOW" dominate "RIGHT" as in the compatibility approach. In this case, the centroid method appears to be more consistent with intuitive perception.



(c) compatibility fuzzy set for "A is RIGHT OF B" for Figure 5.55(b)



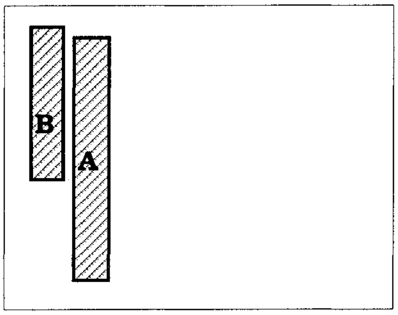
(d) compatibility fuzzy set for "A is BELOW B" for Figure 5.55(b)

Figure 5.57 (con't.) Histogram and compatibility fuzzy sets for Figure 5.55(b)

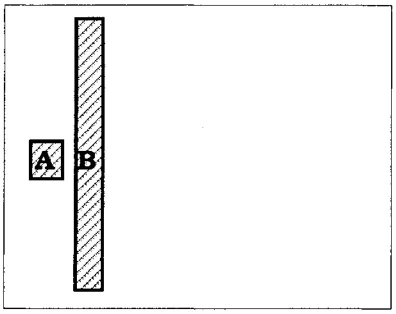
Keller and Wang (1995, 1996) proposed a method to learn spatial relations by training one or more multilayer perceptrons (Section 4.7). Target output values were assigned by human perception of images obtained from 62 participants with different genders, nationalities, ages, working fields, and educational levels.

The inputs to (multiple) neural networks were: (i) 181 feature values representing the angle histogram $H_{\Theta}(A,B) = \{(\theta, n_{\theta})\}$ (one value for every 2 degrees in the range of $[-\pi, \pi]$), (ii) the projected extents of the two objects on the x-axis, (iii) the projected extents of the two objects on the y-axis, (iv) the square roots of the areas of the two objects, (v) the distance between the centers of gravity of the two objects, (vi) the ratios of the overlap projected extent on the x-axis to each of the projected extents of the two objects on the x-axis, and (vii) the ratios of the overlap projected extent on the y-axis to each of the projected extents of the two objects on the y-axis. This gave 192 input features for training. Various neural networks were tested on a large variety of images. The results were found to be better than those obtained by the aggregation, centroid, and compatibility methods, perhaps because the networks were interpolating relationships similar to ones given by human subjects. Keller and Wang (1996) also suggest another approach in which the outputs of multiple neural networks are combined using the Choquet integral (see Section 4.5). Example 5.21 is adapted from Keller and Wang (1996).

Example 5.21 Figure 5.58 shows two typical test images used by Keller and Wang (1996). Note that the image in Figure 5.58(b) is similar to the one in Figure 5.55(b).



(a) Test image (a)



(b) Test image (b)

Figure 5.58 Images for testing spatial relationship definitions

Table 5.20 Spatial relations for images in Figure 5.58

	Figure 5.58(a)				Figure 5.58(b)			
Method	Left	Above	Right	Below	Left	Above	Right	Below
compatibility	0.00	0.63	0.28	0.74	0.00	0.66	0.36	0.66
aggregation	0.00	0.26	0.44	0.63	0.00	0.46	0.46	0.46
MLP	0.00	0.04	0.87	0.25	0.00	0.23	0.94	0.32
human value	0.00	0.03	0.85	0.31	0.00	0.24	0.86	0.25

Table 5.20 gives the results of the neural network method, as well as the values provided by humans and the values generated by the compatibility and aggregation methods. As can be seen from the

table, the memberships produced by the neural network method are closer to the human ones. This might be expected because of the training method used. MLP in Table 5.20 stands for multilayered perceptron as we have defined it in Section 4.7.

With all these definitions, how do we decide the "correct" method to calculate spatial relations? Intuition is useful (everyone's method has some sort of intuitive appeal), but intuition is biased by personal judgments. The real question is (as usual) - which spatial relationship provides features that are useful in a particular application?. Wang et al. (1997) used memberships from three spatial relation definitions together with a few other features in a digit recognition application. They report that memberships do provide powerful discriminating capability. More interestingly, the three definitions used for the spatial relations all provide about the same results. This offers some evidence that the most important point about spatial relationships may not be *which one* you use, but that you *use one* at all, based on some reasonable set of definitions for spatial relationships. Computational complexity is also an important consideration, and may be a way to choose among equally useful alternate definitions.

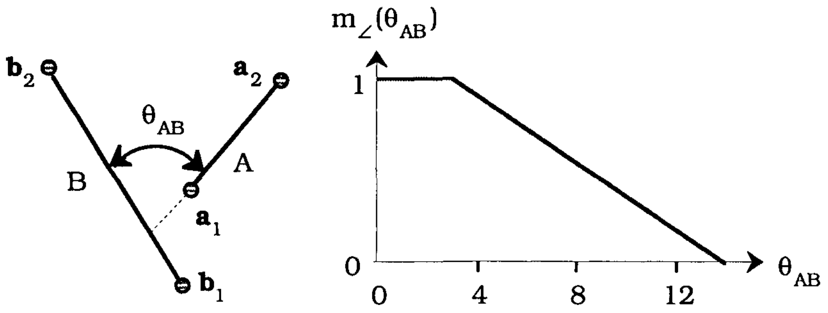
Work in this area is important because scene interpretation often improves if relationships between objects can be inferred computationally. A related area in computer vision is to group together similar structures (e.g. all the guitar players, all the pickup trucks, all the redfish, etc.) at higher levels. This is the subject of Section 5.9. Later in this chapter we give an example where spatial relationships are used to describe a scene.

5.9 Perceptual Grouping

Perceptual grouping involves joining higher level structures (or "tokens") from a lower level of representation to build more complex entities. Perceptual grouping provides a natural interface between top-down and bottom-up reasoning in computer vision systems. The lower-level grouping is typically data oriented (bottom-up), whereas the higher-level grouping is typically model-driven (top-down). For example, lower level grouping might involve merging short line segments (edge fragments) based on constraints such as collinearity. In contrast, higher-level grouping might involve searching for line segments that form the projection of a building modeled as a parallelepiped. In either case, the grouping is usually based on geometric relations and constraints between tokens to be grouped. Since geometric relationships between objects in images are typically ill-defined, fuzzy methods are well suited for determining to what degree the tokens satisfy geometric constraints.

Kang and Walker (1994) discuss several aspects of perceptual grouping. At the lower level, they discuss fuzzy strategies for grouping based on collinearity, parallelism, symmetry, and junction. At the higher level, they consider strategies for recognition tasks such as extraction of curves, natural branching structures, and polyhedral objects. Ralescu and Shanahan (1995) also discuss fuzzy methods for perceptual organization of lines and junctions. To illustrate how fuzzy methods can be used for perceptual grouping, we discuss the Kang-Walker model for line segment grouping based on collinearity in more detail.

To group line segments into higher-level tokens (longer line segments) based on collinearity, Kang and Walker (1994) use three constraints related to proximity and similarity. These are: (i) an angle constraint, (ii) a perpendicular distance constraint, and (iii) an end point constraint. The extent to which the angle constraint is satisfied is based on the angle θ_{AB} (in degrees) between (if needed, extensions of) the line segments A and B in question. The angle is illustrated in Figure 5.59(a), and the membership function $m_{\angle}(\theta_{AB})$ associated with this constraint is shown in Figure 5.59(b).



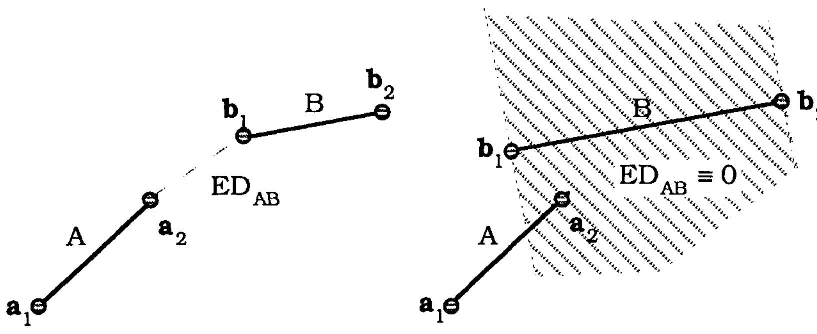
(a) angle between A and B **(b) MF for angle constraint**

Figure 5.59 Angle constraint of Kang and Walker

The end point (Euclidean) distance ED_{AB}, which is proportional to the empty gap distance between the segments A and B, is defined as:

$$ED_{AB} = \frac{\min\{\delta_2(\mathbf{a}_1,\mathbf{b}_1),\delta_2(\mathbf{a}_1,\mathbf{b}_2),\delta_2(\mathbf{b}_1,\mathbf{a}_2),\delta_2(\mathbf{a}_2,\mathbf{b}_2)\}}{\min\{\delta_2(\mathbf{a}_1,\mathbf{a}_2),\delta_2(\mathbf{b}_1,\mathbf{b}_2)\}}, \tag{5.81}$$

where $\mathbf{a}_1$ and $\mathbf{a}_2$ are the end points of segment A, and $\mathbf{b}_1$ and $\mathbf{b}_2$ are the end points of segment B. Figure 5.60(a) illustrates the normal case, for which $ED_{AB} > 0$. When one of the segments is at least partially inside a projected envelope orthogonal to and containing the other, as shown in Figure 5.60(b), then equation (5.81) is *not* used. Instead, ED_{AB} is *defined* to be zero, $ED_{AB} \equiv 0$ (Basically, you find the nearest point on the (infinite) extension of B to $\mathbf{a}_1$ or $\mathbf{a}_2$, and if this point lies between $\mathbf{b}_1$ and $\mathbf{b}_2$, then $ED_{AB} \equiv 0$.) A trapezoidal membership function similar to the one in Figure 5.59(b) is used with the *end point* (EP) distance constraint, yielding $m_{EP\delta_2}(ED_{AB})$.



(a) normal case **(b) overlapping case**

Figure 5.60 Endpoint distance constraint

The third constraint, perpendicular distance, measures how well the extension of one segment is supported by the other. When A and B are primitive (i.e., ungrouped, single line segments), the longer segment is called the *dominant* one of the pair. As illustrated in Figure 5.61, the perpendicular distance PD_{AB} is the orthogonal distance from the mid point ($\blacktriangle$)of the non-dominant segment to the extension of the dominant segment, which is labeled A in the Figure 5.61.

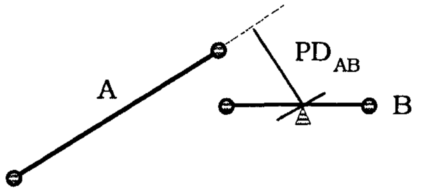


Figure 5.61 Perpendicular distance constraint

Like the end point distance constraint ED_{AB}, we need a membership function for PD_{AB}. Kang and Walker assert that $m_{\perp\delta}(PD_{AB})$ can be increasing or decreasing, and they used a trapezoidal membership function for $m_{\perp\delta}(PD_{AB})$ in their paper. The membership function $m_{COLL}(A, B)$ for *collinearity* of two line segments A and B is obtained by aggregating the values of the three membership functions $m_\angle(\theta_{AB})$, $m_{EP\delta_2}(ED_{AB})$ and $m_{\perp\delta}(PD_{AB})$,

$$m_{COLL}(A,B) = T\left(m_\angle(\theta_{AB}), m_{EP\delta_2}(ED_{AB}), m_{\perp\delta}(PD_{AB})\right). \tag{5.82}$$

Equation (5.82) uses any T-norm such as the min operator. When the value of m_{COLL} for a pair of segments is high enough, a merged segment is created by extending the dominant segment to span the projection of the non-dominant segment. Other ways to create the merged segment may also be used. Kang and Walker defined the *certainty value* CV (A) for a line A with endpoints **a** and **b** that results from grouping a set of smaller line segments $\{A_i\}$ with endpoints $\{(\mathbf{a}_i, \mathbf{b}_i)\}$ as

$$CV(A) = [\sum_{\{A_i\}} CV(A_i) \cdot m_{COLL}(A_i, A) \cdot \delta_2(\mathbf{a}_i, \mathbf{b}_i)] \,/\, \delta_2(\mathbf{a}, \mathbf{b}). \tag{5.83}$$

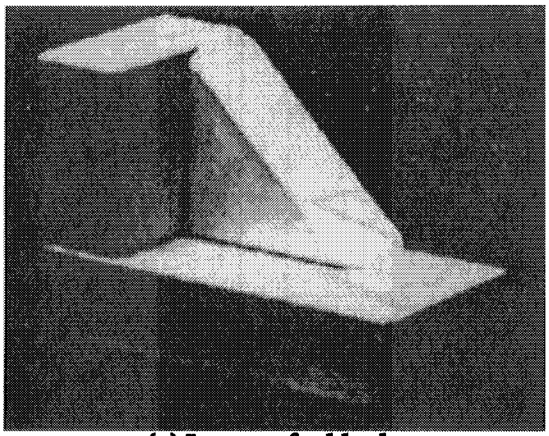
At the lowest level, if the edge detector provides information about the certainty $CV(A_i)$ of an edge fragment A_i, this can be used in (5.83) when determining CV(A). Otherwise, all edge fragments are considered to have equal certainty. For groups of segments, the segment with the greatest product of certainty and length is considered to be the dominant one.

Table 5.21 The collinear grouping algorithm

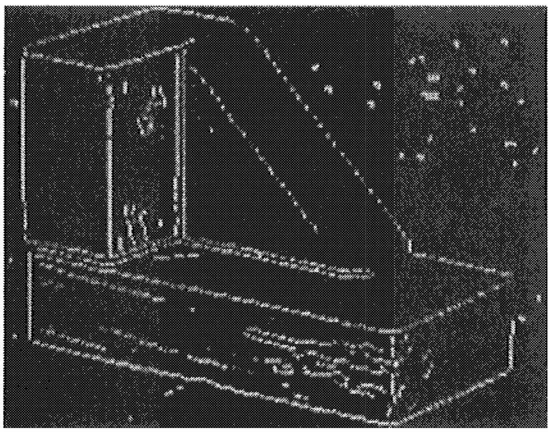
Store	Line segments $X = \{A_1, \ldots, A_n\}$ - (e.g., via an edge detector)
	Certainty values $CV = \{CV(A_1), \ldots, CV(A_n)\}$
Pick	☛ α cut value : $\alpha \in [0,1]$
Do	REPEAT UNTIL (no merger takes place)
	Choose dominant segment A_{seed} as seed
	Compute $COLL_\alpha = \{A_i : m_{COLL}(A_{seed}, A_i) > \alpha, i = 1, \ldots, n\}$
	Sequentially merge $A_i \in COLL_\alpha$
	(in decreasing order of (length $\times$ certainty)
	Add grouped segments from previous step to the data
	Eliminate the seed segment and all merged segments included in grouped segments from the data
	% When you merge several segments, what results is a grouped segment
	END UNTIL

Kang and Walker's (1994) algorithm for collinear grouping is summarized in Table 5.21. Although it may appear that the value of α used in the algorithm acts like a threshold, since the system retains the membership values for all the collinear groups, the uncertainty can be propagated, and the actual decision-making can be deferred.

Example 5.22 This example is adapted from Kang and Walker (1994), and they supplied the images shown. Figure 5.62(a) shows the image of a block and Figure 5.62(b) shows an edge image made from the image in view (a). Kang and Walker do not specify the edge detector used. Figure 5.62(c), shows the initial set of edge fragments extracted from the edge image in view (b).

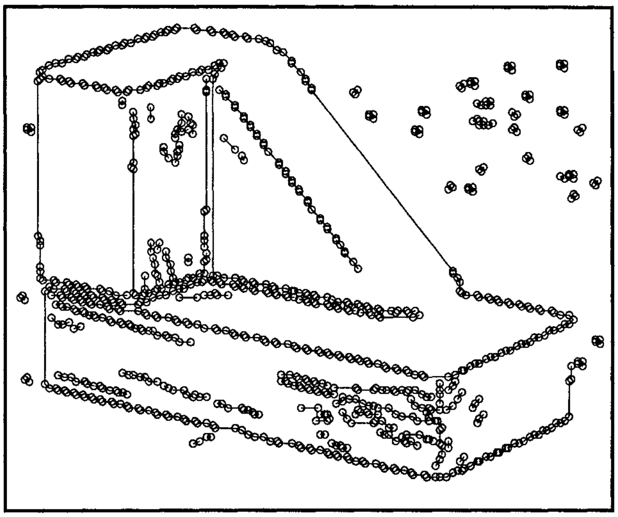


(a) Image of a block



(b) Corresponding edge image of block

Figure 5.62 Raw data for collinear grouping



(c) Initial tokens from edge image

Figure 5.62 (con't.) Raw data for collinear grouping

Figure 5.63 shows a set of five membership functions for the linguistic values {*not, slightly, roughly, almost, exact*}. Kang and Walker (1994) use this termset, with the domain of the membership functions normalized to [0, 1] if necessary, to fuzzify several numerical variables in different parts of their overall system for perceptual grouping. For example, when doing collinear grouping of objects A and B at the *rough* level, the horizontal x axis as shown in Figure 5.63 is interpreted as $x = m_{COLL}(A, B)$, and the vertical axis is interpreted as the extent to which A and B are roughly collinear.

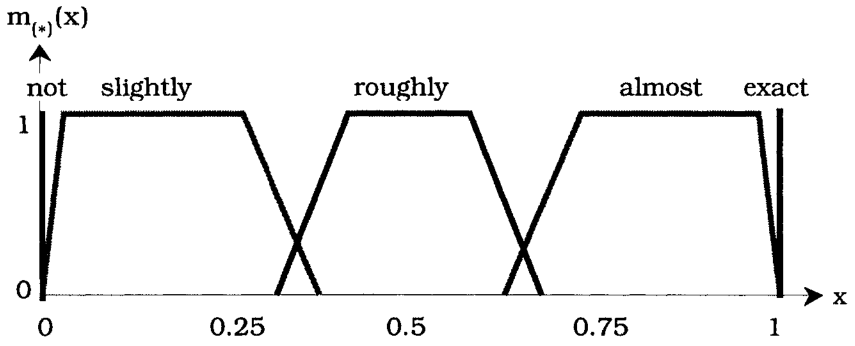
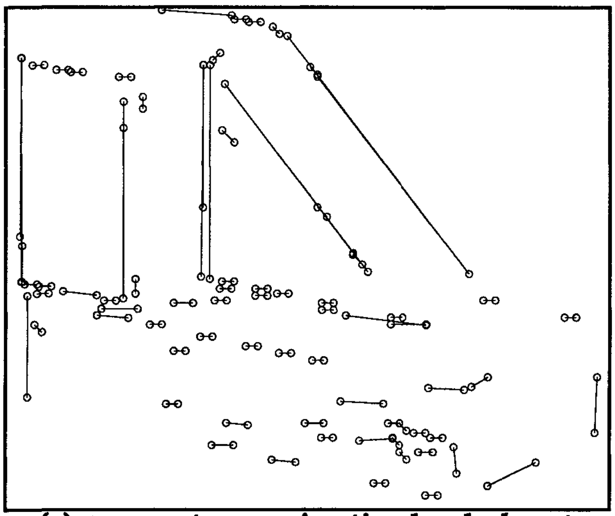


Figure 5.63 Linguistic term set for perceptual grouping tasks

Kang and Walker then use the area centroid of the membership function for ROUGHLY in Figure 5.63 as the α value in the collinear grouping algorithm in Table 5.21. In this example only the function for roughly is used; all five of them are shown here for graphical economy - they are not a termset for the input to a set of rules.

Figure 5.64(a) shows the groupings of the initial edge fragments shown in Figure 5.62(c) at the approximation level *almost* collinear. Figure 5.64(b) and (c) show the groupings at approximation levels *roughly* collinear and *slightly* collinear.


(a) groups at approximation level *almost*

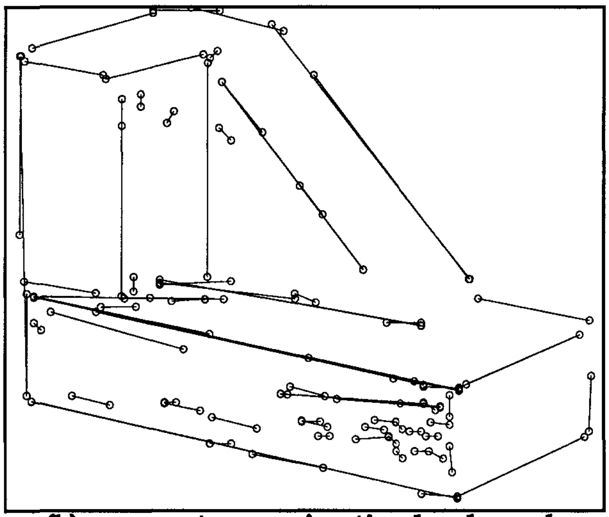
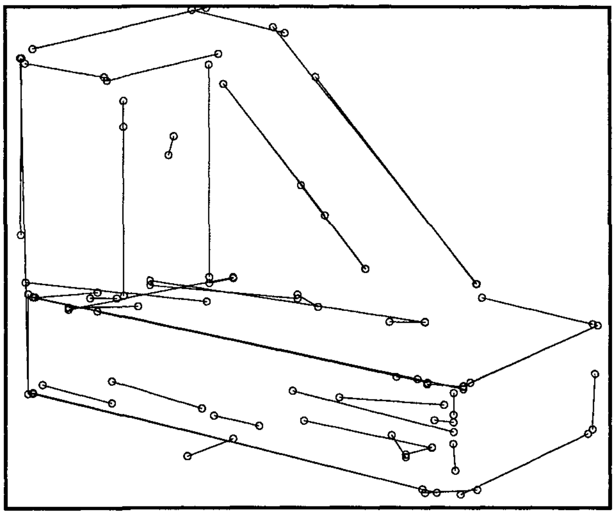

(b) groups at approximation level *rough*

Figure 5.64 Collinear grouping of edge fragments



(d) groups at approximation level *slightly*

Figure 5.64 (con't.) Collinear grouping of edge fragments

5.10 High-Level Vision

High level vision tries to take advantage of goals and knowledge to guide visual activities. Therefore, we need methods to model goals and capabilities and reason about them. A system that employs high-level techniques should also be able to evaluate the success of its approaches. Knowledge representation and object modeling is an important part of high-level vision. Very often, world knowledge can be described in vague and imprecise terms, and therefore, fuzzy methods are ideally suited for this application. High-level vision also requires powerful inferencing methods, and fuzzy logic can play a role here. Other examples of high-level activities to which fuzzy methods can contribute are: matching models to data, belief maintenance, and constraint relaxation.

We illustrate one way to use a fuzzy model in this context with an abbreviated summary of Wang and Keller's (1999a, b) work in high level vision. The fuzzy MA rule base developed for scene description contains 242 rules and was implemented using the software package *CubiCalc* (Watkins, 1990), with product inference (firing strengths are computed with the T_2 norm) and centroid defuzzification (Figure 4.33, and see Klir and Yuan, 1995). Table 5.22 summarizes various combinations of linguistic values for the five premise variables representing the 242 rules used by Wang and Keller.

Table 5.22 Summary of input PMFs for 242 scene description rules

Rule-type	L Left	A Above	R Right	B Below	S Surround
1	x	x	x	x	*h*
2	*m, h*	*m, h*	*m, h*	*m, h*	*m*
3	*m*	*m, h*	*m*	*ℓ*	*m*
4	*h*	*m, h*	*h*	*ℓ*	*m*
5	*m*	*m, h*	*h*	*ℓ*	*m*
6	*m,h*	*ℓ*	*m, h*	*ℓ*	*m*
7	*ℓ*	*h*	*h*	*ℓ*	*ℓ, m*
8	*ℓ*	*m*	*m*	*ℓ*	*ℓ, m*
9	*ℓ*	*m*	*h*	*ℓ*	*ℓ, m*
10	*ℓ*	*m,h*	*ℓ*	*ℓ*	*ℓ, m*
11	*ℓ*	*ℓ*	*ℓ*	*ℓ*	*m*
12	*m, h*	*m, h*	*m, h*	*m, h*	*ℓ*
13	*m*	*m, h*	*m*	*ℓ*	*ℓ*
14	*m*	*m, h*	*h*	*ℓ*	*ℓ*
15	*h*	*m, h*	*h*	*ℓ*	*ℓ*
16	*m, h*	*ℓ*	*m, h*	*ℓ*	*ℓ*

Note: x denotes "don't care" in Table 5.22

Wang and Keller used five spatial relations between objects as linguistic variables about "RELATION BETWEEN" to fuzzify the LHS of the rule base and build a linguistic description of the scene. The premise or input linguistic variables were: LEFT_OF = L, RIGHT_OF = R, ABOVE = A, BELOW = B and SURROUND = S.

Table 5.23 summarizes the various combinations of output linguistic values for the antecedent clauses in Table 5.22. The output or consequent side of their MA rule base was fuzzified with 10 consequent linguistic variables: TL, TS, TR, TB, TA, AR, AL, BR, BL and AM. This notation abbreviates compound words made by juxtaposition of T= TOTAL with L, R, A, B and S. So, for example, TA = TOTALLY_ABOVE, AR = ABOVE_RIGHT, BR = BELOW_RIGHT, etc. AM is a standalone acronym that means "AMONG"; this variable was used for groups of objects, as in "Tank is AMONG the Armored Personnel Carriers". These variables head the columns of Table 5.23. Each input and output linguistic variable took 3 linguistic values : *ℓ* = LOW, *m* = MEDIUM and *h* = HIGH.

Table 5.23 Summary of output CMFs for 242 scene description rules

Rule	TL	AL	TA	AR	TR	BR	TB	BL	TS	AM
1	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	h	ℓ
2	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	m	ℓ
3	ℓ	ℓ	m,h	ℓ	ℓ	ℓ	ℓ	ℓ	m	ℓ
4	ℓ	ℓ	m,h	ℓ	ℓ	ℓ	ℓ	ℓ	m	ℓ
5	ℓ	ℓ	ℓ	m	ℓ	ℓ	ℓ	ℓ	m	ℓ
6	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	m	ℓ
7	ℓ	ℓ	ℓ	h	ℓ	ℓ	ℓ	ℓ	ℓ,m	ℓ
8	ℓ	ℓ	ℓ	h	ℓ	ℓ	ℓ	ℓ	ℓ,m	ℓ
9	ℓ	ℓ	ℓ	ℓ	h	ℓ	ℓ	ℓ	ℓ,m	ℓ
10	ℓ	ℓ	h	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ,m	ℓ
11	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	m	ℓ
12	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	h
13	ℓ	ℓ	m,h	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	m
14	ℓ	ℓ	ℓ	m	ℓ	ℓ	ℓ	ℓ	ℓ	m,h
15	ℓ	ℓ	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	h
16	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	m,h

If there are two adjacent relations, such as ABOVE and RIGHT, with both "medium" or both "high", and the others "low", then the output ABOVE_RIGHT will be "high". Because the 4 primitive spatial relations are symmetric, we just list one case; rules for the other three cases can be obtained by symmetry. For instance, if the inputs are (medium, high, medium, low, medium) for (L, A, R, B, S), then for the outputs, TOTAL_SURROUND will be "medium", TOTAL_ABOVE will be "high" and the others will be "low". This rule is displayed in the third rule-type of Tables 5.22 and 5.23 (note the choice of "high" for ABOVE). Now, if the 4 primitive relations of the inputs turn 90 degrees clockwise, i.e., (low, medium, high, medium, medium) for (L, A, R, B, S), then TOTAL_SURROUND will be "medium", TOTAL_RIGHT will be "high", and the others will be "low".

Among the 10 output values, the variable with the highest confidence value was picked to describe the relation between objects in the scene. This simple linguistic approximation approach produced very good results in the experiments described later. After the system was constructed, it was tuned using two images to adjust some factors which affect the performance of the system, such as definitions of the membership functions, the rules, and grouping parameters. A typical rule summarizing the spatial relation between a pair of objects is found by matching a premise row from Table 5.22 to a consequent in Table 5.23. For example:

Premise clauses from row 3 of Table 5.22

IF	L = *m*
AND	A = *m*
AND	R = *m*
AND	B = *ℓ*
AND	S = *m*

Consequent clauses from row 3 of Table 5.23

THEN	TL = *ℓ*
AND	AL = *ℓ*
AND	TA = *m*
AND	AR = *ℓ*
AND	TR = *ℓ*
AND	BR = *ℓ*
AND	TB = *ℓ*
AND	BL = *ℓ*
AND	TS = *m*
AND	AM = *ℓ*

Wang and Keller (1999a) show that when SURROUND exists, we don't need to consider the four primitive spatial relationships. Thus, if SURROUND is "high", whatever the other four input variables are, TOTAL_SURROUND will be "high" and the other linguistic values will be "low" in the consequent membership function (CMF) set. When SURROUND exists but to a weaker extent, the other 4 primitive spatial relations may exist at the same time. So, if SURROUND is "medium", then TOTAL_SURROUND will be "medium", and the other relations from output will depend on the 4 primitive relations of the inputs. This is shown, for example, in the third rule-type of Tables 5.22 and 5.23, where the inputs are (medium, high, medium, low, medium) for (L, A, R, B, S) - (note the choice of "high" for ABOVE). These outputs give TOTAL_SURROUND: "medium", TOTAL_ABOVE: "high" and the others: "low".

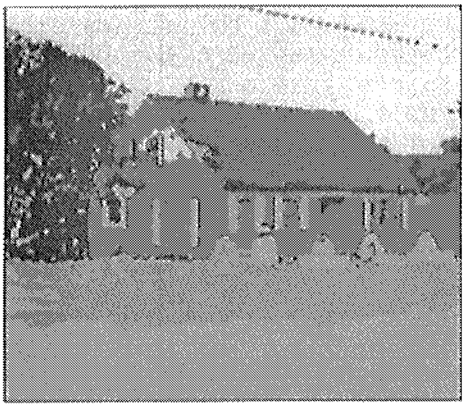
Writing out the rules amounts to simulating a human's reasoning about spatial relations. When humans think about spatial relations, they consider the relations comprehensively. For example, when the spatial relational membership values (0.88, 0.76, 0.00, 0.00) are given for (A, L, R, B), humans might reason that the compound relationship is "ABOVE_LEFT" because "LEFT" is "high" and "ABOVE" is "high" and "RIGHT" is "low" and "BELOW" is "low". Since the system attempts to model the reasoning process of a human expert, the designer can understand the cause of the change in the performance after he or she manipulates the rules and membership functions or uses different parameters. We refer to this tuning process as training of the system. This scene description approach requires *a priori* information (domain knowledge) which can be encoded as separate rules or procedures. For example, in an automatic target recognition application, it was necessary to include procedures to decide if individually detected vehicles should be considered a group or a convoy, or if some buildings should be grouped together for the purpose of scene description. Details and other applications can be found in (Wang and Keller, 1998b). In Example 5.23 we show the results of applying the Wang and Keller rule-based system to the segmentation of the "man-and-house" image in Figure 5.30(d).

Example 5.23 Because angles between all pairs of object points must be computed (perhaps many times), direct computation can be quite costly. An arctangent lookup table of the same size as the image was created to reduce this computational burden. Thus, needed values come directly from the table instead of being calculated for each pair of pixels. Also, before determining the spatial relations between two different objects, we have to decide which pixels belong to the two objects. To reduce computation, we do not need to examine the region labels for all the pixels in the image. The smallest rectangles containing the objects can be generated automatically after labeling. We just search the rectangle areas, which can be much smaller than the entire image.

Table 5.24 Spatial relationship values for Figure 5.30(d)

arg, ref	L	A	R	B	S	Output
roof, tree	0.00	0.17	0.70	0.05	0.00	TR = 0.99
wall, tree	0.00	0.02	0.61	0.26	0.00	TR = 0.79
sky, tree	0.01	0.40	0.69	0.01	0.00	TR = 0.98
wall, roof	0.13	0.01	0.01	0.73	0.00	TB = 1.00
sky, roof	0.07	0.68	0.12	0.00	0.00	TA = 0.93
lawn, wall	0.06	0.01	0.02	0.76	0.00	TB = 1.00
road, wall	0.08	0.00	0.03	0.83	0.00	TB = 1.00
road, lawn	0.06	0.01	0.03	0.68	0.00	TB = 0.90

Columns L, A, R, B and S in Table 5.24 show (spatial relation) membership values for the labeled regions in Figure 5.30(d) before invoking the rule base; these are the input values to the PMFs in the 242 rules. The neural network approach of Wang and Keller (1995) was used to obtain the input values in Table 5.24. The output variable having the maximum value after invoking the rule base is displayed in the last column of Table 5.24. The crisp output of the rule base after hardening by linguistic approximation is shown in Figure 5.65 (b). Figure 5.65(a) replicates Figure 5.30(d), the scene to which the rule-based statements apply.



The roof is right of the tree

The wall is right of the tree

The sky is right of the tree

The wall is below the roof

The sky is above the roof

The lawn is below the wall

The road is below the wall

The road is below the lawn

(a) Figure 5.30(d), repeated **(b) rule based relationships**

Figure 5.65 Output of the scene description rules for Figure 5.30(d)

Figure 5.30(d) shows that some parts of the image were labeled incorrectly. For example, some roof edges were labeled tree, and the black shutters were labeled road, etc. But these parts did not dominate the regions to which the parts were assigned, and so the system still gave very good results. Wang and Keller's fuzzy rule-based system also works well on other outdoor scenes.

5.11 Comments and bibliography

On stuff we did not cover

A better title for this chapter would be "Selected Topics in Computer Vision and Image Processing", since the algorithms presented in this chapter are just a few needles in the haystack of literature on fuzzy models in this field. There have been many recent surveys that discuss the use of very innovative and clever fuzzy models in various imaging applications that we did not discuss. For example, Pal and Pal (1993) give a good exposition of many topics that received scant attention here. Keller et al. (1996) give an in depth discussion of the use of rule based systems in computer vision - touched on here only briefly. Bezdek and Sutton (1999) provide a fairly comprehensive review of fuzzy models for medical imaging, with concentration on segmentation and edge detection in tumor detection. Most of the textbooks we are aware of have pretty dispersed coverage of image processing, and are content to scatter an example here and there. Notable exceptions are the books by Pienkowski (1989) and Chi et al. (1997).

The topic of how to evaluate image processing algorithms always sparks a lively debate. Since humans excel at visual pattern recognition, it is easy for most of us to tell really bad outputs from good ones. But it is nearly impossible to see a five percent improvement (which may be enough to save a life), much less measure it, in, say, a segmentation of a digital mammogram to detect microcalcifications. Even when participants are domain experts (e.g., radiologists evaluating MR segmentations), there is a lot of (perhaps unavoidable) subjectivity in performance analysis.

Heath et al. (1998) provide an interesting and thought provoking article on performance evaluation in image processing. The context of their paper is edge detection, but their ideas deserve careful attention by workers in related areas. For example, Table 1 in Heath et al. lists all edge detection papers that were published in the four journals *IEEE Trans. PAMI*, *CVGIP*, *Image Understanding* and *Pattern Recognition* from 1993-1995. 21 papers are referenced - *none* used real image ground truth! Table 2 in Heath et al. lists 12 papers that have described evaluation methods for edge detection - 8 of the 12 require ground truth! What's wrong with this picture? Most of us are computer professionals, engineers or mathematicians. We develop image algorithms, grab a few images, run them, choose the ones we like, and rush to judgment. That's basically what we did when we wrote this chapter. Heath et al. advocate the use of quantitative rating instruments used by human panelists who visually compare outputs of (more or less) comparable algorithms that have been exercised in a carefully controlled and integrated software setting. We think this is a very useful paper; have a look at it.

Tanaka and Sugeno (1991) provide a different twist to image evaluation. Their work focuses not on the comparative evaluation of image processing algorithms, but rather, on evaluation of how different humans evaluate the content of color images. They build a two stage evaluation model based on the Choquet integral (Section 4.5) that aims to understand how different humans perceive and respond to visual images, and in particular, how they select the most preferable reproduction of a color photograph.

An important topic that we have virtually ignored in this chapter is *image compression* (probably because none of us know much about it!). Fuzzy models have been used pretty successfully in this area, competing well with more standard approaches to vector quantization. Karayiannis and Pai (1996) and Karayiannis (1997b) show some results using various mutations of the FLVQ method (Section 4.3) to compress and reconstruct poor Lena, and they report very favorable signal to noise ratios using their FALVQ family of algorithms. Wang and Karayiannis (1997) use FLVQ and other techniques to compress digital mammograms of breasts containing microcalcifications. Kosko (1992) has a chapter on this topic written by Kosko and Kong that utilizes fuzzy associative memories and competitive learning models for image coding. Kosko (1997) has a chapter written by Kim and Kosko that extends this work with subband coding and vector quantization.

Wang et al. (1996) use fuzzy reasoning for image compression. Their method adaptively adjusts the 3D position of triangular patches that approximate the corresponding luminance curved surfaces of the original image. The adaptive adjustment is done considering all pixels contained in the projection of a patch using a six-rule fuzzy reasoning system.

Another important topic that we have paid scant attention to is image processing based on *fuzzy mathematical morphology*; and again, it's the case that none of us know very much about it. A large body of work on this topic is due to Bloch and her colleagues, and for an introduction to the literature in this area we can do no better than to recommend her eminently readable survey (Bloch, 1995). Another important source for fuzzy morphology in image processing is the work of Dougherty and his colleagues, which is well summarized in Sinha and Dougherty (1995), and which comes to fruition in Sinha et al. (1997).

Binary (crisp) morphological operators such as erosion, dilation, opening, etc. are built on the concept of fitting a structuring element to structures in the image. We think that Goetcherian (1980) first discussed fuzzification of crisp morphological operators by applying standard operators to α-cuts of fuzzy images, and aggregating the results over the α-cuts to get a final result. Bloch and Maitre (1993) also suggested fuzzification of morphological operators in the context of α-cuts.

Sinha and Dougherty (1992) introduced intrinsically fuzzy morphological operators, in which both the input and output images are fuzzy. Erosion is defined using fuzzy set inclusion to represent the idea of "more or less fits" (the structuring element). Sinha and Dougherty (1995) propose a set of axioms that seem desirable for these operators based on a fuzzy subset-hood index, and provide an extensive study of properties of their system. A general paradigm for "lifting" crisp morphological algorithms to fuzzy generalizations is given in Sinha et al. (1997). These authors discuss algorithms for three important image processing tasks: shape detection, edge detection and clutter removal. An example of their fuzzy morphological approach to word recognition is provided.

Bloch et al. (1997) apply the morphological approach to three-dimensional reconstruction of blood vessels to assist vascular lesion interpretation. Fuzziness is incorporated in four different areas (segmentation, modeling imprecision, mathematical morphology and data fusion), without a priori geometrical model information. The approach fuses information from digital angiographic and echographic data to make a final binary decision, resolving possible contradictions between the two modalities. Redundancy in the combined data from two orthogonal X-ray angiographic projections (which provide longitudinal information and overall 3D geometry of the vessel), and a series of endovascular echographic slices (high resolution vessel cross sections) helps to reduce imprecision and uncertainty in the final model. Segmentation of the endovascular echographic images is based on fuzzy classification and mathematical morphology, whereas the digital angiographic images are segmented based on dynamic tracking of vessel centerlines and contours. During model reconstruction, fuzzy dilation is used to handle the spatial imprecision of the detected contours. The data are then fused using a fuzzy operator, and a binary decision about the contour is based on a 3D watershed algorithm which connects maximal membership points. Example images from a dog aorta are provided.

Park and Keller (1997) developed a segmentation approach to detect white blood cells in human bone marrow images that combines mathematical morphology with fuzzy relaxation labeling. Other work in this area includes DiGesu et al. (1991) and Koskinen et al. (1991).

Feature analysis in image processing

Perhaps the most important choice you will make in image processing is which numerical features to extract from raw sensor data. We cannot give you a set of guidelines for getting good features, because here, as in Chapters 2 and 4, "good" is very problem dependent - what are the features used for? what properties of a fuzzy model do we hope they match up with? what type of data do we have? And so on. Many authors have fuzzified input features to classifier networks, including features used in image processing. And of course all of the papers discussed in this chapter use features, but these are rarely selected or extracted using fuzzy techniques. Most always, conventional features are used in fuzzy models and algorithms.

Shell type clustering algorithms, for example, provide a way to extract features related to boundaries of objects in images. The divide and conquer noise fuzzy clustering method presented in Section 5.6 combines the *Hough transform* (HT) with fuzzy shell clustering towards this end. There are many alternatives to the methods we have discussed. Illingworth and Kittler (1987) discuss an adaptive (crisp) Hough transform which probably could be made better by generalization to the fuzzy domain.

The concept of the fuzzy Hough transform was introduced by Han et al. (1993). In Han et al. (1993) fuzziness is used to generalize the HT by extending the HT to points that have memberships other than 0 or 1. The membership degrees are summed to aggregate information about points on a circle, just as the membership value of 1 is added in the ordinary HT. The membership functions for the fuzzy points are taken to be isotropic, i.e., all α-cuts are disks. This approach is used by Philip et al. (1994) to extract features from medical images. Geci (1996) defined another fuzzy Hough transform that makes the angle as well as the spatial location of each point fuzzy, and used it to determine stained cell counts in images of rat livers.

Bhandarkar, (1994) proposed a fuzzy probabilistic model for the *generalized Hough transform* (GHT) based on qualitative labeling of scene features and used it in object recognition and localization. A popular paradigm for model-based vision is recognition via localization, which is banked up on propagation and satisfaction of local constraints arising from matching of local geometric features. The GHT is a frequently used technique for constraint propagation and satisfaction. The GHT works well when the scene has a single object or there are no occlusions of objects in a multi-object scene. Multiple object scenes with partial occlusion results in a combinatorial explosion in the size of the search space of possible scene interpretations and generates several spurious scene hypotheses.

The conventional GHT computes a range of transform values for a given match between a scene feature and a model feature. The range of transform values is represented by a volume in the Hough space (accumulator) H, and all buckets in H that intersect this volume are incremented. For a given quantization level of H, the main reason for this redundancy of GHT is uncertainty in the computed parameters due to occlusion. For occluded objects, using the lengths of the scene feature (s) and the length of the model feature (m), the author defines a "degree of occlusion" measure m_{sm}. Values of m_{sm} are viewed as the extent to which the qualitative attribute "occlusion" is satisfied. In this fuzzy generalization of the GHT (Bhandakar calls it the *weighted GHT* (WGHT)), if a bucket intersects the volume in the parameter space defined by the match, then the bucket count is incremented by the fuzzy membership value m_{sm}. Note that for the GHT this increment is 1. As a result, the WGHT tends to favor matches with unoccluded features over those with occlusion. Unoccluded features typically correspond to objects on top of others, and WGHT favors their recognition. This is very appropriate. According to the author, for the WGHT the Hough accumulator which corresponds to transform values with high redundancy factors are selectively de-emphasized.

One of the central ideas in microcalcification studies for digital mammography is the use of wavelet-based correlation filters to extract features that can be used as a basis for discriminating clusters of microcalcifications. Wang and Karayiannis (1997), Strickland and Lukens (1997) and Strickland and Theodosiou (1998) all use wavelet-based features with various fuzzy models in digital mammography. Runkler and Bezdek (1997) propose several fractal-like features that are derived from images, and illustrate their use for segmentation of a digital mammogram with several fuzzy models.

Li and Yang (1989) give an image enhancement technique based on fuzzy relaxation. Lee and Hsueh (1995) proposed a simple filter based on fuzzy reasoning for noise removal. They first convert the digital image into a fuzzy one where each pixel intensity represents the degree to which the pixel is uniform with respect to its local surroundings. The fuzzy image is then smoothed using a set of three fuzzy rules. The smooth digital image is finally obtained by defuzzifying the output of the rule-base with the inverse of the fuzzification function.

Edge detection and enhancement

There have been many attempts through the years to improve edge detection and edge enhancement with fuzzy models. However, the issue of how best to do this runs deeper than just "to fuzzify or not to fuzzify". The most important aspect of edge detection may well be the features used, and this issue is independent of the incorporation of fuzzy uncertainty into an edge detection model. To appreciate this, contrast Jain et al.'s view of edge images in Section 5.3 to that of Hall (1979), who states that human psychovisual perception of contrast at some spatial location depends on more than just the gradient or difference in intensity levels between a pixel and its background. What we learn from this is that different authors have very different ideas about edges in images, so there are many models of edge detection, and while the gradient is often predominant, many other numerical features are also used in some of these models.

The first work on fuzzy edge detection was apparently Pal and King (1983a). Tyan and Wang (1993) use gray level values as input variables to a fuzzy rule based edge detector. Two fuzzy sets, *bright* and *dark*, are defined on the gray level domain. Their idea of fuzzy edge detection is based on the following heuristic rule:

IF	a *dark* region and a *bright* region meet
THEN	the transition area is an edge

Tyan and Wang use a 2×2 mask. There are 16 cases where *dark* or *bright* pixels can occur in a given 2×2 window. Out of these, there are 4 cases where an edge occurs, and 12 cases where a non-edge occurs. Tyan and Wang build a fuzzy rule for each case, so there are 16 rules in the rule base.

Tao et al. (1993) use gray-level differences between a center pixel and its eight neighboring pixels. Two linguistic labels, *small* and large are used for the input gray level differences. Sixteen structures corresponding to possible edge configurations are considered using the small and *large* linguistic labels. One rule is associated with each edge structure.

In psychophysiology the *perceived contrast* C_{ob} between an object o and its background b is the ratio of the absolute difference in illumination between o and b to the average intensity of the surroundings, $C_{ob} = |I_o - I_b| / I_b$, where I_o and I_b denote the intensities of o and b. Pal and Mukhopadhyay (1996) argue that most edge detection models ignore the wide variation in the perceived contrast over the scale of intensities encountered in real images (Buchsbaum, 1980). They propose a simple edge detector that attempts to integrate psychovisual theory with MA style fuzzy reasoning as described in Section 4.7, and thus, call their model a *psychovisually motivated fuzzy reasoning edge detector* (PSYFRED). Here are the basic elements of their approach, the forerunner of which was ostensibly Bezdek and Shirvaikar's (1994) *fuzzy reasoning edge detector* (FRED), which had roughly the same architecture but very different input features.

Let $I_{\mathcal{B}}$ denote the background intensity at location X_i, the pixel under consideration. Among the many possibilities for computing $I_{\mathcal{B}}$ from, say, a 3×3, window W_i centered at X_i, Pal and Mukhopadhyay use the average intensity in the window. The authors then obtain an estimate of the horizontal and vertical digital gradients g_h and g_y, respectively, from the intensities in W_i. This can be done using any of the standard estimates (Sobel, Prewitt, Roberts operators, Gonzalez and Woods, 1992), but these authors use an aggregation operator instead.

The two gradients and the background intensity (as embodied by $I_{\mathcal{B}}$) are used by a *pair* of simple fuzzy rule bases $\mathcal{R}_h$ and $\mathcal{R}_v$ to produce estimates of the strength of an edge, say E_h and E_y, in the horizontal and vertical directions. The inputs to $\mathcal{R}_h$ are $(g_h, I_{\mathcal{B}})$, and the inputs to $\mathcal{R}_v$ are $(g_v, I_{\mathcal{B}})$. $\mathcal{R}_v$ and $\mathcal{R}_h$ are identical except for the inputs. The overall output is the edge strength $E(X_i)$; of the many possible ways to compute this aggregate, Pal and Mukhopadhyay use the maximum, $E(X_i) = \max\{E_h(X_i), E_v(X_i)\}$.

Three linguistic values, {positive big = PB, positive small = PS, zero = ZE} oversee the action of both gradient estimates and the overall edge strength $E(X_i)$; and four linguistic values, {positive big = PB, positive small = PS, medium = ME, zero = ZE} are used for $I_{\mathcal{B}}$, the average intensity of the window centered at X_i. Each rule base in PSYFRED has 12 rules (four of which can be combined into one rule) of the form : *If* g_h is PB *and* $I_{\mathcal{B}}$ is PB *then* E_h is PS. This rule, for example, would give X_i a low edge value even though the gradient is very high because the background intensity is also very high. Defuzzifciation in each rule base was done using the local mean of maximum rule (Klir and Yuan, 1995).

Segmentation

Once features have been extracted, the most frequently used low level image processing operation is segmentation. Some authors also refer to edge images as segmented images, but we classify edge detection as a separate operation. In any case, there are literally hundreds of papers about fuzzy models for segmentation, and we have barely scratched the surface of this vast and important subject. Just to give you a feel for the extent of this topic, we briefly discuss a very few of these articles.

First, many, many studies and even entire textbooks of non-fuzzy segmentation methods have been published. For example, Morrison and Attikiouzel (1994) describe segmentation by statistical and neural network models; Jain and Flynn (1996) provide a wonderful survey of image segmentation by non-fuzzy cluster analysis. Fuzzy rule-based segmentation has been discussed by many authors (Keller et al., 1996).

There are many ways to classify segmentation methods, none of which leads to a crisp partition of them. For example, Dellipiane (1997) gives a classification tree rooted at image segmentation that subdivides segmentation algorithms based on the parameters that guide them to their goal. Dellipiane identifies three main groups of segmentation methods based on *density*, *topology* and *geometry*. All of the methods covered in our Chapter 5 fall into the first and perhaps oldest category (density), where leaves at a depth of 5 in Dellipiane's tree include segmentation approaches for regions (2D regions or 3D volumes); and for boundaries (2D edges or 3D surfaces).

Perhaps the leading source of fuzzy models for image segmentation is in medical computing. First, there is a rich variety of imaging sensors (PET, X-Ray, MR, CATSCAN, Sonic, Gamma, etc.), all of which produce diagnostically useful information to clinicians and physicians. Second, there are powerful economic forces driving the development of medical imaging devices. And most importantly, the problems that can be solved with medical imaging are attractive; it is hard to imagine a more rewarding accomplishment than, say, reducing the fatality rate from breast cancer by even a few percent with the use of an imaging technology you helped develop. We have already mentioned the survey by Bezdek and Sutton (1999), which is specialized to medical image processing. We repeat a few of the references to papers discussed by Bezdek and Sutton, and add comments on some other papers that are not discussed there.

Microcalcifications in the female breast often appear as small bright areas in mammogram images (i.e. tiny dots), and are taken as a potential early indication of the onset of a breast tumor. Brzakovic et al. (1990) study a fuzzy pyramid linking scheme for the detection of microcalcifications and nodules. Lo et al. (1996) focus on the detection of clustered microcalcifications using fuzzy classification modeling.

Strickland and Lukens (1997) and Strickland and Theodosiou (1998) discuss the use of a TS fuzzy system for the detection of microcalcifications in digital mammograms. They process images by first applying a wavelet filter, and then using a TS system with eight rules to classify pixels in the image. The TS system is trained with labeled data which is manually extracted from the images. Surprisingly, this is one of the few applications of fuzzy models to mammography that we are aware of; see Bezdek and Sutton (1999) for several others.

Sameti and Ward (1996) begin segmentation with an initial fuzzy membership function m_{X0} whose domain is P_{IJ}. First, these authors normalize the gray levels; then they find T, the value at which the histogram of the normalized intensities $\{\hat{I}_{ij}\}$ minimizes. T is used to set the crossover point where $m_{X0}(T) = 0.5$. The graph of m_{X0} is displayed in their paper, but its equation is not. The function shown bears a striking resemblance to a truncated unipolar sigmoid. Following initialization, an iteration scheme that mimics gradient descent updates m_{X0} until a termination criterion is satisfied, resulting in a binary image (i.e., a crisp 2-partition of the image). This procedure is subsequently applied repeatedly to each of the two crisp subsets created by successive phases of the processing until a satisfactory segmentation of P_{IJ} is obtained. Consequently, P_{IJ} is segmented into c crisp regions where $c = 2^k \, \exists k$. Sameti and Ward segment 20 MR images into c = 4 crisp regions this way, and allude to comparing suspicious regions in them to known suspicious regions. Details about the evaluation procedure are incomplete.

Hata et al. (1997, 1998) have an approach to the segmentation of medical images based almost entirely on reasoning with a set of fuzzy if-then rules. Both referenced papers describe the use of a fuzzy rule base that processes numerical pixel-based features. Numerical features include intensities, spatial locations, Euclidean distances and boundary proximities. Membership functions for the rules are given, but no tuning or training is described. Instead, the shapes and parameters of these functions are evidently based on domain specific knowledge about human physiology, such as intracranial structure (for MR brain images), and joint structure (for *computerized tomagraphic* (CT) images of the human foot). Hata et al. (1997, 1998) use a fuzzy rule base to represent a single fuzzy membership function $m: \mathbf{P}_{IJ} \mapsto [0,1]$ on digital images, and then apply region growing based on thresholding the values $\{m(i,j)\}$ to segment $\mathbf{P}_{IJ}$ into a prespecified number of crisp clusters.

Many authors have used one of the c-means models or a derivative of one for image processing on a wide variety of medical imagery. Boudraa (1997) and Boudraa et al. (1993) concentrate on cardiac images and FCM processing. Brandt and Kharas (1993) compared the effectiveness of HCM, FCM and PCM for unsupervised segmentation to separate three simulated clusters in brain images as the amount of boundary overlap is increased. Rezaee et al. (1995) combine FCM with the Hough transform to segment MR images of the ventricle.

Namasivayam and Hall (1995) assert that over a large set of MR images from different patients, rules perform reliably when they are based on relative differences in pixel intensities for different tissue types. These authors state that fuzzy rules and ssFCM applied to the unlabeled pixels in test MR images of normal patients can yield more accurate and much faster segmentation than naive FCM segmentation (but see our discussion about crisp rules for the images in Figure 5.31). In this application crisply labeled training pixels are chosen by a set of rules that identify tissue types with a high degree of confidence.

A very different approach to supervised segmentation from the methods discussed in this chapter is *region growing* from *user selected* seed pixels (or voxels). Delliapiane et al. (1996) give a *fuzzy isovolumes* approach to segmentation of 2D and 3D images based on this idea. A connectivity measure based on fuzzy topology and homogeneity is constructed from image intensities and is used to drive the segmentation process. Supervision is begun by an expert user, who interactively chooses a single pixel (or voxel) from a known class as the *seed* for a region (or volume) growing technique (the training data is thus a crisply labeled singleton). One class at a time is built by thresholding an image that possesses a property they call (fuzzy) *intensity connectedness*, which is an extension of Rosenfeld's (1984) idea of fuzzy connected components in a digital image. A number of potential fuzzy isovolumes are grown from the selected seed, and the operator then chooses the most appropriate one before proceeding to the next region (tissue class) in the image. This style of segmentation proceeds non-iteratively, one region at a time, and is terminated by a human expert - not an algorithmic criterion.

Udupa et al. (1997b) also discuss segmentation models based on various topological notions of fuzzy connectedness and region growing. Pixel intensities are used as a basis for measures of fuzzy similarity between image elements in the same tissue class of various medical images. Their technique is, like Dellipiane et al.'s, initiated by a user chosen seed for each object. Image segmentation and object classification are achieved by thresholding a fuzzy relation in the given image, resulting in various output images. These authors give some nice examples of their model to visualization and rendering of lesions in multiple sclerosis patients.

Herndon et al. (1996) discuss the use of crisply labeled training data created by pooling opinions from experts who label *every pixel* in training images. This training data are then used to derive a classifier that segments an input image into c "fuzzy tissue images", one for each labeled tissue class. Technically, this classifier is *possibilistic* since the pixel memberships assigned to the c tissue images are not constrained to sum to 1. This method of segmentation is very different than the other types discussed so far, and is subsequently used for tissue volume estimation in normalized T1 MR images.

Bombardier et al. (1997) investigate automated enhancement strategies in digital subtraction angiography. In this work two cooperating fuzzy segmentation operators based on textural and geometric properties are used to successively enhance aorta and renal artery boundaries. First, fuzzy linguistic rules are derived from their definition of an edge as "...a high transition near an homogeneous region". These rules are applied as a set of 5×11 masks over the whole image to find characteristic homogeneous and heterogeneous regions indicative of aorta outlines. Second, bifurcation points along these outlines then determine the regions of interest where subsequent analysis using an FCM-based edge operator extracts renal artery boundaries. Results are provided for a real 2D angiogram. The final edge image created would still need to be post-processed to characterize any lesion boundary abnormalities (e.g., narrowing of the artery diameter, as in stenosis).

Much remote sensing work has been done with fuzzy models for segmentation of aerial photographs, LANDSAT images, SEASAT images, etc. Cannon et al. (1986b) use FCM to segment a thematic mapper image. Fuzzy models have been used in remote sensing applications such as tax assessment, crop damage, thermal pollution, bioresources analysis, and so on. Chi and Yan (1993) segment map images based on fuzzy rules and thresholding. Other representative literature includes Burroughs and Frank (1996), Fisher and Pathirana (1990), Gopal and Woodcock (1994), Wang

(1990a, b), Canters (1997), Blonda et al. (1991, 1996b, c), and Binaghi et al. (1996, 1997).

Roux and Desachy (1997) use an interesting combination of possibility theory and representation of fuzzy rules by neural networks to segment a LANDSAT image. The image is 4 band data, augmented by information about the image region such as distance to rivers, elevations, etc. Each image provides training data pixels chosen by an operator in one of c = 9 classes that are related to vegetation and cultivation. A possibility distribution is assigned for each band from histograms of the labeled pixels. Another possibility distribution is obtained for the geographical data using a set of neural networks that represent rules obtained from a photo interpretation expert. The possibilities are then fused with a conjunctive fusion operator (Bloch, 1996c) to create a final decision stream for each of the 9 classes. In operation, the possibility distribution for an unlabeled pixel is computed, and hardened in the usual way to yield a crisp label.

Another fertile area for fuzzy image processing is the analysis of (non aerial) color and black and white photographs. Here we meet applications in fingerprint analysis, face recognition, environmental monitoring, etc. Lim and Lee (1990) segment color images with a two stage coarse-fine strategy based on FCM partitions, and compare their method to several crisp segmentation techniques. Araki et al. (1993) segment photographs of office scenes to identify the occupants in a room using a region growing technique combined with FCM and several of the validity criteria we discussed in Section 2.5. Moghaddamzadeh and Bourbakis (1997) discuss the combination of edge detection and region growing approaches to segmentation using two fuzzy criteria. Applications to both image compression and object detection are described and illustrated with a color photograph of a pic (ture) of (pickled) peppers, a house, some fruits, and of course, last and always, Lena.

Trivedi and Bezdek (1986) proposed an unsupervised segmentation algorithm for aerial imagery based on FCM clustering that used a hierarchical pyramidal data structure to represent regions at different resolutions in different levels of the pyramid. A homogeneity test is done on the FCM determined regions at a particular level to decide whether regions should be split at the next level. A recent twist on using multiple resolutions appeared in Tolias and Panos (1998), who combine fuzzy segmentation at different resolutions with an "adaptive" fuzzy clustering segmentation scheme that is a hybrid algorithm. No objective function is optimized: rather, update equations based on FCM/PCM that localize the prototypes to neighborhoods in the image are defined using heuristic arguments. These authors demonstrate their method by segmenting -who else, but ? Lena. Zugaj and Lattuati (1998) discuss the fusion of region and edge segmentation outputs -

two types of algorithmic information - for segmentation of color images.

Fuzzy logic has been successfully used in human face characterization (Grabisch et al., 1996, Figue et al., 1998). Grabisch et al. proposed a multi-stage scheme for interpretation of human faces. The system has three complementary channels for extraction of (1) face boundary, (2) face features like eyes, mouth, eyebrows, and (3) detection of eyes. Each channel has three stages : segmentation of objects of interest from the raw data; quantitative characterization of the extracted regions and interpretation of the segments. The interpretation stage integrates domain knowledge and the features extracted from the segmented regions using fuzzy rules. Finally, the outputs from all three channels are fused again using fuzzy reasoning. The authors report quite satisfactory performance of their system in identifying eyes, mouth, eyebrows etc.

Digital surfaces and boundary representation

Anderson and Bezdek (1984) and Bezdek and Anderson (1985) give a method for solving problems of the type illustrated in Example 5.11. Their scheme finds corners in linear fits to data based on the use of fuzzy c-lines clustering and thresholding of cluster memberships. Corners are defined using commutators (functions of the eigenvalues) of scatter matrix pairs of points in the plane. Examples using both spatial coordinates and chain coded data sets in the plane are very similar to the results in Figure 5.37. However, the number of clusters must be specified a priori, and is fixed during the iterative procedure.

Section 5.7 presented a few of the many concepts that may be useful for accurate description, rendering, visualization and analysis of objects and object regions in 2D images. As 3D sensors become more widespread, generalization of the material in Section 5.7 to multidimensional digital images is inevitable. Udupa (1992, 1994) provides an in depth treatment of some aspects of this evolving discipline for the crisp 3D and multidimensional cases. Udupa and Samarasekera (1996) give a very general framework for fuzzy connectedness and object definitions, and illustrate their theory with 3D rendering of patient knee joints. Udupa et al. (1997a) use their definitions of fuzzy affinity, adjacency and connectivity for the multidimensional case, and give an application of their fuzzy model in an interactive system for viewing 3D renderings of blood vessels.

High level vision and spatial relations

Defining spatial relations is a popular topic nowadays. Del Bimbo and Vicario (1998) discuss spatial relationships as if no previous work had been done with them. Why? Because humans are good at perceiving spatial relationships and we all want to try out our intuition. How do you pick an appropriate definition for a spatial relation? The development in Section 5.8 argues intuitively or uses comparisons of reactions by human subjects in a limited study. In computer vision the payoff is whether or not spatial relation features can be used as features, either to recognize objects or describe the regions in a scene (Wang et al., 1997, Wang and Keller, 1999a, b).

Literature on the application of fuzzy methods to high-level vision problems is rather sparse. This might be partially attributed to the fact that interest in fuzzy methods peaked only in the 90's, well after most researchers in computer vision abandoned developing large rule-based computer systems that involved high-level vision techniques. However, there are some notable exceptions. Miyajima and Ralescu (1993) discuss how fuzziness can be incorporated in modeling object attributes and in matching. Zhang and Sugeno (1993) propose a memory model that contains the necessary knowledge for scene understanding. Fuzzy sets are used to represent the knowledge and fuzzy logic is used for reasoning. Gasós and Ralescu (1995) discuss how (fuzzy) knowledge about location and size of objects can be used to guide object recognition and scene interpretation. Kawade (1995) discusses how to represent and use (fuzzy) knowledge in an interactive vision system to recognize objects in a dynamic environment. Nakagawa and Hirota (1995) discuss an image understanding system for road scenes that uses fuzzy if-then rules to incorporate weather conditions in the knowledge base and generates answers to user queries.

There are several other aspects of image processing and computer vision where fuzzy methods can be used effectively. For example, model-based interpretation of 3D reconstruction algorithms has received much attention in the recent years. Specifically, 3D data matching relative to a reference model, produced either interactively or automatically is of crucial importance. Tarel and Boujemaa (1995) proposed a new 3D registration method in three steps. First, view-invariant 3D features in the data as well as in the object model are selected, and 3D matching transformation parameters are obtained by using a combinatorial approach similar to the generalized Hough Transform. Then, coarse 3D object pose is obtained by applying a robust fuzzy clustering algorithm in parameter space. Data participation in the fuzzy clustering process is weighted by their relevance according to a confidence value based on the geometric feature-to-feature correspondence. Finally, local fine fitting is performed between data and the model to obtain accurate 3D registration.

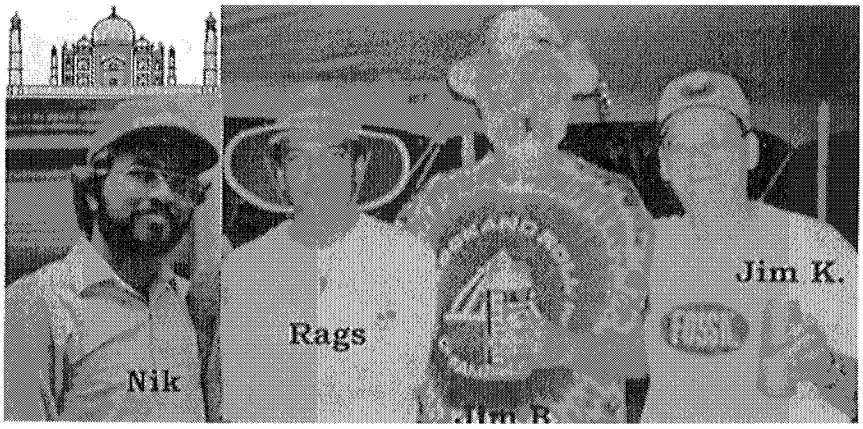
Much work remains to be done in all of the areas covered in Chapter 5 before reliable automatic interpretation of scene content is achieved in fielded applications. Potential areas for the future are:

image matching, solving the correspondence problem, extraction of linguistic descriptions from images, searching image databases based on linguistic queries and hand-drawn sketches, segmentation and volume estimation in 3D (voxel) data, and rule-based high-level vision.

Epilogue

We four have had a lot of fun working in many of the areas covered in this book. Writing the book has taken us a lot more time than we thought it would (*if* four guys who all think they are right all of the time write a book together, *then* there will be endless bickering about small points!)- and it could easily be twice the size it is now. We hope you have had fun reading the book, and that you will look us up when you have a chance, to let us know what you like (love) and dislike (hate) about it. Here is our group portrait. It illustrates the truth of that old saying,

"Never wear a hat that has more attitude than you do"



References cited in the text

Abe S. and Lan M. S. (1995). A method for fuzzy rules extraction directly from numerical data and its application to pattern classification, *IEEE Trans. Fuzzy Systs.*, 3 (1), 18-28.

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA.

Akinniyi, F. A., Wong, A.K. C. and Stacey, D. A. (1986). A new algorithm for graph monomorphism based on the projections of the product graph, *IEEE Trans. Syst., Man and Cyberns.*, 16, 740-751.

Anderson, E. (1935). The Irises of the Gaspe peninsula, *Bull. Amer. Iris Soc.*, 59, 2-5.

Anderson, I. A. and Bezdek, J. C. (1984). Curvature and tangential deflection of discrete arcs; a theory based on the commutator of scatter matrix pairs and its application to vertex detection in planar shape data, *IEEE Trans. Patt. Anal. and Machine Intell.*, 6(1), 27-40.

Anderson, I. A., Bezdek, J. C. and Davé, R. (1982). Polygonal shape description of plane boundaries, in *Syst. Science and Science*, ed. Len Troncale, SGSR Publ., Louisville, KY, 1, 295-301.

Anderson, J. A. (1982). Logistic discrimination, in *Handbook of Statistics 2: Classification, Pattern Recognition and Reduction of Dimensionality*, eds. P. R. Krishnaiah and L. N. Kanal, North Holland, Amsterdam, 169-191.

Anderson, T. W. (1966), Some nonparametric multivariate procedures based on statistical equivalent blocks, *Proc. Int. Symp. Anal.*, ed. P. R. Krishnaiah, Academic Press, NY.

Andrews, D. F. (1972). Plots of high dimensional data, *Biometrics*, 28, 125-136.

Araki, S., Nomura, H. and Wakami, N. (1993). Segmentation of thermal images using the fuzzy c-means algorithm, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 719-724.

Atlas, L., Cole, R., Muthusamy, Y., Lippman, A., Connor, J., Park, D., El-Sharkawi, M. and Marks, R. J. (1990). A performance comparison of trained multilayer perceptrons and trained classification trees, *Proc. IEEE*, 78(10), 1614-1619.

Babuska, R. and Kaymak, U. (1995). Application of compatible cluster merging to fuzzy modeling of multivariable systems, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 565-569.

Back, C. and Hussain, M. (1995). Validity measures for fuzzy partitions, *Data analysis and information systems*, ed. H.H Bock and W. Polasek, Springer, Berlin, 114-125.

Backer, E. (1978). *Cluster analysis by optimal decomposition of induced fuzzy sets*, Delft U. Press, Delft, Netherlands.

Backer, E. and Jain, A.K. (1981). A clustering performance measure based on fuzzy set decomposition, *IEEE Trans. Patt. Anal. and Machine Intell.*, 3, 66-75.

Bagui, S. C. (1993). Classification with the first stage rank nearest neighbor rule for multiple classes, *Patt. Recog. Lett.*, 14, 537-544.

Bagui, S. C. and Pal, N. R. (1995). A multistage generalization of the rank nearest neighbor classification rule, *Patt. Recog. Lett.*, 16, 601-614.

Ball, G. and Hall, D. A. (1967). A clustering technique for summarizing multivariate data, *Behav. Sci.*, 12, 153-155.

Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes, *Patt. Recog.*, 13(2), 111-122.

Baraldi, A. and Alpaydm, E. (1998). Simplified ART : A new class of ART algorithms, Technical Report, International Computer Science Institute, Berkeley, CA.

Baraldi, A. and Parmiggiani, F. (1995). A self-organizing neural network merging Kohonen's and ART models, *Proc. IEEE Int. Conf. on Neural Networks*, IEEE Press, Piscataway, NJ, 2444-2449.

Baraldi, A. and Parmiggiani, F. (1997a). Fuzzy combination of Kohonen's and ART neural network models to detect statistical regularities in a random sequence of multi-valued input pattern, *Proc. IEEE Int. Conf. on Neural Networks*, IEEE Press, Piscataway, NJ, 281-286.

Baraldi, A. and Parmiggiani, F. (1997b). Novel neural network model combining radial basis functions, competitive Hebbian learning rule, and fuzzy simplified adaptive resonance theory, *Proc. SPIE Applications of FuzzyLogic Tech. IV*, eds. J. C. Bezdek and B. Bosacchi, 3165, SPIE, Bellingham, WA, 98-112.

Baraldi, A., Blonda, P., Parmiggiani, F., Pasquariello, G. and Satalino, G. (1998). Model transitions in descending FLVQ, *IEEE Trans. Neural Networks*, 9(5), 724-738.

Barni, M., Cappellini, V. and Mecocci, A. (1996). Comments on 'A Possibilistic approach to clustering', *IEEE Trans. Fuzzy Syst.*, 4(3), 393-396.

Barone, J. M., Filev, D. P. and Yager, R. Y. (1995). Mountain method based fuzzy clustering : methodological considerations, *Int. J. Gen. Syst.*, 23, 281-305.

Baum, E.B. and Haussler, D. (1989). What size net gives valid generalization?, *Neural Computation*, 1, 151-160.

Beckenbach, E. F. and Bellman, R. (1961). *Inequalities*, Springer-Verlag, Heidelberg (3rd printing, 1971).

Bell, E. T. (1966). *Men of mathematics*, 5th printing, Simon and Schuster, NY.

Bellman, R.E., Kalaba, R. and Zadeh, L.A. (1966). Abstraction and pattern classification, *J. Math. Anal. Applications*, 13, 1-7.

Bensaid, A., Bouhouch, N., Bouhouch, R., Fellat, R. and Amri, R. (1998). Classification of ECG pattterns using fuzzy rules derived from ID3-induced decision trees, *Proc. NAFIPS Conf.*, eds. J. C. Bezdek and L.O. Hall, 34-38.

Bensaid, A., Hall, L. O., Bezdek, J. C. and Clarke, L. P. (1996a). Partially supervised clustering for image segmentation, *Patt. Recog.*, 29(5), 859-871.

Bensaid, A., Hall, L. O., Bezdek, J. C., Clarke, L., Silbiger, M., Arrington, J. and Murtagh, R. (1996b). Validity-guided (re) clustering for image segmentation, *IEEE Trans. Fuzzy Syst.*, 4(2), 112-123.

Berenji, H. R. and Khedkar, P. (1992). Learning and tuning fuzzy logic controllers through reinforcements, *IEEE Trans. Neural Networks*, 3(5), 724-740.

Besl, P. C. and Jain, R. C. (1988). Segmentation using variable surface fitting, *IEEE Trans. Patt. Anal. and Machine Intell.*, 10, 167-192.

Bezdek, J. C. (1973). *Fuzzy mathematics in pattern classification, Ph. D. Thesis*, Cornell U., Ithaca, NY.

Bezdek, J. C. (1974a). Numerical taxonomy with fuzzy sets, *J. Math. Bio.*, 1(1), 57-71.

Bezdek, J. C. (1974b). Cluster validity with fuzzy sets, *J. Cyber.*, 3(3), 58-72.

Bezdek, J. C. (1975). Mathematical models for systematics and taxonomy, *Proc. Int. Conf. on Numerical Taxonomy*, ed. G. Estabrook, Freeman, San Franscisco, CA, 143 - 166.

Bezdek, J. C. (1976). A Physical interpretation of fuzzy ISODATA, *IEEE Trans. Syst., Man and Cyberns.*, 6(5), 387-389.

Bezdek, J. C. (1980). A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms, *IEEE Trans. Patt. Anal. and Machine Intell.*, 2(1), 1-8.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum, NY.

Bezdek, J. C. (1992). On the relationship between neural networks, pattern recognition and intelligence, *Int. J. Approx. Reasoning*, 6(2), 85-107.

Bezdek, J. C. (1993). A review of probabilistic, fuzzy and neural models for pattern recognition, *J. Intell. and Fuzzy Syst.*, 1(1), 1-23.

Bezdek, J. C. (1997). Computational intelligence and edge detection, *Proc. AI 5 : Fuzzy Neuro Syst. '97 : Computational Intelligence*, eds. A. Grauel, W. Becker and F. Belli, Infix Press, Zoest, Germany, 1-31.

Bezdek, J. C. (1998). Computational intelligence defined - by everyone!, *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*, eds. O. Kaynak, L.A. Zadeh, B. Turksen and I. J. Rudas, Physica-Verlag, Heidelberg, Germany,10-37.

Bezdek, J. C. and Anderson, I. (1985). An application of the c-varieties clustering algorithms to polygonal curve fitting, *IEEE Trans. Syst., Man and Cyberns.*, 15(5), 637-641.

Bezdek, J. C. and Castelaz, P. F. (1977). Prototype classification and feature selection with fuzzy sets, *IEEE Trans. Syst., Man and Cyberns.*, 7, 87-92.

Bezdek, J. C. and Chiou, E. R. (1988). Core zone scatterplots : A new approach to feature extraction for visual displays, *CVGIP*, 41,186-209.

Bezdek, J. C. and Dunn, J. C. (1975). Optimal fuzzy partitions: A heuristic for estimating the parameters in a mixture of normal distributions, *IEEE Trans. Computers*, 24(8), 835-838.

Bezdek, J. C. and Harris, J. D. (1978), Fuzzy relations and partitions: An axiomatic basis for clustering, *Fuzzy Sets and Syst.*, 1,111-127.

Bezdek, J. C. and Harris, J. D. (1979). Convex decompositions of fuzzy partitions, *J. Math. Anal. and Appl.*, 67(2), 490-512.

Bezdek, J. C. and Hathaway, R. J. (1989). Relational duals of the c-means clustering algorithms, *Patt. Recog.*, 22(2), 205-212.

Bezdek, J. C. and Hathaway, R. J. (1992). Numerical convergence and interpretation of the fuzzy c-shells clustering algorithm, *IEEE Trans. Neural Networks*, 3(5), 787-793.

Bezdek, J. C. and Pal, N. R. (1995). Two soft relatives of learning vector quantization, *Neural Networks*, 8(5), 729-743.

Bezdek, J. C. and Pal, N. R. (1998). Some new indices for cluster validity, *IEEE Trans. Syst., Man and Cyberns.*, C28(3), 301-315.

Bezdek, J. C. and Pal, S. K. (1992). *Fuzzy Models for Pattern Recognition, IEEE* Press, Piscataway, NJ.

Bezdek, J. C. and Shirvaikar, M. (1994). Edge detection using the fuzzy control paradigm, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1, 1-12.

Bezdek, J. C. and Sutton, M. A. (1999). Image processing in medicine, *Applications of Fuzzy Systems*, ed. H. J. Zimmerman, Kluwer, Norwell, MA, in Press.

Bezdek, J. C., Biswas, G. and Huang, L. Y. (1986b). Transitive closures of fuzzy thesauri for information retrieval systems, *Int. J. Man-Machine Studies*, 25, 343-356.

Bezdek, J. C., Chandrasekar, R. and Attikiouzel, Y. A. (1998a). A geometric approach to edge detection, *IEEE Trans. Fuzzy Syst.*, 6(1), 52-75.

Bezdek, J. C., Cheong, F.M, Dillon, T. and Karla, D. (1995). Edge detection using fuzzy reasoning and model-based training, in *Computational Intelligence : A Dynamic System Perspective*, ed. M. Palaniswami, Y. Attikiouzel, R. J. Marks, D. Fogel and T. Fukuda, IEEE Press, Piscataway, NJ, 108-125.

Bezdek, J. C., Chuah, S. and Leep, D. (1986c). Generalized k-nearest neighbor rules, *Fuzzy Sets and Syst.*, 8(3), 237-256.

Bezdek, J. C., Coray, C., Gunderson, R. and Watson, J. (1981a). Detection and characterization of cluster substructure: I. Linear structure: fuzzy c-Lines, *SIAM J. Appl. Math*, 40(2), 339-357.

Bezdek, J. C., Coray, C., Gunderson, R. and Watson, J. (1981b). Detection and characterization of cluster substructure: II. Fuzzy c-varieties and convex combinations thereof, *SIAM J. Appl. Math*, 40(2), 358-372.

Bezdek, J. C., Gunderson, R., Ehrlich, R. and Meloy, T. (1978). On the extension of fuzzy k-means algorithms for the detection of Linear clusters,*Proc. IEEE Conf. on Decision and Control*, IEEE Press, Piscataway, NJ, 1438-1443.

Bezdek, J. C., Hall, L. O., Clark, M., Goldgof, D. and Clarke, L. (1997a). Medical image analysis with fuzzy models, *Statistical Methods in Medical Research*, 6, 191-214.

Bezdek, J. C., Hathaway, R. J. and Huggins, V.J. (1985). Parametric estimation for normal mixtures, *Patt. Recog. Lett.*, 3, 79-84.

Bezdek, J. C., Hathaway, R. J. and Pal, N. R. (1995). Norm induced shell prototype (NISP) clustering, *Neural, Parallel and Scientific Computation*, 3, 431-450.

Bezdek, J. C., Hathaway, R. J., Howard, R. E. and Wilson, C. A. (1986a). Coordinate descent and clustering, *Control and Cyberns.*, 15(2), 195-203.

Bezdek, J. C., Hathaway, R. J., Howard, R. E., Wilson, C. A. and Windham, M. P. (1987a). Local convergence analysis of a grouped variable version of coordinate descent,*J. Optimization Theory and Applications*, 54(3), 471-477.

Bezdek, J. C., Hathaway, R. J., Sabin, M. J. and Tucker, W.T. (1987b). Convergence theory for fuzzy c-means : counterexamples and repairs, *IEEE Trans. Syst., Man and Cyberns.*, 17(5), 873-877.

Bezdek, J. C., Li, W.Q., Attikiouzel, Y. A. and Windham, M. P. (1997b). A geometric approach to cluster validity, *Soft Computing*, 1, 166-179.

Bezdek, J. C., Reichherzer, T., Lim, G. S. and Attikiouzel, Y. A. (1998b). Multiple prototype classifier design, *IEEE Trans. Syst., Man and Cyberns.*, C28(1), 67-79.

Bezdek, J. C., Trivedi, M., Ehrlich, R. and Full, W. (1981c). Fuzzy clustering: A new approach for geostatistical analysis, *Int. J. Syst., Measurement and Decision*, 1/2, 13-24.

Bezdek, J. C., Windham, M. and Ehrlich, R. (1980). Statistical parameters of fuzzy cluster validity functionals, *Int. J. Comp. and Inf. Sci.*, 9(4), 1980, 232-336.

Bhandari, D., Pal, N R. and Dutta Majumder, D. (1992). Fuzzy divergence, probability measures of fuzzy events and image thresholding, *Patt. Recog. Lett.*, 13, 857-867.

Bhandarkar, S. M. (1994). A fuzzy probabilistic model of the generalized Hough transform, *IEEE Trans. Syst., Man and Cybers.*, 24(5), 745-759.

Binaghi, E., Brivio, P. A., Ghezzi, P., Rampini, A. and Zilioli, E. (1996). A hybrid approach to fuzzy land cover mapping, *Patt. Recog. Lett.*, 17, 1399-1410.

Binaghi, E., Madella, P., Montesano, M. G. and Rampini, A. (1997). Fuzzy contextual classification of multiresource remote sensed images, *IEEE Trans. Geosci. and Remote Sensing*, 35(2), 326-340.

Bloch, I. (1996a). Fuzzy relative positions between objects in images: a morphological approach, *Proc. SPIE/EUROPTO Conf. on Image and Signal Proc. for Remote Sensing*, 2955, SPIE, Bellingham, WA, 141-152.

Bloch, I. (1996b). Fuzzy spatial relationships: a few tools for model based pattern recognition in aerial images, *Proc. IEEE Conf. on* Image Processing, II, IEEE Press, Piscataway, NJ, 987-990.

Bloch, I. (1996c). Information combination operators for data fusion: a comparative review with classification, *IEEE Trans. Syst., Man and Cybers.*, A26(1), 52-67.

Bloch, I. and Maitre, H. (1993). Mathematical morphology on fuzzy sets, *Proc. EURASIP Conf. on Math. Morphology Applications Signal Processing*, Barcelona, 151-156.

Bloch, I. and Maitre, H. (1995). Fuzzy mathematical morphologies: A comparative study. *Patt. Recog.*, 28(9), 1341-1387.

Bloch, I., Pellot, C., Sureda, F. and Herment, A. (1997). Fuzzy modeling and fuzzy mathematical morphology applied to 3D reconstruction of blood vessels by multi-modality data fusion, *Fuzzy Information Engineering*, eds. D. Dubois, H. Prade and R.R. Yager, Wiley and Sons, NY, 93-110.

Blonda, P. N., Benardo, A., Satalino, G. and Pasquariello, G. (1996b). Fuzzy logic and neural techniques integration: an application to remotely sensed data, *Patt. Recog. Lett.*, 17, 1343-1348.

Blonda, P. N., Benardo, A., Satalino, G., Pasquariello, G., De Blasi, R. and Milella, D. (1996a). Fuzzy neural network based segmentation of multispectral magnetic resonance brain images, *Proc. SPIE Applications of Fuzzy Logic Technology III*, eds. B. Bosacchi and J. C. Bezdek, 2761, SPIE, Bellingham, WA, 146-153.

Blonda, P. N., Benardo, A., Pasquariello, G., Satalino, G. and La Forgia, V. (1996c). Application of the fuzzy Kohonen clustering network to remotely sensed data processing, *Proc. SPIE Applications of Fuzzy Logic Technology III*, eds. B. Bosacchi and J. C. Bezdek, 2761, SPIE, Bellingham, WA, 119-129.

Blonda, P. N., Pasquariello, G., Losito, S., Mori, A., Posa, F. and Ragno, D. (1991). An experiment for the interpretation of multitemporal remotely sensed images based on a fuzzy logic approach, *Int. J. Remote Sensing*, 12(3), 463-476.

Blonda, P. N., Satalino, G., Baraldi, A. and De Blasi, R. (1998). Segmentation of multiple sclerosis lesions in MRI by fuzzy neural networks: FLVQ and FOSART, *Proc. NAFIPS Conf.*, eds. J. C. Bezdek and L. O. Hall, 39-43.

Bobrowski, L. and Bezdek, J. C. (1991). c-Means Clustering with the ℓ_1 and ℓ_∞ Norms, *IEEE Trans. Syst., Man and Cyberns.*, 21(3), 545-554.

Bookstein, F. L. (1979). Fitting conic sections to scattered data, *CVGIP*, 9, 56-71.

Bombardier, V., Jaulent, M.-C., Bubel, A. and Bremont, J. (1997). Cooperation of two fuzzy segmentation operators for digital substract angiograms analysis. *Proc. Sixth IEEE Int. Conf. on Fuzzy Syst.*, 2, IEEE Press, Piscataway, NJ, 1057-1062.

Boudraa, A. E. (1997). Automated detection of the left ventricle region in magnetic resonance images by the fuzzy c-means model, *Int. J. of Cardiac Imaging*, 13, 347-355.

Boudraa, A. E., Mallet, J. J., Besson, J. E., Bouyoucef, S. E. and Champier, J. (1993). Left ventricle automated detection method in gated isotropic ventriculography using fuzzy clustering, *IEEE Trans. Med. Imaging*, 12(3), 451-465.

Boujemaa N., Stamon G. and Lemoine J. (1992b). Fuzzy iterative image segmentation with recursive merging, *Proc. SPIE, Conf. on Visual Comm. and Image Processing*, 1818, 1271-1281.

Boujemaa N., Stamon G., Lemoine J. and Petit E. (1992a). Fuzzy ventricular endocardiogram detection with gradual focusing decision, *Proc. IEEE Int. Conf. of the Engineering in Medicine and Biology Society*, 14, 1893-1894.

Box, G. and Jenkins, G. (1970). *Time Series Analysis: Forecasting and Control*, Holden Day.

Brandt, M. E. and Kharas, Y. F. (1993). Simulation studies of fuzzy clustering in the context of brain magnetic resonance imaging, *Proc. Int. Conf. on Industrial Fuzzy Control and Intell. Syst.*, IEEE Press, Piscataway, NJ, 197-203.

Brant-Zawadzki, M., Gillan, G. D. and Nitz, W. R. (1992). MP-RAGE: a three dimensional, T1-weighted, gradient-echo sequence, *Radiology*, 182(3), 769-775.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123-140.

Breiman, L., Freidman, J. H., Olshen, R. A. and Stone, C. J. (1984). *Classification and regression trees*, Wadsworth, Inc., Belmont, CA.

Broomhead, D. S. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks, *Complex Syst.*, 2, 321-355.

Brzakovic, D., Luo, X. M. and Brzakovic, P. (1990). An approach to automatic detection of tumors in mammograms, *IEEE Trans. Medical Imaging*, 9(3), 233-241.

Buchsbaum, G. (1980). An analytical derivation of visual non-linearity, *IEEE Trans. Biomed. Engr.*, 27, 237-242.

Buckley, J. J. and Hayashi, Y. (1994). Fuzzy neural networks, in *Fuzzy Sets, Neural Networks & Soft Computing*, eds., R. R. Yager and L. A. Zadeh, Van Nostrand Reinhold, NY, 233-249.

Bunke, H. (1992). *Advances in Structural and Syntactic Pattern Recognition*, World Scientific, Singapore.

Burroughs, P. A. and Frank, A. U., eds., (1996). *Geographic Objects with Indeterminate Boundaries*, Taylor and Francis, London.

Cannon, R., Davé, J. and Bezdek, J. C. (1986a). Efficient implementation of the fuzzy c-means clustering algorithms, *IEEE Trans. Patt. Anal. and Machine Intell.*, 8(2), 248-255.

Cannon, R., Davé, J., Bezdek, J. C. and Trivedi, M. (1986b). Segmentation of a thematic mapper image using the fuzzy c-means clustering algorithm, *IEEE Trans. Geo. and Remote Sensing*, 24(3), 400-408.

Canters, F. (1997). Evaluating the uncertainty of area estimates derived from fuzzy land-cover classification, *Photogrammetric Engineering and Remote Sensing*, 63(4), 403-414.

Carpenter, G. A. (1989). Neural network models for pattern recognition and associative memory, *Neural Networks*, 243-257.

Carpenter, G. A. and Grossberg, S. (1987a). A massively parallel architecture for a self-organizing neural pattern recognition machine, *CVGIP*, 37, 54-115.

Carpenter, G. A. and Grossberg, S. (1987b). ART2 : self-organization of stable category recognition codes for analog input patterns, *Applied Optics*, 26, 4919-4930.

Carpenter, G. A. and Grossberg, S. (1988a). The ART of adaptive pattern recognition by a self-organizing neural network, *Computer*, 21(3), 77-88.

Carpenter, G. A. and Grossberg, S. (1988b). Neural dynamics of category learning and recognition : attention, memory consolidation and amnesia, *Brain Structure Learning and Memory*, eds. J. Davis, R. Newburgh, and E. Wegman, Wegman,Boulder, CO: Westview Press, 233-290.

Carpenter, G. A. and Grossberg, S. (1990). ART3 : hierarchical search using chemical transmitters in self-organizing pattern recognition architectures, *Neural Networks*, 3(2), 129-152.

Carpenter, G. A., Grossberg, S. and Rosen, D. B. (1991a). Fuzzy ART : fast stable learning and categorization of analog patterns by an adaptive resonance system, *Neural Networks*, 4, 759-771.

Carpenter, G. A., Grossberg, S. and Rosen, D. B. (1991b). A neural network realization of fuzzy ART, *Tech. rep.*, CAS/CNS-TR-91-021, Boston, MA, Boston U.

Carpenter, G. A., Grossberg, S., Markuzon, N, Reynolds, J. H. and Rosen, D. B., (1992). Fuzzy ARTMAP : A neural network architecture for incremental supervised learning of analog multidimensional maps, *IEEE Trans. Neural Networks*, 3(5), 698-713.

Carpenter, G. A., Grossberg, S., Markuzon, N. and Reynolds, J. H. (1991c). ARTMAP : supervised real-time learning and classification of non-stationary data by a self-organizing neural network, *Neural Networks*, 4, 565-588.

Chan, K. P. (1996). Learning templates from fuzzy examples in structural pattern recognition, *IEEE Trans. Syst., Man and Cyberns.*, B26(1), 118-123.

Chan, K. P. and Cheung, Y. S. (1992). Correction to "fuzzy attribute graph with application to Chinese character recognition", *IEEE Trans. Syst. Man and Cyberns.*, 22(2), 402-410.

Chang, C.L. (1974). Finding prototypes for nearest neighbor classification, *IEEE Trans. Computer*, 23(11), 1179-1184.

Chang, R.L.P. (1976). Application of fuzzy techniques to pattern recognition and curve fitting, *Ph.D. Thesis*, Princeton U., Princeton, NJ.

Chang, R.L.P. and Pavlidis, T. (1977). Fuzzy decision tree algorithms, *IEEE Trans. Syst. Man and Cyberns.*, 7(1), 28-35.

Chen, C. H. (1996). *Fuzzy Logic and Neural Networks Handbook*, McGraw Hill, NY, Ch. 27.

Chen, S., Cowan, C. F. N. and Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Trans. Neural Networks*, 2(2), 302-309.

Chen, W., Gader P. and Shi, H. (1997). Improved dynamic programming-based handwritten word recognition using optimal order statistics, *Proc. SPIE Conf. on Statistical and Stochastic Methods in Image Processing II*, SPIE, Bellingham, WA, 246-256.

Cheng, H. D., Chen, J.-R. and Li, J. (1998). Threshold selection based on fuzzy c-partition entropy approach, *Patt. Recog.*, 31(7), 857-870.

Cheng, T. W., Goldgof, D. B. and Hall, L. O. (1995). Fast clustering with application to fuzzy rule generation, *Proc. IEEE Int. Conf. of Fuzzy Syst.*, IV, IEEE Press, Piscataway, NJ, 2289-2295.

Chernoff, H. (1973). The use of faces to represent points in *k*-dimensional space, *J. Amer. Stat. Assoc.*, 68, 361-368.

Cheung, Y. S. and Chan, K. P. (1986). Modified fuzzy ISODATA for the classification of handwritten Chinese characters, *Proc. Int. Conf. on Chinese Computing*, Singapore, 361-364.

Chi, Z. and Jabri, M. (1991). A comparison of MLP and ID3-derived approaches for ECG classification, *Proc. Australian Conf. on Neural Networks*, Sydney, Australia, 263-266.

Chi, Z. and Yan, H. (1993). Map image segmentation based on thresholding and fuzzy rules, *Elect. Lett.*, 29(21), 1841-1843.

Chi, Z. and Yan, H. (1995). Handwritten numeral recognition using a small number of fuzzy rules with optimized defuzzification parameters, *Neural Networks*, 8(5), 821-827.

Chi, Z. and Yan, H. (1996). ID3-derived fuzzy rules and optimized defuzzification for handwritten numeral recognition, *IEEE Trans. Fuzzy Syst.*, 4(1), 1996, 24-31.

Chi, Z., Suters, M. and Yan, H. (1996b). Handwritten digit recognition using combined ID3-derived fuzzy rules and Markov chains, *Patt. Recog.*, 29(11), 1821-1833.

Chi, Z., Wu, J. and Yan, H. (1995). Handwritten numeral recognition using self-organizing maps and fuzzy rules, *Patt. Recog.*, 28(1), 59-66.

Chi, Z., Yan, H. and Pham, T. (1996a). *Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition*, World Scientific, Singapore.

Chiang, J. and Gader, P. (1997). Hybrid fuzzy-neural systems in handwritten word recognition. *IEEE Trans. Fuzzy Syst.*, 5(4), 497-510.

Chien, Y. T. (1978). *Interactive pattern recognition*, Marcel Dekker, Monticello, NY.

Chiu, S. L. and Cheng, J. J. (1994). Automatic rule generation of fuzzy rule base for robot arm posture selection, *Proc. NAFIPS Conf.*, San Antonio, Texas, 436-440.

Chiu, S. L. (1994). Fuzzy model identification based on cluster estimation, *J. Intell. and Fuzzy Syst.*, 2, 267 - 278.

Chiu, S. L. (1995). Extracting fuzzy rules for pattern classification by cluster estimation, *Proc. IFSA Congresss*, 1- 4.

Chiu, S. L. (1997). Extracting fuzzy rules from data for function approximation and pattern classification, *Fuzzy Information Engineering*, eds. D. Dubois, H. Prade and R.R. Yager, Wiley and Sons, NY, 149-162.

Cho, S.-B. (1995). Fuzzy aggregation of modular neural networks with ordered weighted averaging operators, *Int. J. of Approximate Reasoning*, 13, 359-375.

Cho, S.-B. and Kim, J. (1995). Combining multiple neural networks by fuzzy integral for robust classification. *IEEE Trans. Syst., Man and Cyberns.*, 25(2), 380 - 384.

Choi J. J., Arabshahi R. J., Marks, R. J. and Caudell T. P. (1992). Fuzzy parameter adaptation in neural systems, *IEEE Int. Joint Conf. on Neural Networks*, 1, 232-238.

Choi, Y. and Krishnapuram, R. (1995). Image enhancement based on fuzzy logic, *Proc. IEEE Int. Conf. on Image Processing*, IEEE Press, Piscataway, NJ, 167-170.

Choi, Y. and Krishnapuram, R. (1996). Fuzzy and robust formulations of maximum-likelihood-based Gaussian mixture decomposition, *Proc. of the IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1899-1905.

Choi, Y. and Krishnapuram, R. (1997). A robust approach to image enhancement based on fuzzy logic, *IEEE Trans. Image Processing*, 6, 808-825.

Chomsky, N. (1965), *Aspects of the Theory of Syntax*, M.I.T. Press, Cambridge, MA.

Chung, F. L. and Lee, T. (1994). A fuzzy learning model for membership function estimation and pattern classification, *Proc. EEE Int. Conf. on Fuzzy Syst.*, 1, IEEE Press, Piscataway, NJ, 426-431.

Cios, K. and Liu, N. (1992). A machine learning algorithm for generation of a neural network architecture: a continuous ID3 algorithm, *IEEE Trans. Neural Networks*, 3(2), 280-291.

Cios, K. and Sztandera, L. M. (1992). Continuous ID3 algorithm with fuzzy entropy measures, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ 469-476.

Clark, M., Hall, L. O., Goldgof, D, Clarke, L., Velthuizen, R. and Silbiger, M. (1994). MRI segmentation using fuzzy clustering techniques: integrating knowledge, *IEEE Engineering in Medicine and Biology Magazine*, 13(5), 730-742.

Clark, M., Hall, L. O., Goldgof, D, Velthuizen, R., Murtaugh, F.R. and Silbiger, M. (1998). Automatic tumor segmentation using knowledge-based techniques, *IEEE Trans. Med. Imaging*, 17(2), 187-201.

Coray, C. (1981). Clustering algorithms with prototype selection, *Proc. of Hawaii Int. Conf. on Syst. Sci.*, II, Western Periodicals Co., 945-955.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990). *Introduction to Algorithms*, McGraw Hill, NY.

Cottrell, G.W., Munroe, P. and Zipser, D. (1989). Image compression by back-propagation; An example of extensional programming, in *Models of Cognition: A Review of Cognitive Science*, ed. N. Sharkey, Norwood, NJ, 208-240.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification, *IEEE Trans. Inf. Theory*, 13, 21-27.

CubiCalc, (1990). *Comp. Software Manual*, HyperLogic Corporation, IBM-PC.

DARPA Neural Network Study, (1988). AFCEA Press, Fairfax, VA.

Das Gupta, S. and H. E. Lin (1980). Nearest neighbor rules of statistical classification based on ranks, *Sankhya A*, 42, 419-430.

Dasarathy, B.V. (1990). *Nearest Neighbor (NN) Norms : NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, CA.

Dasarathy, B.V. (1994a). Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design, *IEEE Trans. Syst., Man and Cyberns.*, 24(3), 511-517.

Dasarathy, B.V. (1994b). *Decision Fusion*, IEEE Computer Society Press, Los Alamitos, CA.

Davé, R. N. (1989). Use of the adaptive fuzzy clustering algorithm to detect lines in digital images, *Proc. SPIE Conf. on Intell. Robots and Comp. Vision*, 1192 (2), 600-611.

Davé, R. N. (1990a). An adaptive fuzzy c-elliptotype clustering algorithm, *Proc. NAFIPS Conf.* 9-12.

Davé, R. N. (1990b). Fuzzy shell-clustering and applications to circle detection in digital images, *Int. J. of Gen. Syst.*, 16(4), 343-355.

Davé, R. N. (1991a). Characterization and detection of noise in clustering, *Patt. Recog. Lett.*,12 (11) 657-664.

Davé, R. N. (1991b). New measures for evaluating fuzzy partitions induced through c-shells clustering, *Proc. SPIE Conf. on Intell. Robot and Comp. Vision X*, 1607, 406-414.

Davé, R. N. (1992). Generalized fuzzy c-shells clustering and detection of circular and elliptical boundaries, *Patt. Recog.*, 25(7), 713-721.

Davé, R. N. (1996). Validating fuzzy partitions obtained through c-shells clustering, *Patt. Recog. Lett.*, 17, 613-623.

Davé, R. N. and Bhamidipati, S. (1989). Application of the fuzzy-shell clustering algorithm to recognize circular shapes in digital images, *Proc. Third IFSA Congress*, 238-241.

Davé, R. N. and Bhaswan, K. (1991a). Adaptive fuzzy c-shells clustering,*Proc. NAFIPS Conf.*, 195-199.

Davé, R. N. and Bhaswan, K. (1991b). New measures for evaluating fuzzy partitions induced through c-shells clustering, *Proc. SPIE Intell. Robots and Comp. Vision X: Algorithms and Techniques*, 1607, 406-414.

Davé, R. N. and Bhaswan, K. (1992). Adaptive fuzzy c-shells clustering and detection of ellipses, *IEEE Trans. Neural Networks*, 3(5), 643-662.

Davé, R. N. and Fu, T. (1994). Robust shape detection using fuzzy clustering: practical applications, *Fuzzy Sets and Syst.*, 65(2/3), 161-185.

Davé, R. N. and Krishnapuram, R. (1997). Robust clustering methods: a unified view, *IEEE Trans. Fuzzy Syst.*, 5(2), 270-293.

Davé, R. N. and Patel, K. J. (1990). Progressive fuzzy clustering algorithms for characteristic shape recognition, *Proc. NAFIPS Conf.*, 121-125.

Davé, R. N. and Sen, S. (1998). Generalized noise clustering as a robust fuzzy c-means estimator model, *Proc. NAFIPS Conf.*, eds. J. C. Bezdek and L. O. Hall, 256-260.

Davenport, J. W., Bezdek, J. C. and Hathaway, R. (1988). Parameter estimation for finite mixture distributions, *Int. J. Comp. and Math. with Applications*, 15(10), 819-828.

Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure, *IEEE Trans. Patt. Anal. and Machine Intell.*, 1(4), 224-227.

De Gruijter, J.J. and McBratney, A.B. (1988). A modified fuzzy k-means method for predictive classification, in *Classification and Related Methods of Data* Analysis, ed. H. H. Bock, Elsevier, North Holland.

De Veaux, R. D. (1989). Mixtures of linear regressions, *Comp. Stat. and Data Anal.*, 8, 227-245.

De, R. K., Pal, N. R. and Pal, S. K. (1997). Feature analysis : neural network and fuzzy set theoretic approaches, *Patt. Recog.*, 30(10), 1579-1590.

Del Bimbo, A. and Vicario, E. (1998). Using weighted spatial relationships in retrieval by visual contents, *Proc. IEEE Workshop on Content-Based Access of Image and Video Libraries*, IEEE Computer Society Press, Los Alamitos, CA, 35-39.

Delgado, M., Gomez-Skarmeta, A. F. and Martin, F. (1997). A fuzzy clustering based rapid-prototyping for fuzzy rule-based modeling, *IEEE Trans. Fuzzy Syst.*, 5(2), 223-233.

Delgado, M., Gomez-Skarmeta, A. F. and Vila, M.A. (1995). Hiearchical clustering to validate fuzzy clustering, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, 1807-1812.

Dellepiane, S. (1997). The active role of 2-D and 3-D images: semi-automatic segmentation, *Contemporary Perspectives in Three-Dimensional Biomedical Imaging*, eds. C. Roux and J. L. Coatrieux, IOS Press, 165-189.

Dellepiane, S., Fontana, F. and Vernazza, G. L. (1996). Nonlinear image labeling for multivalued segmentation, *IEEE Trans. Image Processing*, 5(3), 429-446.

Deluca, A. and Termini, S. (1972). A definition of nonprobabilistic entropy in the setting of fuzzy sets theory, *Inf. and Control*, 20, 301-312.

Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm, *J. of the Royal Statistical Society*, B(39), 1-38.

Denoeux, T. (1995). A k-nearest neighbor classification rule based on Dempster-Shafer theory, *IEEE Trans. Syst., Man and Cyberns.*, 25, 804-813.

DePalma, G.F. and Yau, S. S. (1975). Fractionally fuzzy grammars with application to pattern recognition, in *Fuzzy Sets and their Applications to Cognitive and Decision Processes*, eds. L. A. Zadeh, K.S. Fu, K. Tanaka and M. Shimura, Academic Press, NY, 329-351.

Devijver, P. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*, Prentice-Hall, Englewood Cliffs, NJ.

Di Gesu, V., Maccarone, M.C. and Tripiciano, M. (1991). MMFuzzy: Mathematical morphology based on fuzzy operators, *Proc. IFSA Congress*, 29-32.

Di Gesu, V. and Maccarone, M.C. (1986). Feature selection and possibility theory, *Patt. Recog.*, 19, 63-72.

Dickerson, J. and Kosko, B. (1993). Fuzzy function learning with covariance ellipsoids, *Proc. IEEE Int. Conf. on Neural Networks*, IEEE Press, Piscataway, NJ, 1162-1167.

Diday, E. (1971). La methode des nuees dynamiques, *Rev. Stat. Appliquee*, XIX(2), 19-34.

Diday, E. (1975). Classification automatique sequentialle pour la grands tableaux, *Rev. Francaise Automat. Inform. Recherche Operationnelle*, 29-61.

Diday, E. and Simon, J.C. (1976). Cluster Analysis, in *Digital Pattern Recognition*, ed. K.S.Fu, Springer Verlag, 47-94.

Diday, E., Schroeder, A. and Ok, Y. (1974), The dynamic clusters method in pattern recognition, *Proc. Int. Federation for Inf. Processing Congress*, American Elsevier, North Holland, NY, 691-709.

Dong, W., Shah, H., and Wong, F. (1985). Fuzzy computations in risk and decision analysis, *Civil Engineering Syst.*, 2, 201-208.

Driankov, D., Hellendorn, H. and Reinfrank, M. (1993). *An Introduction to Fuzzy Control*, Springer Verlag, NY.

Drucker, H., Schapire, R., and Simard, P. (1993). Improving performance of neural networks using a boosting algorithm. *Advances in Neural Inf. Processing Syst.*, 5, Morgan Kaufmann, San Mateo, CA, 42-49.

Dubois, D. and Prade, H. (1980). *Fuzzy sets and systems: theory and applications*, Academic Press, NY.

Dubois, D. and Prade, H. (1982). A class of fuzzy measures based on triangular norms, *Int. J. Gen. Syst.*, 8 43-61.

Dubois, D. and Prade, H. (1985). A Review of fuzzy set aggregation connectives, *Inf, Sciences*, 36(1/2), 85-121.

Dubois, D. and Jaulent, M. C. (1987). A general approach to parameter evaluation in fuzzy digital pictures, *Patt. Recog. Lett.*, 6, 251-259.

Dubois, D., Nguyen, H.T., Prade, H. and Sugeno, M. (1998). Introduction: the real contribution of fuzzy systems, *Fuzzy Systems: Modeling and Control*, eds. H. T. Nguyen and M. Sugeno, Kluwer, Boston, MA, 8-10.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*, Wiley Interscience, NY.

Dudani, S. A. (1976). The distance weighted k-nearest neighbor rule, *IEEE Trans. Syst., Man and Cyberns.*, 6, 325-327.

Dunn, J. C. (1974a). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters, *J. Cyberns.*, 3(3), 32-57.

Dunn, J. C. (1974b). A graph theoretic analysis of pattern classification via Tamura's fuzzy relation, *IEEE Trans. Syst., Man and Cyberns.*, 4, 310-313.

Dunn, J. C. (1977). Indices of partition fuzziness and the detection of clusters in large data sets, in *Fuzzy Automata and Decision Processes*, ed. M.M. Gupta, Elsevier, NY.

Dutta, S. (1991). Approximate spatial reasoning: integrating qualitative and quantitative constraints *Int. J. of Approximate Reasoning*, 5, 307-331.

Dyckhoff, H. and Pedrycz, W. (1984). Generalized means as a model of compensation connectives, *Fuzzy Sets and Syst.*, 14, 143-154.

Errington, P. and Graham, J. (1993). Application of artificial neural networks to chromosome classification, *Cytometry*, 14, 627-639.

Everitt, B. S. (1978). *Graphical Techniques for Multivariate Data*, North Holland, NY.

Everitt, B. S. and Hand, D. J. (1981). *Finite Mixture Distributions*, Chapman and Hall, NY.

Fahlman, S. E. and Lebiere, C. (1990). The cascade correlation learning architecture, in *Advances in Neural Inf. Processing*, ed. D. Touretzky, Morgan Kauffman, San Mateo, CA, 524-532.

Fayyad, U. M. and Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation, *Machine Learning*, 8, 87-102.

Fiesler, E. and Beale, R., eds., (1997). *Handbook of Neural Computation*, Institute of Physics Publ., Bristol, UK.

Figue, J., Grabisch, M. and Charbonnel, M. P. (1998). A method for still image interpretation relying on a multi-algorithm fusion scheme, application to human face characterization, *Fuzzy Sets and Syst.,* 103(2).

Fisher, P. F. and Pathirana, S. (1990). The evaluation of fuzzy membership of land cover classes in the suburban zone, *Remote Sensing of the Environment,* 34, 121-132.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems, *Ann. Eugenics,* 7(2), 179-188.

Forgy, E. (1965). Cluster analysis of multivariate data : efficiency vs interpretability of classifications, *Biometrics,* 21(3).

Fowlkes, E. B. and Mallows, C. L. (1983). A method of comparing two hierarchical clusterings, *J. Amer. Stat. Assoc.,* 78, 553-569.

Freeman, J. (1975). The modeling of spatial relations, *Computer Graphics and Image Processing,* 4, 156-171.

Frigui, H. and Krishnapuram, R. (1995). A robust clustering algorithm based on the M-estimator, *Proc. Int. Conf. on Neural, Parallel and Scientific Computations,* 1, 163-166.

Frigui, H. and Krishnapuram, R. (1996a). A comparison of fuzzy shell clustering methods for the detection of ellipses, *IEEE Trans. Fuzzy Syst.,* 4, 193-199.

Frigui, H. and Krishnapuram, R. (1996b). A robust clustering algorithm based on competitive agglomeration and soft rejection of outliers, *Proc. IEEE Conf. Comp. Vision and Patt. Recog.,* IEEE Press, Piscataway, NJ, 550-555.

Frigui, H. and Krishnapuram, R. (1996c). A robust algorithm for automatic extraction of an unknown number of clusters for noisy data, *Patt. Recog. Lett.,* 17(12), 1223-1232.

Frigui, H. and Krishnapuram, R. (1997). Clustering by competitive agglomeration, *Patt. Recog.,* 30(7), 1109-1119.

Frigui, H., Gader, P. and Keller, J. M. (1998a). Fuzzy clustering for landmine detection, *Proc. NAFIPS Conf.,* eds. J. C. Bezdek and L. O. Hall, 261-265.

Frigui, H., Krishnapuram, R., DeKruger, D., Keller, J. and Gader, P. (1998b). Robust and fuzzy preprocessing algorithms for target detection in LADAR images, *Proc. IEEE Int. Conf. on Fuzzy Systems,* IEEE Press, Piscataway, NJ, 1554-1559.

Fu, K.S. (1982). *Syntactic Pattern Recognition and Applications*, Prentice Hall, Englewood Cliffs, NJ.

Fukunaga, K. (1991). *Statistical Pattern Recognition*, second edition (first edition, 1972), Academic Press, San Diego, CA.

Fukuyama, Y. and Sugeno, M. (1989). A new method of choosing the number of clusters for the fuzzy c-means method, *Proc. 5-th Fuzzy Syst. Symp.* (in Japanese), 247-256.

Furukawa, M. and Yamakawa, T. (1998). A fuzzy neural network for pattern recognition using local feature of pattern, *Proc. Int. Conference on Soft Computing*, eds. T. Yamakawa and G. Matsumoto, Iizuka, Japan, World Scientific, Singapore, 1000-1003.

Gader, P. (1997). Fuzzy spatial relations based on fuzzy morphology, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1179-1183.

Gader, P. and Mohamed, M. (1995). Multiple classifier fusion for handwritten word recognition, *Proc. IEEE Int. Conf. Syst., Man, Cyberns.*, IEEE Press, Piscataway, NJ, 2329-2335.

Gader P. and Mohamed, M. (1996). The Choquet fuzzy integral in handwritten word recognition, *Proc. SPIE Conf. on Document Recognition*, San Jose, 309-321.

Gader, P., Keller, J. M. and Cai, J. (1995a). A fuzzy logic system for the detection and recognition of street number fields on handwritten postal addresses, *IEEE Trans. Fuzzy Syst.*, 3(1), 83-95.

Gader, P., Keller, J. M. and Liu, H. (1998b). Landmine detection using fuzzy clustering in DARPA backgrounds data collected with the Geo-Centers ground penetrating radar, *Proc. SPIE Detection and Remediation Technologies for Mines and Minelike Targets III*, 3392, SPIE, Bellingham, WA, 1139-1149.

Gader, P., Keller, J. M., Frigui, H., Liu, H. and Wang, D. (1998a). Landmine detection using fuzzy sets with GPR images, *Proc. 1998 IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 232-236.

Gader, P., Keller, J. M., Krishnapuram, R., Chiang, J., and Mohamed, M. (1997b). Neural and fuzzy methods in handwriting recognition, *IEEE Computer*, 30(2) 79-86.

Gader, P., Mohamed, M. and Chiang, J-H. (1992). Fuzzy and crisp handwritten alphabetic character recognition using neural networks, *Proc. Artificial Neural Networks in Engineering*, St. Louis, MO, 421-427.

Gader, P., Mohamed, M. and Chiang, J-H. (1995b). Comparison of crisp and fuzzy character neural networks in handwritten word recognition, *IEEE Trans. Fuzzy Syst.*, 3(3), 357-363.

Gader, P., Mohamed, M. and Chiang, J-H. (1997a). Handwritten word recognition with character and inter-character neural networks. *IEEE Trans. Syst., Man and Cyberns.*, 27(1), 158-165.

Gader, P., Mohamed, M. and Keller, J. M. (1995c). Applications of fuzzy integrals to handwriting recognition, *Proc. SPIE Applications of Fuzzy Logic Technology II*, SPIE, Bellingham, WA, 102-113.

Gader, P., Mohamed, M. and Keller, J. M. (1996a). Dynamic-programming-based handwritten word recognition using the Choquet integral as the match function, *J. of Electronic Imaging*, 5(1), 15-24.

Gader, P., Mohamed, M. and Keller, J.M. (1996b). Fusion of handwritten word classifiers, *Patt. Recog. Lett.*, 17(6), 577-584.

Gader, P., Whalen, M., Ganzberger, M, and Hepp, D. (1995d). Handprinted word recognition on a NIST data set. *Machine Vision and Its Applications*, 8, 31-40.

Gasós, J. and Ralescu, A. L. (1995). Towards a linguistic instruction based navigation support system - Using environment information for guiding scene interpretation, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1261-1266.

Gath, I. and Geva, A.B. (1989a). Unsupervised optimal fuzzy clustering, *IEEE Trans. Patt. Anal. and Machine Intell.*, 11, 773-781.

Gath, I. and Geva, A.B. (1989b). Fuzzy clustering for the estimation of the parameters of mixtures of normal distributions, *Patt. Recog. Lett.*, 7, 773-781.

Geci, D. (1996). Cell detection and classification using the fuzzy Hough transform, MsCS Thesis, U. of W. Florida, Pensacola, FL.

Genther, H. and Glesner, M. (1994). Automatic generation of a fuzzy classification system using fuzzy clustering methods, *Proc. ACM Symp. on Applied Computing*, ACM Press, NY, 180-183.

Gersho, A. and Gray, R. (1992). *Vector Quantization and Signal Compression*, Kluwer, Boston.

Geva, A. B. and Pratt, H. (1994). Unsupervised clustering of evoked potentials by waveform, *Med. and Bio. Engineering and Computing*, 543-550.

Geyer-Schulz, A. (1998). Fuzzy genetic algorithms, in *Fuzzy sets in Decision Analysis: Operations Research and Statistics*, ed. R. Slowinski, Kluwer, Boston, MA, 403-460.

Ghosh, A., Pal N. R., and Pal S. K. (1993). Self-organization for object extraction using multilayer neural network and fuzziness measures, *IEEE Trans. Fuzzy Systems*, 1, 54-68.

Giarratano, J. and Riley, G. (1994). *Expert Syst. : Principles and Programming*, 2nd ed., PWS, Boston, MA.

Gillies, A., Hepp, D. and Gader, P. (1992). A system for recognizing handwritten words, *ERIM Technical Report submitted to U. S. Postal Service*,.

Gillies, A., Hepp, D., Ganzberger, M.. Rovner, R. and Gader, P. (1993). Handwritten address interpretation, *ERIM Technical Report submitted to U. S. Postal Service*,.

Gitman, I. and Levine, M.D. (1970). An algorithm for detecting unimodal fuzzy sets and its application as a clustering techniques, *IEEE Trans. Computers*, 19, 583-593.

Godjevac, J. (1997). *Neuro-Fuzzy Controllers*, Presses Polytechniques et Univeritaires Romandes, Lausanne.

Goetcherian, V. (1980). From binary to gray tone image processing using fuzzy logic concepts, *Patt. Recog.*, 12(1), 7-15.

Goldberg, D. E. (1989). *Genetic algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Gonzalez, R. C. and Thomason, M. G. (1978). *Syntactic Pattern Recognition : An Introduction*, Addison Wesley, Reading, MA.

Gonzalez, R. C. and Woods, R. E. (1992). *Digital Image Processing*, Addison Wesley, Reading, MA.

Goos, G., Hartmanis, J. and van Leeuwen, J., eds, (1996). *Advances in Structural and Syntactic Pattern Recognition*, Lecture Notes in Comp. Science, 1121, Springer, Berlin.

Gopal, S. and Woodcock, C. (1994). Theory and methods for accuracy estimation of thematic maps using fuzzy sets, *Photogrammetric Engineering and Remote Sensing*, 60, 181-188.

Grabisch, M. (1994). Fuzzy integrals as a generalized class of order filters, *Proc. SPIE*, 2315, SPIE, Bellingham, WA, 128-136.

Grabisch, M. and Nicolas, J-M. (1994). Classification by fuzzy integral: Performance and tests. *Fuzzy Sets and Systems*, 65, 255-271.

Grabisch, M. and Schmitt, M. (1995). Mathematical morphology, order filters, and fuzzy logic, *Proc. IEEE/IFES Int. Joint. Conf. on Fuzzy Syst.*, Yokohama Japan, 2103-2108.

Grabisch, M., Figue, J. and Charbonnel, M. P. (1996). Analysis and modeling of face images, *Proc. IIZUKA'96*, 2, eds. T. Yamakawa and G. Matsumoto, World Scientific, Singapore, 761-764.

Grabisch, M., Murofushi, T, and Sugeno M. (1992). Fuzzy measure of fuzzy events defined by fuzzy integrals, *Fuzzy Sets and Syst.*, 50, 293-313.

Grabisch, M., Murofushi, T. and Sugeno, M., eds., (1999). *Fuzzy measures and integrals - theory and applications*, Springer-Verlag, Heidelberg.

Grabisch, M., Nguyen, H. and Walker E. (1995). *Fundamentals of Uncertainty Calculi with Applications to Fuzzy Inference.* Kluwer, Dordrecht.

Graham, J. and Piper, J. (1994). Automatic karyotype analysis, in *Methods in Molecular Biology*, 29, ed. J. R. Gosden, Humana Press, Totowa, NJ, 141 - 185.

Grossberg, S. (1976a). Adaptive pattern classification and universal recoding I : parallel development and coding of neural feature detectors, *Biological Cyberns.*, 23, 121-134.

Grossberg, S. (1976b). Adaptive pattern classification and universal recoding II : feedback, expectation, olfaction, and illusions, *Biological Cyberns.*, 23, 187-202.

Grossberg, S. (1978), Do all neural networks look alike? A comment on Anderson, Silverstein, Ritz, and Jones, *Psychological review*, 85, 592-596.

Grossberg, S. (1987). Competitive learning : from interactive activation to adaptive resonance, *Cognitive Science*, 11, 23-63.

Gunderson, R. (1978). Applications of fuzzy ISODATA algorithms to star-tracker printing systems, *Proc. Triannual World IFAC Congress*, 1319-1323.

Gunderson, R. (1983). An adaptive FCV clustering algorithm, *Int. J. Man-Machine Studies*, 19, 97-104.

Gunderson, R. and Thrane, K. (1985). Monitoring polycyclic aromatic hydrocarbons : An environmental application of fuzzy c-varieties pattern recognition, in *Environmental Applications of Chemometrics*, eds. J. Breen and P. Robinson, ACS Symp. Series 292, ACS, Washington D.C, 130-147.

Gupta, M. M. and Rao D. H. (1994). On the principles of fuzzy neural networks, *Fuzzy Sets and Syst.*, 61, 1-18.

Gupta M. M. and Qi J. (1991). On fuzzy neuron models, *Proc. IEEE Int. Joint Conf. on Neural Networks*, II, 431-436.

Gustafson, E. E. and Kessel, W. (1979). Fuzzy clustering with a fuzzy covariance matrix, *Proc. IEEE Conf. on Decision and Control*, San Diego, IEEE Press, Piscataway, NJ, 761-766.

Halgamuge, S.K., Pochmuller, W. and Glesner, M. (1995). An alternative approach for generation of membership functions and fuzzy rules based on radial and cubic basis function networks, *Int. J. Approx. Reasoning*, 12(3/4), 279-298.

Hall, D. (1992). *Mathematical Techniques in Multisensor Data Fusion*, Artech House, Norwood, MA.

Hall, E. L. (1979). *Computer Image Processing and Recognition*, Academic Press, NY.

Hall, L. O. (1996). Learned fuzzy decision rules vs. decision trees in classifying microcalcifications in mammograms, *SPIE Proc. on Applications of Fuzzy Logic Technology III*, 2761, eds. B. Bosacchi and J. C. Bezdek, SPIE, Bellingham, WA, 54-61.

Hall, L. O. and Lande, P. (1998). The generation of fuzzy rules from decision trees, *J. Adv. Computational Intelligence*, 2(4), 128-133.

Hall, L. O., Bezdek, J. C., Boggavarapu, S. and Bensaid, A. (1994). Genetic algorithm guided clustering, *Proc. IEEE Int'l Conf. on Evolutionary Computation*, IEEE Press, Piscataway, 34-39.

Hall, L. O., Chawla, N. and Bowyer, K. (1998). Decision tree learning on very large data sets, *Proc. IEEE Conf. on Syst., Man and Cyberns.*, IEEE Press, Piscataway, NJ, 2579-2584.

Hall, L.O, Bensaid, A., Clarke, L., Velthuizen, R. Silbiger, M. and Bezdek, J. C. (1992). A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain, *IEEE Trans. Neural Networks*, 3(5), 672-682.

Hall, P. (1973). Equivalence between AND/OR graphs and context-free grammars, *Comm. of the ACM*, 16(7), 444-445.

Hampel, F. R. (1975). Beyond location parameters: robust concepts and methods, *Proc. Int. Stat. Institute*, 46, 375-382.

Han, J. H., Koczy, L.T. and Poston, T. (1993). The fuzzy Hough transform, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 803-808.

Hanson, A. R. and Riseman, E. M. (1978). Segmentation of natural scenes, in *Comp. Vision Syst.*, eds. A.R. Hanson and E.M. Riseman, Academic Press, NY, 129-144.

Haralick, R. M. and Shapiro, L. G. (1992). *Computer and Robot Vision*, Volume I, Addison Wesley, Reading, MA, Chapter 10.

Haralick, R.M., Shanmugam, K. and Dinstein, I. (1973). Textural features for image classification. *IEEE Trans. Syst. Man and Cyberns.*, 3(6), 610-621.

Hartigan, J. (1975). *Clustering Algorithms*, Wiley, NY.

Hashem, S. (1997). Optimal linear combinations of neural networks, *Neural Networks*, 10(4), 599-614.

Hata, Y., Hirano, S. and Kamiura, N. (1998). Medical image granulation by fuzzy inference, *Proc. NAFIPS Conf.* eds. J. C. Bezdek and L. O. Hall, 188-192.

Hata, Y., Kobashi, N., Kamiura, N. and Ishikawa, M. (1997). Fuzzy logic approach to 3D magnetic resonance image segmentation, in *Infor. Proc. in Medical Imaging*, Lecture notes in Comp. Science, 1230, 387-392.

Hathaway, R. J. and Bezdek, J. C. (1986a). Local convergence of the fuzzy c-means algorithms, *Patt. Recog.*, 19(6), 477-480.

Hathaway, R. J. and Bezdek, J. C. (1986b). On the asymptotic properties of fuzzy c-means cluster prototypes as estimators of mixture subpopulations, *Comm. Stat.*, 5(2), 505-513.

Hathaway, R. J. and Bezdek, J. C. (1988). Recent convergence results for the fuzzy c-means clustering algorithms, *J. Classification*, 5(2), 237-247.

Hathaway, R. J. and Bezdek, J. C. (1991). Grouped coordinate minimization using Newton's method for inexact minimization in one vector coordinate, *J. Optimization Theory and Applications*, 71(3), 503-516.

Hathaway, R. J. and Bezdek, J. C. (1993). Switching regression models and fuzzy clustering, *IEEE Trans. Fuzzy Syst.*, 1(3), 195-204.

Hathaway, R. J. and Bezdek, J. C. (1994a). Optimization of fuzzy clustering criteria using genetic algorithms, *Proc. IEEE Int'. Conf. on Evolutionary Computation*, IEEE Press, Piscataway, 589-594.

Hathaway, R. J. and Bezdek, J. C. (1994b). NERF c-Means : Non-Euclidean relational fuzzy clustering, *Patt. Recog.*, 27(3), 429-437.

Hathaway, R. J. and Bezdek, J. C. (1994c). An iterative procedure for minimizing a generalized sum-of-squared errors clustering criterion, *Neural, Parallel and Scientific Computations*, 2, 1-16.

Hathaway, R. J. and Bezdek, J. C. (1995). Optimization of clustering criteria by reformulation, *IEEE Trans. Fuzzy Syst.*, 3(2), 241-246.

Hathaway, R. J., Bezdek, J. C. and Pedrycz, W. P. (1996). A parametric model for fusing heterogeneous fuzzy data, *IEEE Trans. Fuzzy Syst.*, 4(3), 270-281.

Hathaway, R. J., Davenport, J. W. and Bezdek, J. C, (1989). Relational duals of the c-means clustering algorithms, *Patt. Recog.*, 22 (2), 205-212.

Hayashi, I., Naito, E. and Wakami, N. (1991). A proposal of fuzzy connective with learning function and its application to fuzzy information retrieval, *Proc. Int. Fuzzy Engineering Symp.*, Yokohama, Japan, 446-455.

Haykin, S. (1994). *Neural Networks : A Comprehensive Foundation*, MacMillan, NY.

Heath, M. D., Sarkar, S., Sanocki, T. and Bowyer, K. W. (1997). A robust visual method for assessing the relative performance of edge-detection algorithms, *IEEE Trans. Patt. Anal. and Machine Intell.*, 19(12), 1338-1359.

Hecht-Nielsen, R. (1988). Neurocomputing: picking the human brain, *IEEE Spectrum*, March, 36-41.

Herndon, R. C., Lancaster, J. L., Toga, A. W. and Fox, P. T. (1996). Quantification of white and gray matter volumes from T1 parametric images using fuzzy classifiers, *J. Mag. Res. Imaging*, 6(3), 425-435.

Hershfinkel, D. and Dinstein, I. (1996). Accelerated fuzzy c-means clustering algorithm, in *Proc. SPIE Applications of Fuzzy Logic Technology III*, 2761, eds. B. Bosachi and J. C. Bezdek, 41-52.

Higashi, M. and Klir, G. J. (1983). Measures of uncertainty and information based on possibility distributions, *Int. J. Gen. Syst.*, 9, 43-58.

Hirota, K. and Iwama, K. (1988). Application of modified FCM with additional data to area division of images, *Inf. Sci.*, 45(2), 213-230.

Ho, T. K., Hull, J. J., and Srihari, S. N. (1994). Decision combination in multiple classifier systems. *IEEE Trans. Patt. Anal. and Machine Intell.*, 16(1), 66 - 75.

Hocaoglu A. and Gader, P. (1998). Choquet integral representations of nonlinear filters with applications to LADAR image processing. *Proc. SPIE Conf. Nonlinear Image Processing IX*, SPIE, Bellingham, WA, 66-72.

Hocaoglu, A. K., Gader, P. and Keller, J. (1997). Nonlinear filters for target detection in LADAR range images, *Proc. NAFIPS Conf.*, 177-182.

Hoeppner, F. (1997). Fuzzy shell clustering algorithms in image processing - fuzzy c-rectangular and 2-rectangular shells. *IEEE Trans. Fuzzy Syst.*, 5(4), 599-613.

Hoffman, R. and Jain, A. K. (1987). Segmentation and classification of range images, *IEEE Trans. Patt. Anal. and Machine Intell.*, 9, 608-620.

Hosmer, D. W. (1974). Maximum likelihood estimates of the parameters of a mixture of two regression lines, *Comm. in Stat.*, 3(10), 995-1005.

Hough, P.V.C. (1962). Methods and means for recognizing complex patterns, *U.S. Patent* 3069654.

House, J. M., Lee, Y. W. and Shin, D. R. (1999). Classification techniques for fault detection and diagnosis of an air handling unit, *Trans. ASHRAE*, 105(1).

Huang, T. and Suen, C. (1995). Combination of multiple experts for recognition of unconstrained handwritten numerals. *IEEE Trans. Patt. Anal. and Machine Intell.*, 17(1), 90-94.

Huber, P.J. (1981).*Robust Statistics*, John Wiley and Sons, NY.

Hubert, L. J. and Arabie, P. (1985). Comparing partitions, *J. Classification*, 2, 193-218, 1985.

Hughes, G. F. (1968). On the mean accuracy of statistical pattern recognizers, *IEEE Trans. Inf. Theory*, 14, 55-63.

Hull, J. (1994). A database for handwritten text recognition research, *IEEE Trans. Patt. Anal. and Machine Intell.*, 16(5), 550 - 554.

Huntsberger, T. and Ajjimarangsee, P. (1990). Parallel self-organizing feature maps for unsupervised pattern recognition, *Int. J. Gen. Syst.*, 16, 357-372.

Hwang, S. Y., Lee, H. S. and Lee, J. J. (1998). General fuzzy acceptors for pattern recognition, *Fuzzy Sets and Syst.*, 80, 397-401.

Illingworth, J. and Kittler, J. (1987). The adaptive Hough transform, *IEEE Trans. Patt. Anal. and Machine Intell.*, 9(5), 690-698.

Ismail, M. A. and Selim, S. A. (1986). On the local optimality of the fuzzy ISODATA clustering algorithm, *IEEE Trans. Patt. Anal. and Machine Intell.*, 8(2), 284-288.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and G.E. Hinton. (1991). Adaptive mixtures of local experts, *Neural Computation*, 3, 79-87.

Jacobsen, T. and Gunderson, R. (1983). Trace element distribution in yeast and wort samples: An application of the FCV clustering algorithms, *Int. J. Man-Machine Studies*, 10(1), 5-16.

Jain, A. K. and Dubes, R. (1988). *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ.

Jain, A. K. and Flynn, P. J. (1996). Image segmentation by clustering, in *Advances in Image Understanding*, eds. K. Bowyer and N. Ahuja, IEEE Computer Society Press, Los Alamitos, CA., 65-83.

Jain, R., Kasturi, R. and Schunck, B. G. (1995). *Machine Vision*, McGraw-Hill, NY.

Jajuga, K. (1991). ℓ_1 norm-based fuzzy clustering, *Fuzzy Sets and Syst.*, 39(1), 43-50.

Jang, J.-S. R. (1994). Structure determination in fuzzy modeling: A fuzzy CART approach, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 480-485.

Jang, J.-S. R., Sun C.-T. and Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Upper Saddle River, NJ.

Janikow, C. Z. (1996a). Exemplar learning in fuzzy decision trees, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1500-1505.

Janikow, C. Z. (1996b). A genetic algorithm method for optimizing fuzzy decision trees, *Inf. Sci.*, 89(3-4), 275-296.

Janikow, C. Z. (1998). Fuzzy decision trees: issues and methods, *IEEE Trans. Syst., Man and Cyberns.*, B28(1), 1-14.

Johnson, J. L. (1994). Pulse-coupled neural nets: translation, rotation, scale, distortion and intensity signal invariances for images, *Applied Optics*, 33(26), 6239-6253.

Johnson, R. A. and Wichern, D. W. (1992). *Applied Multivariate Statistical Analysis*, 3rd ed., Prentice Hall, Englewood Cliffs, NJ.

Jolion, J. M., Meer, P. and Bataouche, S. (1991). Robust clustering with applications in computer vision, *IEEE Trans. Pattern Anal. and Machine Intell.*, 13(8),791-802.

Jordan, M. I. and Xu, L. (1995). Convergence results for the EM approach to mixtures of expert architectures, *Neural Networks*, 8(9), 1409-1431.

Jozwik, A. (1983). A learning scheme for a fuzzy k-NN rule, *Patt. Recog. Lett.*, 1, 287-289.

Kamel, M. S and Selim, S.Z. (1994). A new algorithm for solving the fuzzy clustering problem, *Patt. Recog.*, 27(3), 421-428.

Kandel, A. (1982). *Fuzzy Techniques in Pattern Recognition*, Wiley Interscience, NY.

Kandel, A. and Yelowitz, L. (1974). Fuzzy chains, *IEEE Trans. Syst., Man and Cyberns.*, (4), 472-475.

Kang, H.-B. and Walker, E. (1994). Characterizing and controlling approximation in hierarchical perceptual grouping, *Fuzzy Sets and Syst.*, 65, 187-224.

Karayiannis, N. B. (1996). Fuzzy and possibilistic clustering algorithms based on generalized reformulation, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1393-1399.

Karayiannis, N. B. (1997a). Learning vector quantization : a review, *Int. J. of Smart Engineering System Design,* 1(1), 33-58.

Karayiannis, N. B. (1997b). Entropy constrained learning vector quantization algorithms and their application in image compression, *Proc. SPIE Conf. on Applications of Artificial Neural Networks in Image Processing II,* 3030, SPIE, Bellingham, WA, 2-13.

Karayiannis, N. B. (1997c). A methodology for constructing fuzzy algorithms for learning vector quantization, *IEEE Trans. Neural Networks*, 8(3), 505-518..

Karayiannis, N. B. and Bezdek, J. C. (1997). An integrated approach to fuzzy learning vector quantization and fuzzy c-means clustering, *IEEE Trans. Fuzzy Syst.*, 5(4), 622-628.

Karayiannis, N. B. and Mi, W. (1998). Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques, *IEEE Trans. Neural Networks*, 8(6), 1492-1506.

Karayiannis, N. B. and Pai, P. I. (1996). Fuzzy algorithms for learning vector quantization, *IEEE Trans. Neural Networks*, 7(5), 1196-1211.

Karayiannis, N. B., Bezdek, J. C., Pal, N. R., Hathaway, R. J. and Pai, P.I. (1996). Repairs to GLVQ : A new family of competitive learning schemes, *IEEE Trans. Neural Networks*, 7(5), 1062-1071.

Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley Interscience, NY.

Kaufmann, A. (1975). *Introduction to the Theory of Fuzzy Subsets - Fundamental Theoretical Elements*, 1, Academic Press, NY.

Kawade, M. (1995). Object recognition system in a dynamic environment, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1285-1290.

Keller, J. M. and Chen, Z. (1992a). Learning in fuzzy neural networks utilizing additive hybrid operators, *Proc. Int. Conf. on Fuzzy Logic and Neural Networks,* Iizuka, Japan, 85-87.

Keller, J. M. and Chen, Z. (1992b). Detection of unimportant features during the training of a fuzzy neural network, *Proc. Artificial Neural Networks In Engineering*, St. Louis, MO, 357-362.

Keller, J. M. and Gader, P. (1997). Fuzzy logic and sensor fusion for humanitarian demining, oral presentation at the 1st US Army Multidisciplinary U. Research Initiative on Demining, Fort Belvoir, VA.

Keller, J. M. and Hobson, G. (1989). Uncertainty management in a rule-based automatic target recognizer, *Proc. SPIE Conference in Applications of Artificial Intelligence VII*, 126-137.

Keller, J. M. and Hunt, D. J. (1985). Incorporating fuzzy membership functions into the perceptron algorithm, *IEEE Trans. Pattern Anal. Machine Intell.*, 7(6), 693-699.

Keller, J. M. and Krishnapuram, R. (1994). Fuzzy decision models in computer vision, in*Fuzzy Sets, Neural Networks, and Soft Computing*, eds. R. Yager and L. Zadeh,Van Nostrand, 213-232.

Keller J. M. and Osborn, J. (1991). Temporal object identification via fuzzy models, *Proc. SPIE Symp. on Intell. Robots and Comp. Vision X*, SPIE, Bellingham, WA, 466-476.

Keller, J. M. and Osborn, J. (1996). Training the fuzzy integral, *Int. J. of Approx. Reasoning*, 15(1), 1-24.

Keller, J. M. and Qiu, H. (1988). Fuzzy sets methods in pattern recognition, in *Pattern Recognition, Lecture notes in Comp. Science*, 301, ed. J. Kittler, Springer-Verlag, 173-182.

Keller, J. M. and Sztandera, L. (1990). Spatial relationships among fuzzy subsets of an image, *Proc. Int. Symp. on Uncertainty Modeling and Anal.*, U. of Maryland, 207-211.

Keller, J. M. and Tahani, H. (1992a). Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks, *Int. J. Approx. Reasoning*, 6(2), 221-240.

Keller, J. M. and Tahani, H. (1992b). Backpropagation neural networks for fuzzy logic, *Inf. Sci.*, 62(3), 205-221.

Keller, J. M. and Wang, X. (1995). Comparison of spatial relation definitions in computer vision, *Proc. Joint ISUMA/NAFIPS Conf.*, 679-684.

Keller, J. M. and Wang, X. (1996). Learning spatial relationships in computer vision, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 118-124.

Keller J. and Yan, B. (1992). Possibility expectation and its decision making algorithm, *Proc. First IEEE Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 661-668.

Keller, J. M. and Yang, H. (1995). Fuzzy aggregation networks of hybrid neurons with generalized Yager operators, *Proc. IEEE Int. Conf. on Neural Networks*, IEEE Press, Piscataway, NJ, 2270-2274.

Keller J. M., Yager R. R. and Tahani H. (1992). Neural network implementation of fuzzy logic, *Fuzzy Sets and Syst.*, 45, 1-12.

Keller, J. M., Gader, P. and Caldwell, C.W., (1995b). The principle of least commitment in the analysis of chromosome images, *Proc. SPIE Symp. on OE/Aerospace Sensing and Dual Use Photonics*, SPIE, Bellingham, WA, 178-186.

Keller, J. M., Gader, P., Krishnapuram, R., Wang, X., Hocaoglu, A., Frigui, H. and Moore, J. (1998). Fuzzy logic automatic target recognition system for LADAR range images. *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 71-76.

Keller, J. M., Gader, P., Sjahputera, O., Caldwell, C.W. and Huang, H-M., (1995a). A fuzzy logic rule-based system for chromosome recognition, *Proc. IEEE Symp. on Computer Based Medical Syst.*, IEEE Press, Piscataway, NJ, 125-132.

Keller, J. M., Gader, P., Tahani, H., Chiang, J-H. and Mohamed, M. (1994a). Advances in fuzzy integration for pattern recognition, *Fuzzy Sets and Syst.*, 65 (2/3), 273-284.

Keller, J. M., Gray, M. and Givens, J. (1985). A fuzzy k-nearest neighbor algorithm, *IEEE Trans. Syst., Man and Cyberns.*, 15, 580-585.

Keller, J. M., Hayashi, Y. and Chen, Z., (1993). Interpretation of nodes in networks for fuzzy logic, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1203-1207.

Keller, J. M., Hayashi, Y., and Chen, Z. (1994c). Additive hybrid networks for fuzzy logic, *Fuzzy Sets and Syst.*, 66(3), 307-313.

Keller, J. M., Hobson, G., Wootton, J., Nafarieh, A., and Luetkemeyer, K. (1987). Fuzzy confidence measures in midlevel vision, *IEEE Trans. Syst., Man and Cyberns.*, 17(4), 676-683.

Keller, J. M., Krishnapuram, R. and Hobson, G. (1991). Information fusion via fuzzy logic in automatic target recognition, *Proc. Applied Imagery Patt. Recog. Workshop,*, 203-208.

Keller, J. M., Krishnapuram, R. and Rhee, F. (1992). Evidence aggregation networks for fuzzy logic inference, *IEEE Trans. Neural Networks*, 3(5), 761-769.

Keller, J. M., Krishnapuram, R., Chen, Z. and Nasraoui, O. (1994b). Fuzzy additive hybrid operators for network-based decision making, *Int. J. of Intell. Syst.*, 9(11), 1001-1024.

Keller, J. M., Krishnapuram, R., Gader, P. D. and Choi, Y.S. (1996). Fuzzy rule-based models in computer vision, *Fuzzy Modeling: Paradigms and Practice*, ed. W. Pedrycz, Kluwer, Norwell, MA, 353-374.

Keller, J. M., Qiu, H. and Tahani, J. (1986). The fuzzy integral in image segmentation, *Proc. NAFIPS Conf.*, 324-338.

Kendall. M and Gibbons, J. D. (1990). *Rank Correlation Methods*, Oxford U. Press, NY.

Kersten, P. R. (1995). The fuzzy median and the fuzzy MAD, *Proc. Joint ISUMA/NAFIPS Conf.*, 85-88.

Kiefer, N. M. (1978). Discrete parameter variation: efficient estimation of a switching regression model, *Econometrica*, 46(2), 427-434.

Kim, E., Park, M., Ji, S. and Park, M. (1997). A new approach to fuzzy modeling, *IEEE Trans. Fuzzy Syst.*, 5(3), 328-336.

Kim, J. (1997). A robust Hough transform based on validity, *Ph.D. Thesis*, U. of Missouri, Columbia, MO.

Kim, J. S. and Cho, H. S. (1994). A fuzzy logic and neural network approach to boundary detection for noisy imagery, *Fuzzy Sets and Syst.*, 65, 141-160.

Kim, J., Krishnapuram, R. and Davé, R. N. (1995) On Robustifying the c-Means Algorithms, *Proc. Joint ISUMA/NAFIPS Conf.*, 630-635.

Kim, J., Krishnapuram, R. and Davé, R. N. (1996). Application of the least trimmed squares technique to prototype-based clustering, *Patt. Recog. Lett.*, 17, 633-641.

Kim, T., Bezdek, J. C. and Hathaway, R. (1988). Optimality tests for fixed points of the fuzzy c-means algorithm, *Patt. Recog.*, 21(6), 651-663.

Kim, Y. S. and Mitra, S. (1994). An adaptive integrated fuzzy clustering model for pattern recognition, *Fuzzy Sets and Systs.*, 65, 297-310.

Kittler, J. (1998). Combining classifiers: A theoretical approach, *Pattern Anal. and Applications*, 1, 18-27.

Klawonn, F. and Kruse, R. (1997). Constructing a fuzzy controller from data, *Fuzzy Sets and Syst.*, 85(2), 177-193.

Kleiner, B. and Hartigan, J. A. (1981). Representing points in many dimensions by trees and castles, *J. Amer. Stat. Assoc.*,76, 260-269.

Klir, G. and Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic -Theory and Applications*, Prentice Hall, Englewood Cliffs, NJ.

Knuth, D. E. (1968). *Fundamental Algorithms*, Addison-Wesley, Reading, MA.

Kohonen, T. (1989). *Self-Organization and Associative Memory*, 3rd Edition, Springer-Verlag, Berlin.

Koskinen, L., Astola, J. and Neuvo, Y. (1991). Soft morphological filters, *SPIE Proc. Image Algebra and Morphological Image Processing*, 1568, SPIE, Bellingham, WA, 262-270.

Kosko, B. (1992). *Neural Networks and Fuzzy Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Kosko, B. (1997). *Fuzzy Engineering*, Prentice-Hall, Upper Saddle River, NJ.

Kreinovich, V. L., Mouzouris, G. C. and Nguyen, H. T. (1998). Fuzzy rule based modeling as a universal approximation tool, *Fuzzy Systems : Modeling and Control*, eds. H. T. Nguyen and M. Sugeno, Kluwer, Boston, MA, 135-196.

Krishnapuram, R. and Chen, L. (1993). Implementation of parallel thinning algorithms using recurrent neural networks, *IEEE Trans. Neural Networks*, 4, 142-147.

Krishnapuram, R. and Freg, C.-P. (1992). Fitting an unknown number of lines and planes to image data through compatible cluster merging, *Patt. Recog.*, 25(4), 385-400.

Krishnapuram, R. and Keller, J. M. (1993). A possibilistic approach to clustering, *IEEE Trans. Fuzzy Syst.*, 1(2), 98-110.

Krishnapuram, R. and Keller, J. M. (1996). The possibilistic c-means algorithm: insights and recommendations, *IEEE Trans. Fuzzy Syst.*, 4(3), 385-393.

Krishnapuram R. and Lee, J. (1988). Propagation of uncertainty in neural networks," *Proc. SPIE Conf. on Robotics and Computer Vision,*1002, SPIE, Bellingham, WA, 377-383.

Krishnapuram R. and Lee, J. (1989). Determining the structure of uncertainty management networks, *SPIE Proc. on Robotics and Computer Vision,* 1192, SPIE, Bellingham, WA, 592-597.

Krishnapuram, R. and Lee, J. (1992a). Fuzzy-connective-based hierarchical aggregation networks for decision making, *Fuzzy Sets and Syst.,* 46(1), 11-27.

Krishnapuram, R. and Lee, J. (1992b). Fuzzy-set-based hierarchical networks for information fusion in computer vision, *Neural Networks,* 5, 335-350.

Krishnapuram, R. and Medasani, S. (1995). Recovery of geometric properties of binary objects from blurred images, *Proc. IEEE Int. Conf. on Fuzzy Syst.,* IEEE Press Piscataway, NJ, 1793-1798.

Krishnapuram, R. and Rhee, F. C.-H. (1993a). Fuzzy rule generation methods for high-level computer vision, *Fuzzy Sets and Syst.,* 60, 245-258.

Krishnapuram, R. and Rhee, F. C.-H. (1993b). Compact fuzzy rule generation methods for computer vision, *Proc. IEEE Int. Conf. on Fuzzy Syst.,* 809-814.

Krishnapuram, R. Keller, J. M. and Ma, Y. (1993a). Quantitative analysis of properties and spatial relations of fuzzy image regions, *IEEE Trans. Fuzzy Syst.,* 1(3), 222-233.

Krishnapuram, R., Frigui, H. and Nasraoui, O. (1991). New fuzzy shell clustering algorithms for boundary detection and pattern recognition, *SPIE Proc. Robotics and Comp. Vision,,* 1607, SPIE, Bellingham, WA, 458-465.

Krishnapuram, R., Frigui, H. and Nasraoui, O. (1993b). Quadratic shell clustering algorithms and the detection of second degree curves, *Patt. Recog. Lett.,* 14, 545-552.

Krishnapuram, R., Frigui, H. and Nasraoui, O. (1995a). Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation, I, *IEEE Trans. Fuzzy Syst.,* 3, 29-43.

Krishnapuram, R., Frigui, H. and Nasraoui, O. (1995b). Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation, II, *IEEE Trans. Fuzzy Syst.,* 3, 44-60.

Krishnapuram, R., Nasraoui, O. and Frigui, H. (1992). The fuzzy c spherical shells algorithms: A new approach *IEEE Trans. Neural Networks*, 3, 663-671.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the travelling salesman problem, *Proc. AMS*, 7, 48-50.

Kumar, B. V. K. V. (1992). Tutorial survey of composite filter designs for optical correlators, *Applied Optics*, 31(23), 4773-4804.

Kuncheva, L. I. (1997). Application of OWA operators to the aggregation of multiple classification decisions, in*The Ordered Weighted Averaging Operators: Theory and Applications*, eds. R. R. Yager and J. Kacprzyk, Kluwer, Norwell, MA, 330-343.

Kuncheva, L. I. (1998). On combining multiple classifiers, *Proc. Int. Conf. Inf. Proc. and Management of Uncertainty*, 1890-1891.

Kuncheva, L. I. and Bezdek, J. C. (1998). Nearest prototype classification: clustering, genetic algorithms or random search?, *IEEE Trans. Syst. Man and Cyberns.*, C28(1), 160-164.

Kuncheva, L. I. and Bezdek, J. C. (1999). An integrated framework for generalized nearest prototype classifier design, *Int. J. Uncertainty, Fuzziness and Knowledge-Based Syst.*, 6(5), 437-457.

Kuncheva, L. I., Bezdek, J. C. and Duin, R. (1999). Decision templates for multiple classifier fusion: An experimental comparison, *Patt. Recog.*, in press.

Kuncheva, L. I., Bezdek, J. C. and Sutton, M. A. (1998). On combining multiple classifiers by fuzzy templates, *Proc. NAFIPS Conf.*, eds. J. C. Bezdek and L.O. Hall, 193-197.

Kuncheva, L. I., Kounchev, R.K. and Zlatev, R.Z. (1995). Aggregation of multiple classification decisions by fuzzy templates, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1470-1474.

Kundu, S. (1995). Defining the fuzzy spatial relationship LEFT(A,B), *Proc. IFSA Congress*, 1, 653-656.

Kwan, H. K. and Cai Y. (1994). A fuzzy neural network and its application to pattern recognition, *IEEE Trans. Fuzzy Syst.*, 2(3), 185-193.

Lachenbruch, P. A. (1975). *Discriminant Analysis*, Hafner Press, NY.

Larsen, H. L. and Yager, R. R. (1990). Efficient computation of transitive closures, *Fuzzy Sets and Syst.*, 38, 81-90.

Larsen, H. L. and Yager, R. R. (1993). The use of fuzzy relational thesauri for classificatory problem solving in information retrieval and expert systems, *IEEE Trans. Syst., Man and Cyberns.* 23, 31-41.

Law, T., Itoh, H. and Seki, H. (1996). Image filtering, edge detection, and edge tracing using fuzzy reasoning, *IEEE Trans. Pattern Anal. and Machine Intell.*, 18, 481-491.

Lee Y-G. and Hsueh Y-C. (1995). Fuzzy logic approach for removing noise, *Neurocomputing*, 9, 349-355.

Lee, E. T. (1972a). Fuzzy languages and their relation to automata, *Ph.D. Thesis*, U. of California, Berkeley, CA.

Lee, E. T. (1972b). Proximity measures for the classification of geometric figures, *J. Cyber.*, 2, 43-59.

Lee, E. T. (1975). Shape oriented chromosome classification, *IEEE Trans. Syst., Man and Cyberns.*, 6, 47-60.

Lee, E. T. (1976a). Algorithms for finding most dissimilar images with possible applications to chromosome classification, *Bull. Math. Bio.*, 38, 505-516.

Lee, E. T. (1976b). Shape oriented classification storage and retrieval of leukocytes, *Proc. IEEE Int. Joint Conf. on Patt. Recog.*, 8-11.

Lee, E. T. (1977a). Similarity measures and their relation to fuzzy query languages, *Pol. Anal. Inform. Sys.*, 1(1), 127-152.

Lee, E. T. (1977b). A similarity directed picture database, *Pol. Anal. Inform. Syst.*, 1(2), 113-125.

Lee, E. T. (1982). Fuzzy tree automata and syntactic pattern recognition, *IEEE Trans. Patt. Anal. and Machine Intell.*, 4(4), 445-449.

Lee, E. T. and Zadeh, L. A. (1969) Note on fuzzy languages, *Inform. Sci.*, 1, 421-434.

Lee, S. C. and Lee, E. T. (1970). Fuzzy neurons and automata, *Proc. Princeton Conf. on Infor. Science and Syst.*, 381-385.

Lee, S. C. and Lee, E. T. (1975). Fuzzy neural networks, *Mathematical Biosciences*, 23, 151 - 177.

Lee, S. W. (1992). Noisy Hangul character recognition with fuzzy tree classifier, Proc. SPIE, 1661, SPIE, Bellingham, WA, 127-136.

Leon, M., Gader, P., and Keller, J. (1996). Multiple neural network response variability as a predictor of neural network accuracy for chromosome recognition, *Proc. Rocky Mountain Bioengineering Symp.*, 32, 56-62.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals, *Sov. Phys. Dokl.*, 10(8), 707-710.

Li, C., Goldgof, D. and Hall, L. (1993). Knowledge-based classification and tissue labeling of MR images of human brains, *IEEE Trans. Med. Imaging*, 12(4), 740-750.

Li, H. and Yang, H, S. (1989). Fast and reliable image enhancement using fuzzy relaxation technique, *IEEE Trans. Syst., Man and Cyberns.*, 19(5), 1276-1281.

Libert, G. and Roubens, M. (1982). Nonmetric fuzzy clustering algorithms and their cluster validity, in *Approximate Reasoning in Decision Anal.*, eds. M. M. Gupta and E. Sanchez, 147-425.

Lim, W. Y. and Lee, S. U. (1990). On the color image segmentation algorithm based on the thresholding and fuzzy c-means techniques, *Patt. Recog.*, 23(9), 935-952, 1990.

Lin, C. T. and Lee, C. S. G. (1996). *Neural Fuzzy Systems*, Prentice Hall, Upper Saddle River, NJ.

Lin, Y. and Cunningham, G.A. (1995). A new approach to fuzzy-neural system modeling, *IEEE Trans. Fuzzy Syst.*, 3(2), 190-198.

Lippman, R. P. (1987). An introduction to computing with neural nets, *IEEE ASSP Magazine*, 61, 4-22.

Lippman, R. P. (1989). Pattern classification using neural networks, *IEEE Comm. Magazine*, 27, 47-64.

Lo, S.C. B., Lin, J.S., Li, H., Hasegawa, A., Tsujii, O., Freedman, M. T. and Mun, S. K. (1996). Detection of clustered microcalcifications using fuzzy classification modeling and convolution neural network, *Proc. SPIE on Medical Imaging: Image Processing*, 2710, SPIE, Bellingham, WA, 8-15.

Lowe, D. G. (1985). *Perceptual Organization and Visual Recognition*, Kluwer, Norwell, MA.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations,*Proc. Berkeley Symp. Math. Stat. and Prob.*, 1, eds. L. M. LeCam and J. Neyman, Univ. of California Press, Berkeley, 281-297.

Maher, P. E. and St. Clair, D. (1992). Uncertain reasoning in an ID3 machine learning framework, *Proc. IEEE Int. Conf. on Fuzzy Systems*, IEEE Press, Piscataway, NJ, 7-12.

Mamdani, E. H. and Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. of Man-Machine Studies*, 7(1),1-13.

Mammographic Image Anal. Society, (MIAS), Digital Mammogram Database. Published electronically (1994). Electronic contacts as of July, 1998 are mias@sv1.smb.man.ac.uk; http://s20c.smb.man.ac.uk/services/MIAS/MIAScom.html.

Man, Y. and Gath, I. (1994). Detection and separation of ring-shaped clusters using fuzzy clustering, *IEEE Trans. Patt. Anal. and Machine Intell.*,16(8), 855-861.

Mancuso, M., Poluzzi, R. and Rizzotto, G. A. (1994) Fuzzy filter for dynamic range reduction and contrast enhancement, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 264-267.

Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979) *Multivariate Analysis*, Academic Press, NY.

Marks, D. and Egenhofer, M. (1994). Modeling spatial relations between lines and regions: combining formal mathematical models and human subjects training, *Cartography and Geographic Inf. Syst.*, 21, 195-212.

Marks, R. (1993). Intelligence : Computational versus Artificial, *IEEE Trans. Neural Networks*, 4(5), 737-739.

Marr, D. (1982) *Vision*, W. H. Freeman, San Francisco, CA.

Martinetz, T., Bekovich, G. and Schulten, K. (1994). Topology representing networks, *Neural Networks*, 7(3), 507-522

Mazurov, V., Krivonogov, A., and Kasantsev, V. (1987). Solving of optimization and identification problems by the committee methods, *Patt. Recog. Lett.*, 20(4), 371-378.

McBratney, A.B. and Moore, A.W. (1985). Application of fuzzy sets to climatic classification, *Ag. and Forest Meteor*, 35, 165-185.

McCulloch W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophysics*, 7, 115-133.

McKenzie, P. and Alder, M. (1994). Initializing the EM algorithm for use in Gaussian mixture modeling, in *Pattern Recognition in Practice IV ; Multiple Paradigms, Comparative Studies and Hybrid Syst.*, eds. E.S. Gelsema and L. N. Kanal, Elsevier, NY, 91-105.

McLachlan, G. J. and Basford, K. E. (1988). *Mixture Models : Inference and Applications to Clustering*, Marcel Dekker, NY.

Medasani, S. and Krishnapuram, R. (1997). Determination of the number of components in Gaussian mixtures based on agglomerative clustering, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1412-1417.

Medasani, S., Krishnapuram, R. and Keller, J. M. (1999). Are fuzzy definitions of image properties really useful?, *IEEE Trans. Syst., Man and Cyberns.*, in press.

Milligan, G. W. and Cooper, M. C. (1986). A study of the comparability of external criteria for the hierarchical cluster analysis, *Mult. Behav. Res.*, 21, 441-458.

Mitra, S. and Yang, S.-Y. (1999). High fidelity adaptive vector quantization at very low bit rates for progressive transmission of radiographic Images, *J. Electronic Imaging*, in press.

Mitra, S., Castellanos, R. and Pemmaraju, S. (1999). Neuro-fuzzy clustering for image segmentation, *Applications of Neuro-Fuzzy Algorithms*, eds. S.K. Pal and A. Ghosh, Springer-Verlag, in press.

Miyajima, K. and Ralescu, A. L. (1993). Modeling of natural objects including fuzziness and application to image understanding, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1049-1054.

Miyajima, K. and Ralescu, A. L. (1994). Spatial organization in 2D segmented images: representation and recognition of primitive spatial relations, *Fuzzy Sets and Syst.*, 65, 225-236.

Miyamoto, S. (1990). *Fuzzy Sets in Information Retrieval and Cluster Analysis*, Kluwer, Boston.

Miyamoto, S., Miyake, T. and Nakayama, K. (1983). Generation of a psuedothesaurus for information retrieval based on coocurences and fuzzy set operations, *IEEE Trans. Syst., Man and Cyberns.*, 13, 62-70.

Mizumoto, M. (1988). Fuzzy controls under various fuzzy reasoning methods, *Inf. Sci.*, 45, 129-151.

Mizumoto, M. (1989). Pictorial representations of fuzzy connectives: 1: cases of t-norms, t-conorms and averaging operators, *Fuzzy Sets and Syst.*, 31, 217-242.

Moghaddamzadeh, A. and Bourbakis, N. (1997). A fuzzy region growing approach for segmentation of color images, *Patt. Recog.*, 30(6), 867-881.

Mohamed, M. and Gader, P. (1994). Generalization of hidden Markov models using fuzzy integrals. *Proc. Joint NAFIPS/IFIS/NASA Conf.*, TX, 3-7.

Mohamed, M., and Gader, P. (1995). Handwritten word recognition using segmentation-free hidden Markov modeling and segmentation-based dynamic programming techniques. *IEEE Trans. Patt. Anal. and Machine Intell.*, 18(5), 548-554.

Moody, J. E. and Darken, C. J. (1989). Fast learning in networks of locally tuned processing units, *Neural Computation*, 1, 281-294.

Moore, B. (1988). ART1 and pattern clustering, *Proc. 1988 Connectionist Models Summer School*, eds. D.S. Touretzky, G. Hinton and T. J. Sejnowski, Morgan Kaufmann, San Mateo, CA, 174-185.

Moore, J. (1998). A fuzzy logic automatic target detection system for LADAR range images, MS Thesis, U. of Missouri, Columbia, MO.

Morrison, M. and Attikouzel, Y. (1994). An introduction to the segmentation of Magnetic Resonance images, *Aust. Comp. J.*, 26(3), 90-98.

Murofushi, T. and Sugeno, M. (1991). A theory of fuzzy measures: Representations, the Choquet integral, and null sets, *J. Math Anal. and Applications*, 159, 532-549.

Murthy, S.K., Kasif, S. and Salzberg, S. (1994). A system for induction of oblique decision trees, *J. AI Research*, 2, 1-32.

Murthy, C. A. and Pal, S. K. (1990). Fuzzy thresholding: mathematical framework, bound functions and weighted moving average technique, *Patt. Recog. Lett.*, 11, 197-206.

Nakagawa, Y. and Hirota, K. (1995). Fundamentals of fuzzy knowledge base for image understanding *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1137-1142

Nakamori, Y. and Ryoke, M. (1994). Identification of fuzzy prediction models through hyperellipsoidal clustering, *IEEE Trans. Syst. Man and Cybern,* 24(8), 1153-1173.

Namasivayam, A. and Hall, L. O. (1995), The use of fuzzy rules in classification of normal human brain tissues, *Proc. Joint ISUMA-NAFIPS Conf.,* 157-162.

Narazaki, H. and Ralescu, A. (1993). An improved synthesis method for multilayered neural networks using qualitative knowledge, *IEEE Trans. Fuzzy Syst.,* 1(2), 125-137.

Nasraoui, O. and Krishnapuram, R. (1997). A genetic algorithm for robust clustering based on a fuzzy least median of squares criterion, *Proc. NAFIPS Conf.,* Syracuse NY, 217-221.

Newton, S.C., Pemmaraju, S. and Mitra, S. (1992). Adaptive fuzzy leader clustering of complex data sets in pattern recognition, *IEEE Trans. Neural Networks,* 3(5), 794-800.

Nguyen, H. T. and Kreinovich, V. (1998). Methodology of fuzzy control, in*Fuzzy Systems: Modeling and Control,* eds. H. T. Nguyen and M. Sugeno, Kluwer, Boston, MA, 19-62.

Nguyen, H. T. and Sugeno, M., eds., (1998). *Fuzzy Systems: Modeling and Control,* Kluwer, Boston, MA.

Nie, J. H. and Lee, T. H. (1996). Rule-based modeling: fast construction and optimal manipulation, *IEEE Trans. Syst., Man and Cyberns.,* 26(6), 728-738.

Ohashi, Y. (1984). Fuzzy clustering and robust estimation, presentation at the 9th Meeting of SAS Users Group Int., Hollywood Beach, Florida.

Ohta, Y. (1985). *Knowledge-Based Interpretation of Outdoor Natural Color Scenes,* Pitman Advanced Publishing, Boston, MA.

Otsu, N. (1979) A threshold selection method from gray-level histograms, *IEEE Trans. Syst., Man and Cybers.,* 9, 62-66.

Pal K., Pal N. R. and Keller J. M. (1998). Some neural net realizations of fuzzy reasoning, *Int. J. Intell. Systems,*13(9), 859-886.

Pal, N. R. and Bezdek, J. C. (1994). Measuring fuzzy uncertainty, *IEEE Trans. Fuzzy Syst.,* 2(2), 107-118.

Pal, N. R. and Bezdek, J. C. (1995). On cluster validity for the fuzzy c-means model, *IEEE Trans. Fuzzy Syst.,* 3(3), 370-379.

Pal, N. R. and Biswas J. (1997). Cluster validation using graph theoretic concepts, *Patt. Recog.*, 30(6), 847-857.

Pal, N. R. and Chakraborty, S. (1997). A hierarchical algorithm for classification and its tuning by genetic algorithm, *Proc. Int. Conf. Neural Info. Processing*, Springer, Singapore, 1, 404-407.

Pal, N. R. and Chintalapudi K. (1997), A connectionist system for feature selection, *Neural, Parallel and Scientific Computation*, 5(3), 359-381.

Pal, N. R. and Mukhopadhyay (1996). A psychovisual fuzzy reasoning edge detector, *Proc. Int. Conf. on Soft Computing, Methodologies for the Conception, Design and Applications of Intelligent systems*, eds, T. Yamakawa and G. Matsumoto, 1, Iizuka, Japan, 201-204.

Pal, N. R. and Pal, S. K. (1993). A review of image segmentation techniques, *Patt. Recog.*, 26(9), 1277-1294.

Pal, N. R., Bezdek, J. C. and Tsao, E. C. K. (1993). Generalized clustering networks and Kohonen's self-organizing Scheme, *IEEE Trans. Neural Networks*, 4(4), 549-558.

Pal, N. R., Chakraborty, S. and A. Bagchi (1997). RID3 : An ID3 like algorithm for real data, *Inf. Sciences (Applications)*, 96, 271-290.

Pal, N. R., Nandi, S. and Kundu, M. K. (1998). Self-crossover: a new genetic operator and its application to feature selection, *Int. J. Syst. Sci.*, 29(2), 207-212.

Pal, N. R., Pal, K. and Bezdek, J. C. (1997a). A mixed c-means clustering model, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 11-21.

Pal, N. R., Pal, K., Bezdek, J. C. and Runkler, T. A. (1997b). Some issues in system identification using clustering, *Proc. IEEE Int. Conf. on Neural Networks*, 4, IEEE Press, Piscataway, NJ, 2524-2529.

Pal, S. K. (1991). Fuzzy tools for the management of uncertainty in pattern recognition, image analysis, vision and expert system, *Int. J. Syst. Sci.*, 22(3), 511-549.

Pal, S. K. (1992a). Fuzzy set theoretic measure for automatic feature evaluation - II, *Inform. Sci.*, 64, 165-179.

Pal, S. K. (1992b). Fuzziness, image information and scene analysis, in *An Introduction to Fuzzy Logic Applications in Intelligent Systems,* eds. R. R. Yager and L. A. Zadeh, Kluwer, Norwell MA, 147-184.

Pal, S. K. and Bhattacharyya, A. (1990). Pattern recognition techniques in analyzing the effect of thiourea on brain neurosecretory cells, *Patt. Recog. Lett.*, 11, 443-451.

Pal, S. K. and Chakraborty, B. (1986). Fuzzy set theoretic measure for automatic feature evaluation, *IEEE Trans. Syst. Man and Cyberns.*, 16(5), 754-760.

Pal, S. K. and Dutta-Majumder, D.K. (1986). *Fuzzy Mathematical Approach to Pattern Recognition*, Wiley, NY.

Pal, S. K. and Ghosh, A. (1990). Index of area coverage of fuzzy subsets and object extraction, *Patt. Recog. Lett.*, 11, 831-841.

Pal, S. K. and King, R.A. (1981). Image enhancement using smoothing with fuzzy sets, *IEEE Trans. Syst., Man and Cyberns.*, 11(7), 494-501.

Pal, S. K. and King, R.A. (1983). On edge detection of x-ray images using fuzzy sets, *IEEE Trans. Patt. Anal. and Machine Intell.*, 5(1), 69-77.

Pal S. K. and Mitra S. (1992). Multilayer perceptron, fuzzy sets and classification, *IEEE Trans. Neural Networks*, 3(5), 683-697.

Pal, S. K. and Rosenfeld, A. (1988). Image enhancement and thresholding by optimization of fuzzy compactness, *Patt. Recog. Lett.*, 7, 77-86.

Pal, S. K., King, R. A. and Hishim, A. A. (1983a). Automatic grey level thresholding through index of fuzziness and entropy, *Patt. Recog. Lett.* 1, 141-146.

Pal, S. K., King, R.A. and Hishim, A. A. (1983b). Image description and primitive extraction using fuzzy sets, *IEEE Trans. Syst., Man and Cyberns..*, 13(1), 94-100.

Panayirci, E. and Dubes, R. C. (1983), A test for multidimensional clustering tendency, *Patt. Recog.*, 16, 433-444.

Pao, Y.H. (1989). *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA

Parizeau, M. and Plamondon, R. (1995). A fuzzy-syntactic approach to allograph modeling for cursive script recognition, *IEEE Trans. Patt. Anal. and Machine Intell.*, 17(7), 702-712.

Parizeau, M., Plamondon, R. and Lorette, G. (1993). Fuzzy shape grammars for cursive script recognition, in *Advances in Structural and Syntactic Pattern Recognition*, ed. H. Bunke, World Scientific, Singapore, 320-332.

Park, J-S and Keller, J. M. (1997). Fuzzy patch label relaxation in bone marrow cell segmentation, *Proc. IEEE Int. Conference on Syst., Man, and Cyberns.*, IEEE Press, Piscataway, NJ, 1133-1138.

Park, J. and Sandberg, I. W. (1991). Universal approximation using radial basis function networks, *Neural Computation*, 3, 246-257.

Park, J. and Sandberg, I. W. (1993). Approximation and radial basis functions, *Neural Computation*, 5, 305-316.

Pathak, A. and Pal, S.K. (1986). Fuzzy grammars in syntactic recognition of skeletal maturity from X-rays, *IEEE Trans. Syst., Man and Cyberns.*, 16(5), 657-667.

Pavlidis, T. (1980). *Structural Pattern Recognition*, Springer-Verlag, NY.

Pedrycz, W. (1985). Algorithms of fuzzy clustering with partial supervision, *Patt. Recog. Lett.*, 3, 13-20.

Pedrycz, W. (1990a). Direct and inverse problem in comparison of fuzzy data, *Fuzzy Sets and Syst.*, 34, 223-236.

Pedrycz, W. (1990b). Fuzzy sets in pattern recognition : methodology and methods, *Patt. Recog.*, 23(1/2), 121-146.

Pedrycz, W., Bezdek, J. C. and Hathaway, R. J. (1998). Two non-parametric models for fusing heterogeneous fuzzy data, *IEEE Trans. Fuzzy Syst.*, 4(3), 270-281.

Pedrycz, W., Kandel, A. and Zhang, Y. (1998). Neurofuzzy systems, in *Fuzzy Systems : Modeling and Control*, eds. H. Nguyen and M. Sugeno, Kluwer, Boston, 311-380.

Peeva, K. (1991). Fuzzy acceptors for syntactic pattern recognition, *Int. J. Approx. Reasoning*, 5, 291-306.

Peli, T. and Lim, J. (1982). Adaptive filtering for image enhancement, *Optical Engineering*, 21, 108-112.

Peng, S. and Lucke, L. (1994). Fuzzy filtering for mixed noise removal during image processing, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 89-93.

Petersen, J., Stockmanns, G., Kochs, H-D. and Kochs, E. (1997). Automatic feature extraction in clinical monitoring data, *Proc. Fuzzy-Neuro Systems- Computational Intelligence,* eds. A. Grauel, W. Becker and F. Belli, Infix Press, St. Augustine, Germany, 411-418.

Pham, T. and Yan, H. (1997). Fusion of handwritten numeral classifiers based on fuzzy and genetic algorithms, *Proc. NAFIPS Conf.,* 257-262.

Phansalkar and Davé, R. N. (1997). On generating solid models of mechanical parts through fuzzy clustering, *Proc. IEEE Int. Conf. on Fuzzy Syst.,* 1, IEEE Press, Piscataway, NJ, 225-230.

Philip, K.P., Dove, E.L., McPherson, D.D., Gotteiner, N. L., Stanford, W. and Chandran, K.B. (1994). The fuzzy Hough transform - feature extraction in medical images, *IEEE Trans. Med. Imaging,* 13(2), 235-240.

Pienkowski, A. (1989). *Artificial Colour Perception using Fuzzy Techniques in Digital Image Processing,* Verlag TUV Rheinland, Germany.

Piper, J. and Granum, E. (1989). On fully automatic feature measurement for banded chromosome classification, *Cytometry,* 10, 242-255.

Poggio, T. and Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks, *Science,* 247, 978-982.

Powell, M.J.D. (1990). The theory of radial basis function approximation in 1990, *DAMTP 1990/NA11,* Cambridge U., Cambridge, England.

Prasad, N. R. (1998). Neural networks and fuzzy logic, in *Fuzzy Sets in Decision Analysis: Operations Research and Statistics,* ed. R. Slowinski, Kluwer, Boston, MA. 381-400.

Pratt, W.K.(1991). *Digital Image Processing,* 2nd ed., J.Wiley, NY.

Prewitt, J. M. (1970). Object enhancement and extraction in picture processing and psychopictorics, eds. B.S. Lipkin and A. Rosenfeld, Academic Press, NY, 75-149.

Prim, R. C. (1957). Shortest connection matrix network and some generalizations, *Bell Syst. Tech. J.,* 36, 1389-1401.

Qiu, H. and Keller, J. M. (1987). Multispectral segmentation using fuzzy techniques, *Proc. NAFIPS Conf.,,* 374-387.

Quandt, R. E. and Ramsey, J. B. (1978). Estimating mixtures of normal distributions and switching regressions, *J. Amer. Stat. Soc.*, 73, 730-752.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end-games, *Machine Learning*, eds. R. Michalski, J. Carbonell and T. Mitchell, Tioga, Palo Alto, CA.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Learning*, 1, 81-106.

Quinlan, J. R. (1987). Simplifying decision trees, *Int. J. Man-Machine Studies*, 27, 221-234.

Quinlan, J. R. (1993). C4.5: *Programs for Machine Learning*, Morgan Kauffman, San Mateo, CA.

Radecki, T. (1976). Concept of fuzzy thesaurus, *Inf. Proc. and Management*, 12, 313-318.

Ralescu, A. L. and Shanahan, J. G. (1995). Fuzzy perceptual grouping in image understanding, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1268-1272.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods, *J. Amer. Stat. Soc.*, 66, 846-850.

Rappaport, C. and Reidy, D. (1996). Focused array radar for real time imaging and detection, *Proc. SPIE*, 3079, SPIE, Bellingham, WA, 202-212.

Redner, R., Hathaway, R. J. and Bezdek, J. C. (1987). Estimating the parameters of mixture models with modal estimators, *Comm. in Stat. (A)*, 16(9), 2639-2660.

Retz-Schmidt, G. (1988). Various views on spatial relations, *AI Magazine*, Summer Issue, 95-105.

Rezaee, R. M., Nyqvist, C., van der Zwet, P. M. J., Jansen, E. and Reiber, J. H. C. (1995). Segmentation of MR images by a fuzzy c-means algorithm, *Proc. IEEE Conf. on Computers in Cardiology*, IEEE Press, Piscataway, NJ, 21-24.

Rhee, F. (1993). Fuzzy rule generation and inference methods for pattern recognition and computer vision, *Ph.D. Thesis*, U. of Missouri, Columbia, MO.

Rogova, G. (1994). Combining the results of several neural network classifiers. *Neural Networks*, 7, 777-781.

Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton, *Cornell Aeronaut Lab. Report, 85-460-1*, Cornell U., Ithaca, NY.

Rosenfeld, A. (1979). Fuzzy digital topology, *Inf. and Control*, 40, 76-87.

Rosenfeld, A. (1984). The fuzzy geometry of image subsets, *Patt. Recog. Lett.*, 2, 311-317.

Rosenfeld, A. (1992). Fuzzy geometry: An overview, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, 113-118.

Rosenfeld, A. and Haber, S. (1985). The perimeter of a fuzzy set, *Patt. Recog.*, 18, 125-130.

Rosenfeld, A. and Kak, A. (1982). *Digital Picture Processing*, Vol. 2, Academic Press, Orlando, FL.

Rosenfeld, A. and Klette, R. (1985) Degree of adjacency or surroundedness, *Patt. Recog.*, 18, 169-177.

Rosenfeld, A. and Haber, S. (1985) The perimeter of a fuzzy set, *Patt. Recog.*, 18, 125-130.

Roubens, M. (1978). Pattern classification problems and fuzzy sets, *Fuzzy Sets and Syst.*, 1, 239-253.

Roubens, M. (1982). Fuzzy Clustering Algorithms and Their Cluster Validity, *Eur. J. Op. Res.*, 10, 294-301.

Rousseeuw P. J. and Leroy, A.M. (1987). *Robust Regression and Outlier Detection*, John Wiley and Sons, NY.

Roux, L. and Desachy, J. (1997). Multiresources information fusion application for satellite image classification, in *Fuzzy Information Engineering*, eds. D. Dubois, H. Prade and R.R. Yager, Wiley and Sons, NY, 111-121.

Rowley, H., Baluja, S., and Kanade, T. (1998). Neural network-based face detection, *IEEE Trans. Patt. Anal. and Machine Intell.*, 20(1), 23-38.

Rumelhart, D. E. and McClelland, J. L. (1982). An interactive activation model of context effects in letter perception, Part 2 : The contextual enhancement effect and some tests and extension of the model, *Psychological Review*, 89, 60-94.

Rumelhart, D. E. and Zipser, D. (1985). Feature discovery by competitive learning, *Cognitive Science*, 9, 75-112.

Runkler, T. A. (1995). Improved cluster validity measures for the fuzzy c-elliptotypes algorithms, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1331-1335.

Runkler, T. A. and Bezdek, J. C. (1997). Image segmentation using fuzzy clustering with fractal features, *Proc. 1997 IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ., 1393-1398.

Runkler, T. A. and Bezdek, J. C. (1998a). Polynomial membership functions for smooth first order Takagi-Sugeno systems, *Fuzzy-Neuro Systems : Proc. in Artificial Intelligence*, 5, eds. A. Grauel, W. Becker and F. Belli, 382-388.

Runkler, T. A. and Bezdek, J. C. (1998b). RACE; Relational alternating cluster estimation and the wedding table problem, *Fuzzy-Neuro Systems : Proc. in Artificial Intelligence*, 7, ed. W. Brauer, 330-337.

Runkler, T. A. and Bezdek, J. C. (1998c). Function approximation with polynomial membership functions and alternating cluster estimation, *Fuzzy Sets and Syst.*, 101(2), 207-218.

Runkler, T. A. and Bezdek, J. C. (1999). ACE : a tool for clustering and rule extraction, *IEEE Trans Fuzzy Syst.*, in Press.

Runkler, T. A. and Palm, R. H. (1996). Identification of nonlinear systems using regular fuzzy c-elliptotype clustering, *Proc. IEEE Int. Conf. Fuzzy Syst.*, New Orleans, 1026-1030.

Ruspini, E. A. (1969). A new approach to clustering, *Inf. and Control*, 15, 22-32.

Ruspini, E. A. (1970). Numerical methods for fuzzy clustering, *Infor. Sci.*, 2, 319-350.

Ruspini, E. H., Bonissone, P. and Pedrycz, W., eds., (1998). *Handbook of Fuzzy Computation*, Institute of Physics Publ., Bristol, UK.

Russo, F. (1993). A new class of fuzzy operators for image processing: design and implementation, *Proc. 1993 IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 815-820

Russo, F. and Ramponi, G. (1992). Fuzzy operator for sharpening of noisy images, *IEE Electronics Lett.*, 28, 1715-1717.

Russo, F. and Ramponi, G. (1994a). Combined FIRE filters for image enhancement, *Proc. 1994 Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 260-264.

Russo, F. and Ramponi, G. (1994b). Edge extraction by FIRE operators, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 249-253.

Ryan, W., Winter, C. L. and Turner, C. J. (1987). Control of an artificial neural system : the property inheritance network, *Applied Optics*, 21(23), 4961-4971.

Sabin, M.J. (1987). Convergence and consistency of the fuzzy c-means/ISODATA algorithms, *IEEE Trans. Patt. Anal. and Machine Intell.*, 661-668.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology, *IEEE Trans. Syst., Man and Cyberns.*, 21(3), 660-674.

Sahoo, P. K., Soltani, S, Wong, A. K. C. and Chen, Y. C. (1988). A survey of thresholding techniques, *CVGIP*, 41(2), 233-260.

Saint Marc, P., Chen, J. and Medioni., G. (1991). Adaptive smoothing: A general tool for early vision, *IEEE Trans. Pattern Anal. and Machine Intell.*, 13, 514-529.

Sameti, M. and Ward, R. K. (1996). A fuzzy segmentation algorithm for mammogram partitioning, in *Digital Mammography '96*, eds. K. Doi, M. L. Giger, R. M. Nishikawa and R. A. Schmidt, Elsevier Science Publishers, 471-474.

Sameti, M., Ward, R. K., Palcic, B. and Morgan-Parkes, J. (1997). Texture feature extraction for tumor detection in mammographic images, *Proc. IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*, 831-834.

Sammon, J. W. (1969). A nonlinear mapping for data structure analysis, *IEEE Trans. Computers*, 18, 401-409.

Sanfeliu, A. and Fu, K. S. (1983). A distance measure between attributed relational graphs for pattern recognition, *IEEE Trans. Syst., Man and Cyberns.*, 13(3), 353-362.

Sankar, A. and Mammone, R. (1993). Growing and pruning neural tree networks, *IEEE Trans. Computers*, 42(3), 291-299.

Sato, M., Sato, Y. and Jain, L.C. (1997). *Fuzzy Clustering Models and Applications*, Physica-Verlag, Heidelberg, Germany.

Schalkoff, R. (1992). *Pattern Recognition: Statitistical, Structural and Neural Approaches*, John Wiley and Sons, NY.

Schwartz, T. J. (1991). Fuzzy tools for expert systems, *AI Expert*, February, 34-41.

Sebestyen, G. S. (1962). *Decision-Making Processes in Pattern Recognition*, Macmillan, NY.

Seidelmann, G. (1993). Using heuristics to speed up induction on continuous-valued attributes, *Proc. European Conf. on Machine Learning*, ed. P. B. Brazdil, Springer, Berlin, 390-395.

Selim, S. A. and Ismail, M. A. (1984). K-means type algorithms: A generalized convergence theorem and characterization of local optimality, *IEEE Trans. Patt. Anal. and Machine Intell.*, 6(1), 81-87.

Sen, S. and Dave, R. N. (1998). Clustering of relational data containing noise and outliers, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1411-1416.

Senay, H. (1992).Fuzzy command grammars for intelligent interface design, *IEEE Trans. Syst., Man and Cyberns.*, 22(5), 1124-1131.

Serrano-Gotarredona, T., Linares-Barranco, B. and Andreou, A. G. (1998). *Adaptive Resonance Theory Microchips : Circuit Design Techniques*, Kluwer, Boston, MA.

Sethi, I. K. (1990). Entropy nets: from decision trees to neural networks, *Proc. IEEE*, 78(10), 1605-1613.

Sethi, I. K. (1995). Neural implementation of tree classifiers, IEEE Trans. Syst., Man and Cyberns., 25(8), 1243-1249.

Sethi, I. K. and Sarvarayudu, G.P.R. (1982). Hierarchical classifier design using mutual information, *IEEE Trans. Patt. Anal. and Machine Intell.*, 4, 441-445.

Setnes, M., Babuska, R. and Verbruggen, H. B. (1998). Rule based modeling: precision and transparency, *IEEE Trans. Systs., Man and Cyberns.*, C28(1), 165-169.

Shafer, G. (1976). *A Mathematical Theory of Evidence*, Princeton U. Press, Princeton, NJ.

Shankar, B. U. and Pal, N. R. (1994). FFCM: An effective approach for large data sets, *Proc. Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, Iizuka, Japan, 331-332.

Shapiro, L. G. and Haralick, R. M. (1985). A metric for comparing relational descriptions, *IEEE Trans. Patt. Anal. and Machine Intell.*, 7, 90-94.

Shavlik, J.W., Mooney, R. J. and Towell, G.G. (1991). Symbolic and neural learning algorithms: an experimental comparison, *Machine Learning*, 6, 111-143.

Shaw, A. C. (1969). A formal picture description scheme as a basis for picture processing systems, *Inf. and Control*, 14, 9-52.

Shephard, R.N. and Arabie, P. (1979). Additive clustering : representation of similarities as combinations of discrete overlapping properties, *Psychological Review*, 86(2), 87-123.

Shi, H., Gader, P. and Chen, W. (1998). Fuzzy integral filters: properties and parallel implementations, *J. of Real-Time Imaging*, 2(4), 233-241.

Shih, F. Y., Moh, J. and Chang, F. (1992). A new ART-based neural architecture for pattern classification and image enhancement without prior knowledge, *Patt. Recog.*, 25(5), 533-542.

Simpson, P. K. (1993). Fuzzy min-max neural networks - Part 2 : Clustering, *IEEE Trans. Fuzzy Syst.*, 1(1), 32-45.

Sims, S. R. and Dasarathy, B. (1992). Automatic target recognition using a passive multisensor suite, *Optical Engineering*, 31(12), 2584-2593.

Sin, S. K. and deFigueiredo, R. J. P. (1993). Fuzzy system design through fuzzy clustering and optimal predefuzzification, *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Francisco, CA, 190- 195.

Sinha, D., Sinha, P., Dougherty, E. R. and Batman, S. (1997). Design and analysis of fuzzy morphological algorithms for image processing, *IEEE Trans. Fuzzy Syst.*, 5(4), 570-584.

Sinha, D. and Dougherty, E. R. (1992). Fuzzy mathematical morphology, *J. Comm. Image Representation*, 3(3), 286-302.

Sinha, D. and Dougherty, E. R. (1995). A general axiomatic theory of intrinsically fuzzy mathematical morphologies, *IEEE Trans. Fuzzy Syst.*, 3(4), 389-403.

Siy, P. and Chen, S. S. (1974). Fuzzy logic for handwritten numeral character recognition, *IEEE Trans. Syst., Man and Cyberns.*, 570-574.

Slowinski, R., ed., (1998). *Fuzzy sets in Decision Analysis: Operations Research and Statistics*, Kluwer, Boston, MA.

Smith, S. P. and Jain, A. K. (1984). Testing for uniformity in multidimensional data, *IEEE Trans. Patt. Anal. and Machine Intell.*, 6(1), 73-81.

Sneath, P. H. A. and Sokal, R. R. (1973). *Numerical Taxonomy - The Principles and Practice of Numerical Classification*, W. H. Freeman, San Francisco, CA.

Solina, F. and Bajcsy, R. (1990). Recovery of parametric models from range images: the case for superquadrics, *IEEE Trans. Patt. Anal. and Machine Intell.*, 12(2), 131-176.

Srinivasan, R., and Kinser, J. (1998). A foveating fuzzy scoring target recognition system, *Patt. Recog.*, 31(8), 1149-1158.

Srinivasan, R., Kinser, J., Schamschula, J., Shamir, J. and Caulfield, H. J. (1996). Optical syntactic pattern recognition by fuzzy scoring, *Optical Lett.*, 21(11), 815-817.

St. Clair, D. C., Bond, W. E., Rigler, A. K. and Aylward, S. (1992). An evaluation of learning performance in backpropagation and decision-tree classifier systems, *Proc. ACM/SIGAPP Symp. on Applications Computers*, ACM Press, 636-642.

Stanley, R. J., Keller, J. M., Caldwell, C. W. and Gader, P. (1995). Automated chromosome classification limitations due to image processing, *Proc. Rocky Mountain Bioengineering Symp.*, Copper Mountain, CO, 183-188.

Stanley, R. J., Keller, J., Gader, P. and Caldwell, C.W. (1998). Data-driven homologue matching for chromosome identification, *IEEE Trans. Medical Imaging*, 17(3), 451-462.

Stewart, C. (1995). MINPRAN: A new robust estimator for computer vision, *IEEE Trans. Patt. Anal. and Machine Intell.*, 17(10), 925-938.

Strickland, R. N. and Lukins, G. J. (1997). Fuzzy system improves the performance of wavelet-based correlation detectors, *Proc. IEEE Int. Conf. on Patt. Recog.*, 3, 404-407.

Strickland, R. N. and Theodosiou, T. (1998). Fuzzy system for detecting microcalcifications in mammograms, *Proc. SPIE Applications and Science of Neural Networks, Fuzzy Systems and Evolutionary Computation*, 3455, eds. B. Bosachhi, J. C. Bezdek and D. Fogel, SPIE, Bellingham, WA, 317-327.

Suen, C. Y. and Wang, Q.R. (1984). ISOETRP - an interactive clustering algorithm with new objectives, *Patt. Recog.*, 17(2), 211-219.

Suen, C. Y., Nadal, C., Legault, R., Mai, T. and Lam, L. (1992). Computer recognition of unconstrained handwritten numerals. *Proc. IEEE*, 80(7), 1162 - 1180.

Sugeno, M. (1977). Fuzzy measures and fuzzy integrals: A survey, in *Fuzzy Automata and Decision Processes*, North Holland, Amsterdam, 89-102.

Sugeno, M. and Yasukawa, T. (1993). A fuzzy-logic-based approach to qualitative modeling, *IEEE Trans. Fuzzy Syst.*, 1(1), 7-31.

Suh, I. H. and Kim, T. W. (1994). Fuzzy membership function based neural networks with applications to the visual servoing of robot manipulators, *IEEE Trans. Fuzzy Syst.*, 2(3), 203-220.

SUNY Buffalo Postal Address Image Database (1989). State U. of NY at Buffalo, CS Department, Buffalo, NY.

Tahani, H. and Keller, J. M. (1990). Information fusion in computer vision using the fuzzy integral, *IEEE Trans. Syst., Man and Cyberns.*, 20(3), 733-741.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its application to modeling and control, *IEEE Trans. Syst., Man and Cyberns.*, 15(1), 116-132.

Tamura, S. and Tanaka, K. (1973). Learning of fuzzy formal language, *IEEE Trans. Syst., Man and Cyberns.*, 3, 98-102.

Tamura, S., Higuchi, S. and Tanaka, K. (1971). Pattern classification based on fuzzy relations, *IEEE Trans. Syst., Man and Cyberns.*, 1(1), 61-66.

Tanaka, K. and Sugeno, M. (1991). A sudy on subjective evaluations of printed color images, *Int. J. of Approx. Reasoning*, 5(3), 213-222.

Tanaka, K. and Sugeno, M. (1998). Introduction to fuzzy modeling, in *Fuzzy Systems: Modeling and Control*, eds. H. T. Nguyen and M. Sugeno, Kluwer, Boston, MA, 63-90.

Tani, T. and Sakoda, M. (1992). Fuzzy modeling by ID3 algorithm and its application to prediction of heater outlet temparature, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 923-930.

Tao, C. W., Thompson, W. E. and Taur, J.S. (1993). Fuzzy if-then approach to edge detection, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1356-1360.

Tarel, J.-P. and Boujemaa, N. (1995). A fuzzy 3D registration method, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1359-1365.

Taubin, G. (1991). Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with application to edge and range image segmentation, *IEEE Trans. Patt. Anal. and Machine Intell.*, 13(11), 1115-1138.

Tax, D. M., Duin, R. P. W. and van Breukelen, M. (1997). Comparison between product and mean classifier combination rules, *Proc. Int. Workshop on Stat. Techniques in Patt. Recog.*, eds. P. Pudil, J. Novovicova, J. Grim, Prague, Inst. of Inf. Theory and Automation, Academy of Sciences of the Czech Republic, Prague, 165-170.

Thawonmas R. and Abe S. (1997). A novel approach to feature selection based on analysis of class regions, *IEEE Trans. Syst., Man and Cyberns.*, B27(2), 196-207.

Thole, U. and Zimmermann, H. -J. (1979). On the stability of the minimum and product operations, *Fuzzy Sets and Syst.*, 2, 167-180.

Thomason, M. G. (1973). Finite fuzzy automata, regular fuzzy languages and pattern recognition, *Patt. Recog.*, 5, 383-390.

Thorndike, R. L. (1953). Who belongs in the family?, *Psychometrika*, 18, 267-276.

Titterington, D., Smith, A. and Makov, U. (1985). *Statistical Analysis of Finite Mixture Distributions*, Wiley, NY.

Tolias, Y. A. and Panos, S. M. (1998). Image segmentation by a fuzzy clustering algorithm using adaptive spatially constrained membership functions, *IEEE Trans. Systs., Man and Cyberns.*, A28 (3), 359-369.

Tou, J. T. (1979). DYNOC - A dynamic optimum cluster-seeking technique, *Int. J. Comp. Syst. Science*, 8, 541-547.

Tou, J. T. and Gonzalez, R. (1974). *Pattern Recognition Principles*, Addison-Wesley, Reading, MA.

Toussaint, G. T. (1974). Bibliography on estimation of misclassification, *IEEE Trans. Inf. Theory*, 20, 472-479.

Trauwaert, E. (1987). $\mathcal{L}_1$ in fuzzy clustering, *Statistical Data Analysis based on the* $\mathcal{L}_1$ *Norm*, ed. Y. Dodge, Elsevier, Amsterdam, 417-426.

Trauwaert, E. (1988). On the meaning of Dunn's partition coefficient for fuzzy clusters, *Fuzzy Sets and Syst.*, 25, 217-242.

Trauwaert, E., Kaufman, L. and Rouseeuw, P. J. (1988). Fuzzy clustering algorithms based on the maximum likelihood principle, *Fuzzy Sets and Syst.*, 25, 213-227.

Trivedi, M. and Bezdek, J. C. (1986). Low level segmentation of aerial images with fuzzy clustering, *IEEE Trans. Syst., Man and Cyberns.*, 16(4), 580-598.

Tsai, W. H. and Fu, K. S. (1979). Error correcting isomorphism of attributed relational graphs for pattern analysis, *IEEE Trans. Syst., Man and Cyberns.*, 9(12), 757-768.

Tucker, W. (1987). Counterexamples to the convergence theorem for fuzzy isodata clustering algorithms, *The Analysis of Fuzzy Information*, ed. J. Bezdek, 3, 109-121, CRC Press, Boca Raton, FL.

Tukey, J. (1977). *Exploratory Data Analysis*, Addison-Wesley, Reading, MA.

Tumer, K. and Ghosh, J. (1998). Classifier combining through trimmed means and order statistics, *Proc. Int. Joint Conf. on Neural Networks*, IEEE Press, Piscataway, NJ, 757-762.

Tyan, C. and Wang, P. (1993). Image processing - enhancement, filtering and edge detection using the fuzzy logic approach, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 600-605.

Udupa, J. K. (1992). Applications of digital topology in medical three-dimensional imaging, *Topology and its Applications*, 46, 181-197,

Udupa, J. K. (1994). Multidimensional digital boundaries, *CVGIP*, 56(4), 311-323.

Udupa, J. K. and Samarasekera, S. W. (1996). Fuzzy connectedness and object definition: theory, algorithms and applications in image segmentation, *Graphical Models and Image Processing*, 58(3), 246-261.

Udupa, J. K., Odhner, D., Tian, J., Holland, G. and Axel, L. (1997a). Automatic clutter-free volume rendering for MR angiography using fuzzy connectedness, *Proc. SPIE Medical Imaging : Image Processing*, 3034, SPIE, Bellingham, WA, 114-119.

Udupa, J. K., Samarasekera, S. W., Miki, Y., van Buchem, M. A. and Grossman, R. I. (1997b). Multiple schlerosis lesion quantification using fuzzy-connectedness principles, *IEEE Trans. Medical Imaging*, 16(5), 598-609.

Umano, M., Okamoto, H., Hatano, I., Tamura, H., Kawachi, F., Umedzu, S. and Kinoshita, J. (1994). Fuzzy decision trees by fuzzy ID3 algorithm and its application to diagnosis systems, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 2113-2118.

Utgoff, P. (1989). Incremental induction of decision trees, *Machine Learning*, 4(2), 161-186.

Vaidyanathan, M., Clarke, L.P., Velthuizen, R. P., Phuphanich, S., Bensaid, A. M., Hall, L. O., Bezdek, J. C., Greenberg, H., Trotti, A. and Silbiger, M. (1995). Comparison of supervised MRI segmentation methods for tumor volume determination during therapy, *Mag. Res. Imaging*, 13(5), 719-728.

Vajda, I. (1970). A note on discrimination information and variation, *IEEE Trans. Inf. Theory*, 16, 771-772.

Velthuizen, R. P., Hall, L. O. and Clarke, L. P. (1996). Feature extraction with genetic algorithms for fuzzy clustering, *Biomed. Engineering Appl., Basis and Communications*, 8(6), 496-517.

Velthuizen, R. P., Hall, L. O., Clarke, L. P. and Silbiger, M. L. (1997). An investigation of mountain method clustering for large data sets, *Patt. Recog.*, 30(7), 1121-1135.

Wang L., Wang M., Yamada M., Seki H., and Itoh H. (1996). Fuzzy reasoning for image compression using adaptive triangular plane patches, *Proc, Int. Conf. on Soft Computing*, 2, eds. T. Yamakawa G. Matsumoto, World Scientific, Singapore, 753-756

Wang, D., Keller, J., Carson, C.A., McAdoo, K., and Bailey, C. (1998). Use of fuzzy logic-inspired features to improve bacterial recognition through classifier fusion, *IEEE Trans. Syst., Man and Cyberns.*, 28(4), 583-591.

Wang, D., Wang, X. and Keller, J. M. (1997). Determining fuzzy integral densities using a genetic algorithm for pattern recognition, *Proc. NAFIPS Conf.*, 263-267.

Wang, F. (1990a). Fuzzy supervised classification of remote sensing images, *IEEE Trans. Geosciences and Remote Sensing*, 28, 194-201.

Wang, F. (1990b). Improving remote sensing image analysis through fuzzy information representation, *Photogrammetric Engineering and Remote Sensing*, 56, 1163-1169.

Wang, L. X. and Mendel, J. M. (1992). Generating fuzzy rules by learning from examples, *IEEE Trans. Syst. Man and Cyberns.* 22(6), 1414-1427.

Wang, L., Wang, M., Yamada, M., Seki, H. and Itoh, H. (1996), Fuzzy reasoning for image compression using adaptive triangular plane patches, *Proc. Int. Conference on Soft Computing*, eds. T. Yamakawa and G. Matsumoto, Iizuka, Japan, World Scientific, Singapore, 753-756.

Wang, Q. R. and Suen, C. Y. (1983). Classification of Chinese characters by phase feature and fuzzy logic search, *Proc. Int. Conf. Chinese Inf. Proc.*, 1, Beijing, Oct., 133-155.

Wang, Q. R. and Suen, C. Y. (1984). Analysis and design of a decision tree based on entropy reduction and its application to large character recognition set, *IEEE Trans. Patt. Anal. and Machine Intell.*, 6(4), 406-417.

Wang, Q. R. and Suen, C. Y. (1987). Large tree classifier with heuristic search and global training, *IEEE Trans. Patt. Anal. and Machine Intell.*, 9(1), 91-102.

Wang, T. C. and Karayiannis, N. B. (1997). Compression of digital mammograms using wavelets and learning vector quantization, in *SPIE Proc. Applications of Artificial Neural Networks in Image Processing II*, 3030, SPIE, Bellingham, WA, 44-55.

Wang, W., Wang, Z. and Klir, G. (1998). Genetic algorithms for determining fuzzy measures from data, *J. Intell. and Fuzzy Syst.*, 6(2), 171-184.

Wang, X. and Keller, J. M. (1999a). Human-based spatial relationship generalization through neural/fuzzy approaches, *Fuzzy Sets and Syst.*, 101(1), 5-20.

Wang, X. and Keller, J. M. (1999b). A fuzzy rule-based approach for scene description involving spatial relationships, *Comp. Vision and Image Understanding*, in review.

Wang, X., Keller, J. M. and Gader, P. (1997). Using spatial relationships as features in object recognition, *Proc. NAFIPS Conf.*, 160-165.

Wang, Z. and Klir, G. (1992). *Fuzzy Measure Theory*, Plenum Press, NY.

Watkins, F. (1990). *Cubicalc*, Computer Software Manual, HyperLogic Corporation, IBM-PC.

Weber, R. (1992). Fuzzy ID3 : a class of methods for automatic knowledge acquisition, *Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks*, Iizuka, Japan, July, 265-268.

Wee, W. G. (1967). On generalizations of adaptive algorithms and applications of the fuzzy sets concept to pattern classification, *Ph.D. Thesis*, Purdue U., W. Lafayette, 1967.

Wee, W. G. and Fu, K. S. (1969). A formulation of fuzzy automata and its application as a model of learning systems, *IEEE Trans. Syst. Science and Cyberns.*, SSC-5(3), 215-223.

Wei, W. and Mendel, J. M. (1994). Optimality tests for the fuzzy c-means algorithms, *Patt. Recog.*, 27(11), 1567-1573.

Weiss, S. M. and Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets and machine learning classification methods, *Proc. 11-th IJCAI Conf.*, 781-787.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems that Learn*, Morgan Kauffman, San Mateo, CA.

Werbos, P. (1974). Beyond Regression : New Tools for Prediction and Regression in the Behavioral Sciences, *Ph.D. Thesis*, Harvard U., Cambridge, MA.

Widrow, B. and Stearns, S. D. (1985). *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, NJ.

Windham, M. P. (1981). Cluster validity for fuzzy clustering algorithms, *Fuzzy Sets and Syst.*, 177-183.

Windham, M. P. (1982), Cluster validity for the fuzzy c-means clustering algorithm, *IEEE Trans. Patt. Anal. and Machine Intell.*, 4(4), 357-363.

Windham. M. P. (1985). Numerical classification of proximity data with assignment measures, *J. Classification* 2, 157-172.

Winston, P. (1975). *The Psychology of Computer Vision*, McGraw-Hill, NY.

Wolfe, J. H. (1970). Pattern clustering by multivariate mixture analysis, *Multivariate Behavioral Research*, 5, 329-350.

Woodbury, M. A. and Clive, J. A. (1974). Clinical pure types as a fuzzy partition, *J. Cybern.*, 4(3), 111-121.

Woods, K., Kegelmeyer, W., and Boyer, K., (1997). Combination of multiple classifiers using local accuracy estimates, *IEEE Trans. Patt. Anal. and Machine Intell.*, 19(4), 405-410.

Wootton, J., Keller, J. M., Carpenter, C. and Hobson, G. (1988). A multiple hypothesis rule-based automatic target recognizer, in *Pattern Recognition, Lecture Notes in Comp. Science*, ed. J. Kittler, 301, (Springer-Verlag) 315-324.

Xie, X.L. and Beni, G. A. (1991). A validity measure for fuzzy clustering, *IEEE Trans. Patt. Anal. and Machine Intell.*, 13(8), 841-846.

Xu, L., Krzyzak, A. and Suen, C. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition, *IEEE Trans. Syst., Man and Cyberns..*, 22(3), 418-435.

Yager, R. R. (1980). On a general class of fuzzy connectives, *Fuzzy Sets and Syst.*, 4, 235-242.

Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Trans. Syst., Man and Cyberns.*, 18(1),183-190.

Yager, R. R. and and Filev, D.P. (1994a). Approximate clustering by the mountain method, *IEEE Trans. Syst., Man and Cyberns.*, 24(8), 1279-1283.

Yager, R. R. and Filev, D. P. (1994b). Generation of fuzzy rules by mountain clustering, *J. Intell. and Fuzzy Syst.*, 2, 209-219.

Yager, R. R. and Filev, D. P. (1998). Fuzzy rule based models in approximate reasoning, in *Fuzzy Systems : Modeling and Control*, eds. H. T. Nguyen and M. Sugeno, Kluwer, Boston, MA, 91-134.

Yager, R. R. and Kacprzyk, J., eds., (1997). *The Ordered Weighted Averaging Operators: Theory and Applications*, Kluwer, Norwell, MA.

Yair, E., Zeger, K. and Gersho, A. (1992). Competitive learning and soft competition for vector quantizer design, *IEEE Trans. SP*, 40(2), 294-309.

Yamakawa, T. (1990). Pattern recognition hardware system employing a fuzzy neuron, *Proc. Int. Conf. on Fuzzy Logic*, Iizuka, Japan, 943-948.

Yamakawa, T., Uchino, E., Miki, T. and Kusanagi, H. (1992). A new fuzzy neuron and its application to system identification and prediction of system behavior, *Proc. Int. Conf. on Fuzzy Logic and Neural Networks* , Iizuka, Japan, 477-483.

Yan, B. and Keller, J. M. (1996). A heuristic simulated annealing algorithm of learning possibility measures for multisource decision making, *Fuzzy Sets and Syst.*, 77, 87-109.

Yan, H. (1993). Prototype optimization for nearest neighbor classifiers using a two-layer neural network, *Patt. Recog.*, 26(2), 317-324.

Yang, M. S. (1993). A survey of fuzzy clustering, *Math. Comput. Modeling*, 18(11), 1-16.

Yen, J. and Chang, C.W. (1994). A multi-prototype fuzzy c-means algorithm, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 539-543.

Yokoya, N. and Levine, M. D. (1989). Range image segmentation based on differential geometry: a hybrid approach, *IEEE Trans. Patt. Anal. and Machine Intell.*, 11(6), 643-694.

Yoshinari, Y., Pedrycz, W. and Hirota, K. (1993). Construction of fuzzy models through clustering techniques, *Fuzzy Sets and Syst.*, 54(2), 157-166.

Yuan, Y. and Shaw, M. J. (1995). Induction of fuzzy decision trees, *Fuzzy Sets and Syst.*, 69, 125-139.

Zadeh, L. A. (1965). Fuzzy Sets, *Inf. and Control*, 8, 338-353.

Zadeh, L. A. (1971). Similarity relations and fuzzy orderings, *Inf. Sci.*, 177-200.

Zangwill, W. (1969). *Non-Linear Programming: A Unified Approach*, Englewood Cliffs, NJ: Prentice Hall, 1969.

Zeidler, J., Schlosser, M., Ittner, A. and Posthoff, C. (1996). Fuzzy decision trees and numerical attributes, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 985-990.

Zenner, B. R. C., Caluwe, M. M. D. and Kerre, E. E. (1985). Retrieval systems using fuzzy expressions, *Fuzzy Sets and Syst.*, 17, 9-22.

Zhang, W. and Sugeno, M. (1993). A fuzzy approach to scene understanding, *Proc. Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 564-569.

Zimmermann, H-J. and Zysno, P. (1980). Latent connectives in human decision making, *Fuzzy Sets and Syst.*, 4 (1), 37-51.

Zimmermann, H-J. and Zysno, P. (1983). Decisions and evaluations by hierarchical aggregation of information, *Fuzzy Sets and Syst.*, 10 (3), 243-260.

Zugaj, D. and Lattuati, V. (1998). A new approach of color image segmentation based on fusing region and edge segmentation outputs, *Patt. Recog.*, 31(2), 105-113.

Zurada, J. (1992). *Introduction to Artificial Neural Systems*, West Publ. Co., St. Paul, MN.

Zurada, J., Marks, R. and Robinson, C. (1994). Introduction, *Computational Intelligence: Imitating Life*, eds. J. Zurada, R. Marks and C. Robinson, IEEE Press, Piscataway, NJ.

References *not cited* in the text

Babuska, R., M. Setnes, U. Kaymak and H. R. Lemke, Rule base simplification with similarity measures, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, 3, 1642-1647, 1996.

Bezdek, J. C., Hall, L.O. and Clarke, L. P. (1993). Review of MR image segmentation techniques using pattern recognition, *Med. Physics*, 20, 1033-1047.

Bezdek, J. C., Hathaway, R. J. and Windham, M. (1991). Numerical comparison of the RFCM and AP algorithms for clustering relational data, *Patt. Recog.*, 24(8), 783-791.

Boudraa, A. E., Champier, J., Damien, J., Besson, J. E., Bordet, J. C. and Mallet, J.J. (1992). Automatic left ventricular cavity detection using fuzzy ISODATA and connected-components labeling algorithms, *Proc. IEEE Conf. in Medicine and Biology*, CH3207-8, Piscataway, NJ, 1895-1896.

Boujemaa, N. and Stamon, G. (1994). Fuzzy modeling in early vision application to medical image segmentation, in *Proc. Int. Conf. on Image Anal. and Processing*, ed. S. Impedovo, World Scientific Singapore, 649-656.

Brandt, M. E., Bohan, T. P., Kramer, L. A. and Fletcher, J. M. (1994). Estimation of CSF, white and gray matter volumes in hydrocephalic children using fuzzy clustering of MR images, *Computerized Medical Imaging and Graphics*, 18(1), 25-34.

Brandt, M. E., Fletcher, J. M. and Bohan, T. P. (1992). Estimation of CSF, white and gray matter volumes from MRIs of hydrocephalic and HIV-positive subjects, *Proc. SimTec/WNN*, IEEE Press, Piscataway, NJ, 643-650.

Buckley, J.J. and Hayashi, Y. (1994). Fuzzy neural nets and applications, *Fuzzy Sets and AI*, 1, 11-41.

Carazo, J. M., Rivera, F.F., Zapata, E.L., Radermacher, M. and Frank, J. (1990). Fuzzy sets-based classification of electron microscopy images of biological macromolecules with an application to ribsomeal particles, *J. Microscopy*, 157(2), 187-203.

Chang, C. W., Hillman, G.R., Ying, H., Kent, T. and Yen, J. (1994). Segmentation of rat brain MR images using a hybrid fuzzy system, *Proc. Joint NAFIPS/IFIS/NASA Conf.*, 55-59.

Chang, C. W., Hillman, G.R., Ying, H., Kent, T. and Yen, J. (1995). Automatic labeling of human brain structures in 3D MRI using fuzzy logic, *Proc. CFSA/IFIS/SOFT Conf.* World Scientific, 27-34.

Chang, C.W., Hillman, G.R., Ying, H. and Yen, J. (1995). A two stage human brain MRI segmentation scheme using fuzzy logic, *Proc. IEEE Int. Conf. on Fuzzy Systems*, IEEE Press, Piscataway, NJ, 649-654.

Choe, H. and Jordan, J. B. (1992). On the optimal choice of parameters in a fuzzy c-means algorithm, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 349-354.

Conover, W. J., Bement, T. R. and Iman, R. L. (1979), On a method for detecting clusters of possible uranium deposits, *Technometrics* 21, 277-282.

Davé R. N. and K. J. Patel, Fuzzy ellipsoidal-shell clustering algorithm and detection of elliptical shapes, *SPIE Proc. Intelligent Robots and Computer Vision IX*, ed. D.P. Casasent, 1607, 320-333.

Davé, R. N. (1992). Boundary detection through fuzzy clustering, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 127-134.

Davé, R. N. (1993). Robust fuzzy clustering algorithms, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, IEEE Press, Piscataway, NJ, 1281-1286.

De La Paz, R., Berstein, R., Hanson, W. and Walker, M. (1986). Approximate fuzzy c-means (AFCM) cluster analysis of medical magnetic resonance image (MRI) data - a system for medical research and education, *IEEE Trans. Geosci. and Remote Sensing*, GE25, 815-824.

De Mori, R. (1983). *Computerized Models of Speech Using Fuzzy Algorithms*, Plenum Press, NY.

De Mori, R. and Laface, P. (1980). Use of fuzzy algorithms for phonetic and phonemic labeling of continuous speech, *IEEE Trans. Patt. Anal. and Machine Intell.*, 2, 136-148.

de Oliveira, M.C. and Kitney, R. I. (1992). Texture analysis for discrimination of tissues in MRI data, *Proc. Computers in Cardiology*, IEEE Press, Piscataway, NJ, 481-484.

Dellepiane, S. (1991). Image segmentation: Errors, sensitivity and uncertainty,*IEEE Trans. EMBS,* 253-254.

Devi, B.B. and Sarma, V.V.S. (1986). A fuzzy approximation scheme for sequential learning in pattern recognition, *IEEE Trans. Syst., Man and Cyberns.,* 16, 668-679.

Di Gesu, V. (1994). Integrated fuzzy clustering, *Fuzzy Sets and Syst.*, 68, 293-308.

Di Gesu, V. and Romeo, L. (1994). An application of integrated clustering to MRI segmentation, *Patt. Recog. Lett.*, 731-738.

Di Gesu, V., De La Paz, R., Hanson, W.A. and Berstein, R. (1991). Clustering algorithms for MRI, in *Lecture notes for medical informatics*, eds. K.P. Adlassing, B. Grabner, S. Bengtsson and R. Hansen, Springer, 534-539.

Di Nola, A., Sessa, S., Pedrycz, W. and Sanchez, E. (1989). *Fuzzy Relational Equations and Their Applications in Knowledge Engineering*, Kluwer, Dordrecht.

Drakopoulos, J.A. and Hayes-Roth, B. (1998). tFPR: A fuzzy and structural pattern recognition system of multivariate time-dependent pattern classes based on sigmoidal functions, *Fuzzy Sets and Syst.*, 99(1), 57-72.

Dzwinel, W. (1995). In search of the global minimum in problems of feature extraction and selection, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1326-1330.

Feldkamp, L.A., Puskorius, G.V., Yuan, F. and Davis, L. I. Jr. (1992). Architecture and training of a hybrid neural-fuzzy system, *Proc. Int. Conf. on Fuzzy Logic and Neural Networks*, Iizuka, Japan, 131-134.

Full, W., Ehrlich, R. and Bezdek, J. C. (1982). Fuzzy QMODEL: A new approach for linear unmixing, *J. Math. Geo.*, 14(3), 259-270.

Gader, P. and Keller, J. M. (1996). Fuzzy methods in handwriting recognition: An overview, *Proc. NAFIPS Conf.*, 137-141.

Gader, P., and Keller, J. M. (1994). Fuzzy logic in handwritten word recognition, *Proc. IEEE Int. Congress on Fuzzy Syst.*, 910-917.

Granath, G. (1984). Application of fuzzy clustering and fuzzy classification to evaluate provenance of glacial till, *J. Math Geo.*, 16(3), 283-301.

Hillman, G. R., Chang, C.W., Ying, H., Kent, T.A. and Yen, J. (1995). Automatic system for brain MRI analysis using a novel combination of fuzzy rule-based and automatic clustering techniques, *SPIE Proc. Med. Imaging: Image Processing*, ed. M. H. Lowe, 2434, SPIE, Bellingham, WA, 16-25.

Hirota, H. and Pedrycz, W. (1994). OR/AND neuron in modeling fuzzy set connectives, *IEEE Trans. Fuzzy Syst.*, 2, 151-161.

Hirota, K. and Pedrycz, W. (1986). Subjective entropy of probabilistic sets and fuzzy cluster analysis, *IEEE Trans. Syst., Man and Cyberns.*, 16, 173-179.

Hong, D. Z., Sarkodie-Gyan, T., Campbell, A.W. and Yan, Y. (1998). A prototype indexing approach to 2-D object description and recognition, Patt. Recog., 31(6), 699-725.

Huang, P. H. and Y. S. Chang, Fuzzy rule based qualitative modeling, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, 1261-1265.

Huntsberger, T., Jacobs, C. L and Cannon, R.L. (1985). Iterative fuzzy image segmentation, *Patt. Recog.*,18, 131-138.

Huntsberger, T., Rangarajan, C. and Jayaramamurthy, S.N. (1986). Representation of Uncertainty in Comp. Vision Using Fuzzy Sets, *IEEE Trans. Comp.*, C-35, 145-156.

Hwang, S. Y., Lee, H.S. and Lee, J. J. (1996). General fuzzy acceptors for syntactic pattern recognition, *Fuzzy sets and Syst.*, 80(3), 397-401.

Ikoma, N., Pedrycz, W. and Hirota, K. (1993). Estimation of fuzzy relational matrix by using probabilistic descent method, *Fuzzy Sets and Syst.*, 57, 335-349.

Kang, S. J. and Kwon, Y.R. (1995). A tightly coupled approach to fuzzy syntactic parsing and neural networks for event-synchronous signal inspection, *J. of Intell, and Fuzzy Syst.*, 3(3), 215-227.

Keller J. M. and Tahani, H. (1992). Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks, *Int. J. Approximate Reasoning*, 6, 1992, 221-240.

Keller, J. M. (1994). Computational intelligence in high level computer vision: determining spatial relationships, in *Computational Intelligence: Imitating Life*, eds. J. Zurada, R. Marks II, C. Robinson, IEEE Press, Piscataway, NJ, 81-91.

Keller, J. M. and Carpenter, C. (1990). Image segmentation in the presence of uncertainty, *Int. J. of Intell. Syst.*, 5(2), 193-208.

Keller, J. M. and Chen, Z. (1994). Image segmentation via fuzzy additive hybrid networks, *Proc. Joint NAFIPS/IFIS/NASA Conf.*, 60-64.

Keller, J. M. and Gader, P. (1995). Fuzzy logic and the principle of least commitment in computer vision, *Proc. IEEE Conf. on Syst., Man and Cybers.*, 4621-4625.

Keller, J. M. and Osborn, J. (1995). A reward/punishment scheme to learn fuzzy densities for the fuzzy integral, *Proc. IFSA Congress*, 97-100.

Keller, J. M. and Tahani, H. (1992). The fusion of information via fuzzy integration, *Proc. NAFIPS Conf.*, 468-477.

Keller, J. M. and Yan, B. (1992). Possibility expectation and its decision making algorithm, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, 661-668.

Keller, J. M., Wang, D., Carson, C.A., McAdoo, K., and Bailey, C. (1995). Improved recognition of *E. coli* O157:H7 bacteria from pulsed-field gel electrophoresis images through the fusion of neural networks with fuzzy training data, *Proc. IEEE Int. Conf. on Neural Networks*, 1606-1610.

Keller, J. M., Yager, R. R. and Tahani, H. (1992). Neural network implementation of fuzzy logic, *Fuzzy Sets and Syst.*, 45, 1-12.

Kersten, P. and Keller, J. M. (1998). Robust fuzzy snakes, *Proc. IEEE Int. Conference on Fuzzy Syst.*, 1554-1559.

Key, J.A., Maslanik, L. and Barry R.G. (1989). Cloud classification from satellite data using a fuzzy-sets algorithm - a polar example, *Int. J. Remote Sensing*, 10, 1823-1842.

Kovalerchuk, B., Triantaphyllou, E., Ruiz, J.F. and Clayton, J. (1997). Fuzzy logic in computer-aided breast cancer diagnosis: analysis of lobulation, *Artificial Intelligence in Medicine*, 11(1), 75-85.

Kroll, A. (1996). Identification of functional fuzzy models using multidimensional reference fuzzy sets, *Fuzzy Sets and Syst.*, 80, 149-158.

Kuncheva, L. I. (1993). An aggregation of pro and con evidence for medical decision support systems, *Comput. Biol. Med.*, 23(6), 417-424.

Kundu, S. and Chen, J. (1994). FLIC: Fuzzy linear invariant clustering with applications in fuzzy control, *Proc. Joint NAFIPS/IFIS/NASA Conf.*, 196- 200.

Larsen, H. L. and Yager, R. R. (1990). An approach to customized end user views in information retrieval systems, in *Multiperson Decision Making Models Using Fuzzy Sets and Possibility Theory*, eds. J. Kacprzyk and M. Fedrizzi, Kluwer, 128-139.

Larsen, H. L. and Yager, R. R.(1997). Query fuzzification for internet information retrieval, in *Fuzzy Information Engineering: A Guided Tour of Applications*, eds. D. Dubois, H. Prade, H. and R. R. Yager, John Wiley and Sons, NY, 291-310.

Lee, H. M., Sheu, C.C. and Chen, J. M. (1998). Handwritten Chinese character recognition based on primitive and fuzzy features via the SEART neural net model, Applied Intell., 8(3), 269-285.

Lesczynski, K., Penczek, P. and Grochulski, W. (1985). Sugeno fuzzy measures and fuzzy clustering, *Fuzzy Sets and Syst.*, 15, 147-158.

Li, H. D., Kallergi, M., Clarke, L. P., Jain, V. K. and Clark, R. A. (1995). Markov random field for tumor detection in digital mammography, *IEEE Trans. on Medical Imaging*, 14 (3), 565-576.

Liang, Z. (1993). Tissue classification and segmentation of MR Images, *IEEE EMB SMagazine*, 12(1), 81-85.

Lu, Y. and Yamaoka, F. (1997). Fuzzy integration of classification results, *Patt. Recog.*, 30(11), 1877-1891.

Marks, L., Dunn, E., and Keller, J. M. (1995). Multiple criteria decision making (MCDM) using fuzzy logic: An innovative approach to sustainable agriculture, *Proc. Joint ISUMA/NAFIPS' Conf.*, 503-508.

Meisels, A, Kandel, A. and Gecht, G. (1989). Entropy and the recognition of fuzzy letters, *Fuzzy Sets and Syst.*, 31, 297-309.

Menhardt, W. and Schmidt, K. H. (1988). Computer vision on magnetic resonance images, *Patt. Recog. Lett.* 8, 73-85 (1988).

Mitra, S. and Pal, S. K. (1992). Multilayer perceptrons, fuzzy sets and classification, *IEEE Trans. Neural Networks*, 3(5), 683-697.

Muroga, S. (1971). *Threshold Logic and Its Applications*, J. Wiley, NY.

Nafarieh, A. and Keller, J. M. (1991). A new approach to inference in approximate reasoning, *Fuzzy Sets and Syst.*, 41(1), 17-37.

Nafarieh, A., and Keller, J. M. (1991). A fuzzy logic rule-based automatic target recognizer, *Int. J. of Intell. Syst.*, 6(3), 295-312.

Nakagowa, K. and Rosenfeld, A (1978). A note on the use of local max and min operations in digital picture processing, *IEEE Trans. Syst. Man and Cyberns*, 8, 632-635.

Nath, P. and Lee, T.T. (1983). On the design of classifier with linguistic variables as input, *Fuzzy Sets and Syst.*, 11, 265-286.

Pal, S.K. and Majumder, D. (1977). Fuzzy sets and decision making approaches in vowel and speaker recognition, *IEEE Trans. Syst., Man and Cyberns.*, 7, 625-629.

Pal, S. K. and Mitra, S. (1990). Fuzzy dynamic clustering algorithm, *Patt. Recog. Lett.*, 11(8), 525-535.

Pal, S.K and Pramanik, P.K. (1986). Fuzzy measures in determining seed points in clustering, *Patt. Recog. Lett.*, 4(3), 159-164.

Pal, S. K., Datta, A.K. and Majumder, D. (1978), Adaptive learning algorithm in classification of fuzzy patterns : an application to vowels in CNC context, *Int. J. Syst. Sci.*, 9(8), 887- 897.

Pathak, A. and Pal, S.K. (1990). Generalised guard zone algorithm (GGA) for learning: automatic selection of threshold, *Patt. Recog.*, 23(3/4), 325- 335.

Pathak, A. and Pal, S.K. (1992). Effect of wrong samples on the convergence of learning processes-II: a remedy, *Inform. Sci.*, 60(1/2), 77-105.

Pedrycz, W. (1991). Neurocomputations in relational systems, *IEEE Trans. Pattern Anal. and Machine Intell.*, 13, 289-296.

Pedrycz, W. (1993). *Fuzzy Control and Fuzzy Systems*, 2nd edition, Research Studies Press/J.Wiley, Taunton, NY.

Pedrycz, W. (1993). Fuzzy neural networks and neurocomputations, *Fuzzy Sets and Syst.*, 56, 1993, 1-28.

Pedrycz, W. (1995). *Fuzzy Set Engineering*, CRC Press, Boca Raton, FL.

Pedrycz, W. and Rocha, A.F. (1993). Fuzzy-set based models of neurons and knowledge-based networks, *IEEE Trans. Fuzzy Syst.*, 1, 254-266.

Peleg, S. and Rosenfeld, A. (1981). A min-max medial axis transformation, *IEEE Trans. Patt. Anal. and Machine Intell.*, 3, 208-210.

Ripley, B. D. and Rasson, J. P. (1977), Finding the edge of a Poisson forest, *Jour. Applied Probability*, 14, 483-491.

Riseman E. M. and Hanson, A. R. (1988). A Methodology for the development of general knowledge-based vision systems, in *Vision Systems and Cooperative Computation*, ed. M. A. Arbib, MIT Press, Cambridge MA, 1988.

Rocha, A.F. (1992). *Neural Nets: A Theory for Brain and Machine.* Lecture Notes in Artificial Intelligence, 638, Springer-Verlag, Berlin.

Saitta, L. and Torasso, P. (1981). Fuzzy characteristics of coronary disease, *Fuzzy Sets and Systs.*, 5, 245-258.

Schneeweiss, W.G.(1989). *Boolean Functions with Engineering Applications*, Springer-Verlag, Berlin

Shi, H., Gader, P., and Keller, J. M. (1996). An O(K)-Time implementation of fuzzy integral filters on an enhanced mesh processor array, *Proc. IEEE Int. Conf. on Fuzzy Syst.*, 1086-1091.

Simpson, P. (1992). Fuzzy min-max neural networks I: classification, *IEEE Trans. Neural Networks*, 3(5), 776-786.

Sugeno, M. and Tanaka, K. (1991). Successive identification of a fuzzy model and its application to prediction of a complex system, *Fuzzy Sets and Syst.*, 42, 315-334.

Sugeno. M. and Kang, G. T. (1988). Structure identification of fuzzy model, *Fuzzy Sets and Syst.*, 28, 15-33.

Sutton, M. A. and Bezdek, J. C. (1997). Enhancement and analysis of digital mammograms using fuzzy models, *SPIE Proc. AIPR Workshop: Exploiting New Image Sources and Sensors*, 3240, SPIE, Bellingham, WA, 179-190.

Takagi, H. and Hayashi, I. (1991). Artificial neural network driven fuzzy reasoning, *Int J. Appr. Reason.*, 5(3), 191-212.

Tsypkin, Y. Z. (1973). *Foundations of the Theory of Learning Systems*, Translator Z. J. Nikolic, Academic Press, NY.

Wang, D., Wang, X., and Keller, J. M. (1997). Determining fuzzy integral densities using a genetic algorithm for pattern recognition, *Proc. NAFIPS Conf.*, 263-267.

Wang, X., and Keller, J. M. (1997). Fuzzy surroundedness, *Proc. IEEE Int. Congress on Fuzzy Syst.*, 1173-1178.

Wang, X., Keller, J. M., and Gader, P. (1997). Using spatial relationships as features in object recognition, *Proc. NAFIPS Conf.*,, 160-165.

Windham, M. P., Windham, C., Wyse, B. and Hansen, G. (1985). Cluster analysis to improve food classification within commodity groups, *J. Amer. Diet. Assoc.*, 85(10), 1306-1314.

Windham, M.P. (1983). Geometrical fuzzy clustering algorithms, *Fuzzy Sets and Syst.*, 3, 271-280.

Yamakawa, T. and Tomoda, S. (1989). A fuzzy neuron and its application to pattern recognition, *Proc. IFSA Congress*, ed. J. Bezdek, 30-38.

Yan, B. and Keller, J. M. (1991). Conditional fuzzy measures and image segmentation, *Proc. NAFIPS Conf.*, 32-36.

Appendix 1 Acronyms and abbreviations

Short form	Long form
↓ FLVQ	descending fuzzy learning vector quantization
1-nmp	nearest multiple prototype (classifier)
1-np	nearest prototype (classifier)
1-snp	nearest syntactic prototype (classifier)
1-Urnn	1-stage univariate rank nearest neighbor (rule)
1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
4D	four-dimensional
ACE	alternating cluster estimation
ADDC	adaptive distance dynamic clusters
AFC	adaptive fuzzy clustering
AFCE	adaptive fuzzy c-elliptotypes
AFCM	approximate fuzzy c-means
AFCS	adaptive fuzzy c-shells
AFCV	adaptive fuzzy c-varieties
AFEI	average feature evaluation index
ANFIS	adaptive-network-based fuzzy inference system
ANN	artificial neural network
AO	alternating optimization
AP	assignment prototype
APC	armored personnel carrier
ART	adaptive resonance theory
ARTMAP	adaptive resonance theoretic MAP
ATR	automatic target recognition
AVR	average
BBB	branch and bound backtracking
BK	background
BKS	belief knowledge space
BNN	biological neural network
bpa	basic probability assignment
CA	competitive agglomeration
CART	classification and regression trees
CATSCAN	computerized axial tomography scan
CCM	compatible cluster merging
CFAR	constant false alarm rate
CI	centromeric index
CL	competitive learning
CLIPS	C language integrated production system
CM	c-means
CMF	consequent membership function
CNN	computational neural network
COG	center of gravity
COLL	collinear
CP	Chang-Pavlidis (fuzzy decision tree)
CPU	central processing unit
CRE	crisp rule extraction

CS	compact and separated
CSF	cerebro-spinal fluid
CT	computerized tomographic
CV	certainty value
CZCS	coastal zone color scanner
D&C	divide and conquer
D&C-NFCS	divide and conquer-noise fuzzy c-shells
DARPA	defense advanced research procurement agency
DEVLIN	deviation from linearity
DS	Dempster-Shafer
DT	decision tree
dt	decision template
DYNOC	dynamic cluster validation
EBLVQ	extended batch learning vector quantization
EM	expeactation-maximization
EP	end point
ER	equivalence relation
ERIM	Environmental Research Institute of Michigan
F-	fuzzy
FALVQ	fuzzy adaptive learning vector quantization
FAN	fuzzy aggregation network
FANNY	fuzzy analysis
FART	fuzzy adaptive resonance theory
FARTMAP	fuzzy adaptive resonance theory MAP
FC2RS	fuzzy c two-rectangular shells
FCE	fuzzy c-elliptotypes
FCES	fuzzy c-ellipsoidal shells
FCHP	fuzzy c-hyperplanes
FCL	fuzzy c-lines
FCM	fuzzy c-means
FCP	fuzzy c-planes
FCPQS	fuzzy c-plano quadric shells
FCQ	fuzzy c-quadrics
FCQS	fuzzy c-quadric shells
FCRM	fuzzy c-regression models
FCRS	fuzzy c-rectangular shells
FCS	fuzzy c-shells
FCSS	fuzzy c-spherical shells
FCV	fuzzy c-varieties
FDT	fuzzy decision tree
FF	feed Forward
FFBP	feed forward back propagation
FHMM	fuzzy hidden Markov model
FI	fuzzy integral
FIRE	fuzzy inference rules-else
FKCN	fuzzy Kohonen clustering network
FLD	Fisher's linear discriminant
FLIR	forward-looking infrared (radar)
FLVQ	fuzzy learning vector quantization
FM	fuzzy measure

FMLE	fuzzy maximum likelihood estimation
FNC	fuzzy noise clustering
FNM	fuzzy non-metric
FNN	fuzzy neural network
FOSART	fully self-organized ART
FPCM	fuzzy possibilistic c-means
FRC	fuzzy robust clustering
FRED	fuzzy reasoning edge detector
FSART	fuzzy adaptive resonance theory
FTCP	fuzzy trimmed c-prototypes
GA	genetic algorithm
GHT	generalized Hough transform
GK	Gustafson-Kessel
GLVQ	generalized learning vector quanitization
GM	gray matter
GM-2	falsely labeled gray matter
GMD	Gaussian mixture decomposition
GMVE	generalized minimum volume ellipsoid
GPR	ground penetrating radar
GT1	ground truth (type 1)
H-	hard (crisp)
HCM	hard c-means
HF	heterogeneous fuzzy (data)
HFD	heteregeneous fuzzy data
HMM	hidden Markov model
HT	Hough transform
IART	improved adaptive resonance theory
ID3	interactive dichotomizer 3
IDW	inverse distance weighted
IEEE	Institute of Electrical and Electronics Engineers
IO	input-output
IOAC	index of area coverage
ISODATA	iterative self-organizing data analysis
ISOETRP	iterative self-organizing entropy
Iris (?)	we use this if we are not sure what version of Iris was actually used (cf. our remarks in the preface)
k-nn	k-nearest neighbor (rule)
KB	knowledge based
LADAR	Lasar radar
LANDSAT	land satellite (a guess)
LBG	Lloyd-Buzo-Gray
LDC	linear discriminant classifier
LHS	left hand side
LMS	least mean squared (error)
LODARK	low and dark
LOG	logistic discriminant classifier
LOS	linear order statistic
LVQ	learning vector quantization
m-Mrnn	m-stage multivariate rank nearest neighbor (rule)
m-Urnn	m-stage univariate rank nearest neighbor (rule)

M3	modified mountain method
MA	Mamdani-Assilian
MAD	median of absolute deviations
MAJ	majority
MAX	maximum
MCM	mountain clustering method
MFV	mountain function values
MIMO	multiple input, multiple output
MIN	minimum
MINPRAN	minimize the probability of randomness
MISO	multiple input, single output
MLE	maximum likelihood estimation
MLP	multilayered perceptron
MP-RAGE	magetization-prepared rapid gradient echo
MPC	modified partition coefficient
MR	magnetic resonance
MS	multiple sclerosis
MST	minimal spanning tree
MYCIN	shorthand for many drugs such as spectromyacin
NB	naive Bayes
NC	noise clustering
NERFCM	non-Euclidean relational fuzzy c-means
NERHCM	non-Euclidean relational hard c-means
NFCS	noise fuzzy c-shells
NFS	neuro-fuzzy systems
NISP	norm induced shell prototypes
NIST	National Institute of Standards and Technology
nmp	nearest multiple prototype
NN	neural network
np	nearest prototype
OFCM	object (data) fuzzy c-means
OFR	optimized fuzzy rules
OLS	orthogonal least squares
OP	optimized prototypes
OR	oracle
OT	other
OWA	ordered weighted average
P-	possibilistic
p-D	p-dimensional
PAM	partitioning around medoids
PCA	principal components analysis
PCM	possibilistic c-means
PCNN	pusle coupled neural network
PCPQS	possibilistic c-plano quadric shells
PCQS	possibilistic c-quadric shells
PDF	probability density function
PET	positron emission tomography
PMF	premise membership function
PPR	probabilistic product
PRO	product

PSYFRED	psychovisually motivated fuzzy reasoning edge detector
PT	pathology
QCCM	quadric compatible cluster merging
QDC	quadratic discriminant classifier
RACE	relational alternating cluster estimation
RADAR	radio detection and ranging
RBF	radial basis function
RCA	robust competitive agglomeration
RFCM	relational fuzzy c-means
RGB	red, green, blue
RHS	right hand side
RID3	real interactive dichotomizer 3
RMS	root mean square (error)
rnn	rank nearest neighbor
ROC	receiver operating characteristic (curve)
RoFCM	robust fuzzy c-means
SAHN	sequential agglomerative hierarchical nested
SART	simplified adaptive resonance theory
SB	single best
SBM	segmentation-based method
SCM	subtractive clustering method
SCS	soft competition scheme
SEASAT	sea satellite (a guess)
sg	string grammar
sgFCM	string grammar fuzzy c-means
sgHCM	string grammar hard c-means
sgPCM	string grammar possibilistic c-means
SHCM	sequential hard c-means
SIMO	single input, multiple output
SISO	single input, single output
SLP	single layer perceptron
snp	syntactic nearest prototype
SOFM	self-organizing feature map
SRE	soft rule extraction
SRS	soft relaxation scheme
ssFCM	semi-supervised fuzzy c-means (Bensaid et al., 1996a)
ssfcm	semi-supervised fuzzy c-means (Pedrycz, 1985)
TFCM	temporal fuzzy c-means
TS	Takagi-Sugeno
UA	universal approximator
UBD	unsupervised boundary description
Urnn	1-stage univariate rank nearest neighbor (rule)
VGC	validity guided reclustering
VQ	vector quantizer
WGHT	weighted generalized Hough transform
WM	white matter

Appendix 2 The Iris Data: Table I, Fisher (1936)

Iris sestosa				Iris versicolor				Iris virginica			
Sepal Leng.	Sepal Width	Petal Leng.	Petal Width	Sepal Leng.	Sepal Width	Petal Leng.	Petal Width	Sepal Leng.	Sepal Width	Petal Leng.	Petal Width
5.1	3.5	1.4	0.2	7.0	3.2	4.7	1.4	6.3	3.3	6.0	2.5
4.9	3.0	1.4	0.2	6.4	3.2	4.5	1.5	5.8	2.7	5.1	1.9
4.7	3.2	1.3	0.2	6.9	3.1	4.9	1.5	7.1	3.0	5.9	2.1
4.6	3.1	1.5	0.2	5.5	2.3	4.0	1.3	6.3	2.9	5.6	1.8
5.0	3.6	1.4	0.2	6.5	2.8	4.6	1.5	6.5	3.0	5.8	2.2
5.4	3.9	1.7	0.4	5.7	2.8	4.5	1.3	7.6	3.0	6.6	2.1
4.6	3.4	1.4	0.3	6.3	3.3	4.7	1.6	4.9	2.5	4.5	1.7
5.0	3.4	1.5	0.2	4.9	2.4	3.3	1.0	7.3	2.9	6.3	1.8
4.4	2.9	1.4	0.2	6.6	2.9	4.6	1.3	6.7	2.5	5.8	1.8
4.9	3.1	1.5	0.1	5.2	2.7	3.9	1.4	7.2	3.6	6.1	2.5
5.4	3.7	1.5	0.2	5.0	2.0	3.5	1.0	6.5	3.2	5.1	2.0
4.8	3.4	1.6	0.2	5.9	3.0	4.2	1.5	6.4	2.7	5.3	1.9
4.8	3.0	1.4	0.1	6.0	2.2	4.0	1.0	6.8	3.0	5.5	2.1
4.3	3.0	1.1	0.1	6.1	2.9	4.7	1.4	5.7	2.5	5.0	2.0
5.8	4.0	1.2	0.2	5.6	2.9	3.6	1.3	5.8	2.8	5.1	2.4
5.7	4.4	1.5	0.4	6.7	3.1	4.4	1.4	6.4	3.2	5.3	2.3
5.4	3.9	1.3	0.4	5.6	3.0	4.5	1.5	6.5	3.0	5.5	1.8
5.1	3.5	1.4	0.3	5.8	2.7	4.1	1.0	7.7	3.8	6.7	2.2
5.7	3.8	1.7	0.3	6.2	2.2	4.5	1.5	7.7	2.6	6.9	2.3
5.1	3.8	1.5	0.3	5.6	2.5	3.9	1.1	6.0	2.2	5.0	1.5
5.4	3.4	1.7	0.2	5.9	3.2	4.8	1.8	6.9	3.2	5.7	2.3
5.1	3.7	1.5	0.4	6.1	2.8	4.0	1.3	5.6	2.8	4.9	2.0
4.6	3.6	1.0	0.2	6.3	2.5	4.9	1.5	7.7	2.8	6.7	2.0
5.1	3.3	1.7	0.5	6.1	2.8	4.7	1.2	6.3	2.7	4.9	1.8
4.8	3.4	1.9	0.2	6.4	2.9	4.3	1.3	6.7	3.3	5.7	2.1
5.0	3.0	1.6	0.2	6.6	3.0	4.4	1.4	7.2	3.2	6.0	1.8
5.0	3.4	1.6	0.4	6.8	2.8	4.8	1.4	6.2	2.8	4.8	1.8
5.2	3.5	1.5	0.2	6.7	3.0	5.0	1.7	6.1	3.0	4.9	1.8
5.2	3.4	1.4	0.2	6.0	2.9	4.5	1.5	6.4	2.8	5.6	2.1
4.7	3.2	1.6	0.2	5.7	2.6	3.5	1.0	7.2	3.0	5.8	1.6
4.8	3.1	1.6	0.2	5.5	2.4	3.8	1.1	7.4	2.8	6.1	1.9
5.4	3.4	1.5	0.4	5.5	2.4	3.7	1.0	7.9	3.8	6.4	2.0
5.2	4.1	1.5	0.1	5.8	2.7	3.9	1.2	6.4	2.8	5.6	2.2
5.5	4.2	1.4	0.2	6.0	2.7	5.1	1.6	6.3	2.8	5.1	1.5
4.9	3.1	1.5	0.2	5.4	3.0	4.5	1.5	6.1	2.6	5.6	1.4
5.0	3.2	1.2	0.2	6.0	3.4	4.5	1.6	7.7	3.0	6.1	2.3
5.5	3.5	1.3	0.2	6.7	3.1	4.7	1.5	6.3	3.4	5.6	2.4
4.9	3.6	1.4	0.1	6.3	2.3	4.4	1.3	6.4	3.1	5.5	1.8
4.4	3.0	1.3	0.2	5.6	3.0	4.1	1.3	6.0	3.0	4.8	1.8
5.1	3.4	1.5	0.2	5.5	2.5	4.0	1.3	6.9	3.1	5.4	2.1
5.0	3.5	1.3	0.3	5.5	2.6	4.4	1.2	6.7	3.1	5.6	2.4
4.5	2.3	1.3	0.3	6.1	3.0	4.6	1.4	6.9	3.1	5.1	2.3
4.4	3.2	1.3	0.2	5.8	2.6	4.0	1.2	5.8	2.7	5.1	1.9
5.0	3.5	1.6	0.6	5.0	2.3	3.3	1.0	6.8	3.2	5.9	2.3
5.1	3.8	1.9	0.4	5.6	2.7	4.2	1.3	6.7	3.3	5.7	2.5
4.8	3.0	1.4	0.3	5.7	3.0	4.2	1.2	6.7	3.0	5.2	2.3
5.1	3.8	1.6	0.2	5.7	2.9	4.2	1.3	6.3	2.5	5.0	1.9
4.6	3.2	1.4	0.2	6.2	2.9	4.3	1.3	6.5	3.0	5.2	2.0
5.3	3.7	1.5	0.2	5.1	2.5	3.0	1.1	6.2	3.4	5.4	2.3
5.0	3.3	1.4	0.2	5.7	2.8	4.1	1.3	5.9	3.0	5.1	1.8

Index