Analysis and Comparison of Data Clustering Algorithms for Linguistic Division of Fuzzy Variables in a Fuzzy Neural Network.

by

Bart C. Veenendaal

M.Sc. Thesis on Practical training period carried out from April 5, 1993 to October 8, 1993 at the OMRON European Technical Centre in 's Hertogenbosch

Commissioned by: Prof. Dr. Ir. A.C.P.M. Backx
under supervision of: B.W. Grant, M.Sc. (OMRON ETC)
Ir. C. de Bruijn (OMRON ETC)
Ir. H.H. van de Ven (TUE)

date: October 8, 1993

The Department of Electrical Engineering of the Eindhoven University of Technology accepts no responsibility for the contents of M.Sc. Theses or reports on practical training periods.
Abstract

The aim of this project is to analyse and compare different data clustering algorithms to find the number, position and shape of membership functions needed to optimally represent fuzzy variables. This information is used in a fuzzy neural network to create a fuzzy knowledge base for process control.

Five clustering algorithms, the Fuzzy C-Means algorithm, the Possibilistic C-Means algorithm, the Kohonen Clustering Network, the UPGMA algorithm and the ART-2 network are presented and analysed relying on a priori knowledge about the number of clusters. Adaptations of the Fuzzy C-Means algorithm and the Kohonen Clustering Network are also presented which do not rely on a priori knowledge of the number of clusters. The adaptive Fuzzy C-Means is shown to be the most applicable algorithm.
Preface

This project has been carried out at the European Technical Centre (ETC) of Omron Electronics Europe B.V. in ’s Hertogenbosch, the Netherlands. The European Technical Centre is the European Research & Development centre of Omron.

Omron is a multinational company developing PLCs, computer systems for factory automation, information systems and relays, sensors and switches. The three core technologies of Omron are: Computers, Communications and Control. Omron is a recognised leader in the field of fuzzy logic.

At the European Technical Centre, research is done in the field of PLC programming, software- and hardware development and cognitive computing such as Genetic Algorithms, Fuzzy Logic and Neural Networks.

This project has been done in the field of Fuzzy Logic, Neural Networks and Clustering.
Acknowledgements

I would like to thank Prof. Dr. A.J. van der Wal, manager of the European Technical Centre, for giving me the opportunity to do my Masters project at the European Technical Centre in 's Hertogenbosch, the Netherlands.

I would especially like to express my gratitude to Brian W. Grant, research engineer ETC, for proposing this project and for his supervision and guidance. I would also like to thank Ir. Corrie de Bruijn and the other members of ETC for their support.

From the Eindhoven University of Technology, I would like to thank Prof. Dr. Ir. A.C.P.M. Backx for allowing me to graduate under his responsibility and Ir. H.H. van de Ven for his guidance on behalf of the Measurement and Control Section of the Department of Electrical Engineering.

Bart C. Veenendaal

's Hertogenbosch, October 8 1993.
## Contents

Abstract .......................................................... 1

Preface ............................................................. 2

Acknowledgements .................................................. 3

List of Figures ..................................................... 6

1. Introduction ...................................................... 7

2. Fuzzy Logic ...................................................... 8

2.1 Introduction ................................................... 8

2.2 Fuzzy set theory ............................................... 8

2.3 Fuzzy logic control .......................................... 9

2.3.1 Fuzzification ............................................... 9

2.3.2 Fuzzy inference .......................................... 10

2.3.3 Defuzzification .......................................... 11

2.4 Applications ................................................ 12

3. A Fuzzy Neural Network ...................................... 13

3.1 Introduction ................................................ 13

3.2 A neural network ............................................ 13

3.3 A fuzzy neural network ..................................... 14

3.4 Construction of a fuzzy neural network .................. 16

3.4.1 Stage 1: Data clustering ................................ 16

3.4.2 Stage 2: Rule generation ................................ 17

3.4.3 Stage 3: Rule elimination and rule combination ......... 17

3.4.4 Stage 4: Membership fine tuning ....................... 18

3.5 Application strategy and target environment ............. 18

3.6 Extension of the Omron FS-30AT fuzzy inference software package .............................................. 19

3.6.1 Introduction ............................................. 19

3.6.2 The FS-30AT fuzzy inference software package ... 20

3.7 Separation of minerals in a column flotation pilot plant ................................................................. 20

4. Data Clustering .................................................. 21

4.1 Clustering ..................................................... 21

4.2 Classification of clustering methods ...................... 21

4.3 Cluster validity ............................................. 22

4.4 Mean and variance .......................................... 23

5. Analysis of Clustering Algorithms ......................... 24

5.1 Fuzzy C-Means algorithm .................................. 24

5.1.1 Introduction ........................................... 24

5.1.2 Theory .................................................. 24

5.1.3 Adaptive Fuzzy C-Means algorithm .................. 25

5.1.4 Restrictions of the FCM algorithm ................. 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Possibilistic C-Means algorithm</td>
<td>27</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Introduction</td>
<td>27</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Theory</td>
<td>27</td>
</tr>
<tr>
<td>5.3</td>
<td>Kohonen Clustering Network</td>
<td>29</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Introduction</td>
<td>29</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Theory</td>
<td>29</td>
</tr>
<tr>
<td>5.3.3</td>
<td>The learning-rate</td>
<td>30</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Multiple winner update</td>
<td>31</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Adaptive Kohonen clustering network</td>
<td>31</td>
</tr>
<tr>
<td>5.3.6</td>
<td>Drawbacks of the Kohonen clustering network</td>
<td>31</td>
</tr>
<tr>
<td>5.4</td>
<td>UPGMA Algorithm</td>
<td>32</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Introduction</td>
<td>32</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Analysis of the UPGMA algorithm</td>
<td>32</td>
</tr>
<tr>
<td>5.4.3</td>
<td>A basic example</td>
<td>33</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Drawbacks of the UPGMA algorithm</td>
<td>34</td>
</tr>
<tr>
<td>5.5</td>
<td>The analogue Adaptive Resonance Theory network</td>
<td>35</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Introduction</td>
<td>35</td>
</tr>
<tr>
<td>5.5.2</td>
<td>The network</td>
<td>35</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Vigilance test</td>
<td>37</td>
</tr>
<tr>
<td>5.5.4</td>
<td>An alternative structure</td>
<td>38</td>
</tr>
<tr>
<td>6.1</td>
<td>Implementation</td>
<td>40</td>
</tr>
<tr>
<td>6.2</td>
<td>Used data sources</td>
<td>40</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Data from the pilot column flotation process</td>
<td>40</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Data generation program</td>
<td>40</td>
</tr>
<tr>
<td>6.3</td>
<td>Clustering with a fixed number of clusters</td>
<td>41</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Fuzzy C-Means algorithm</td>
<td>41</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Possibilistic C-Means algorithm</td>
<td>42</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Kohonen Clustering Network</td>
<td>42</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Comparison between clustering algorithms</td>
<td>43</td>
</tr>
<tr>
<td>6.4</td>
<td>Adaptive clustering algorithms</td>
<td>45</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Adaptive FCM</td>
<td>45</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Adaptive KCN</td>
<td>47</td>
</tr>
<tr>
<td>7.</td>
<td>Conclusions</td>
<td>48</td>
</tr>
<tr>
<td>8.</td>
<td>Future Work</td>
<td>49</td>
</tr>
<tr>
<td>9.</td>
<td>Bibliography</td>
<td>50</td>
</tr>
<tr>
<td>Appendix</td>
<td>A: The column flotation process</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>B: Logged data of the column flotation process</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>C: Generated data sets</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>D: C-source code of data generation program</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>C-source code of the adaptive FCM clustering algorithm</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>C-source code of KCN clustering algorithm</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>C-source code of PCM clustering algorithm</td>
<td>56</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Fig.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Crisp sets</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Fuzzy sets</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Shapes of membership functions</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Fuzzification</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>Fuzzy inference and defuzzification</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>Model of a neuron with n inputs and n synapses</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>Topology of a fuzzy neural network</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>Flow chart of the construction of a fuzzy neural network</td>
<td>16</td>
</tr>
<tr>
<td>3.4</td>
<td>High activation areas determining the number of MFs</td>
<td>16</td>
</tr>
<tr>
<td>3.5</td>
<td>Rule combination</td>
<td>18</td>
</tr>
<tr>
<td>3.6</td>
<td>Application strategy and target environment</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>Classification of clustering methods</td>
<td>21</td>
</tr>
<tr>
<td>5.1</td>
<td>Example of FCM restriction</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>The Kohonen clustering network</td>
<td>29</td>
</tr>
<tr>
<td>5.3</td>
<td>Sequence of matrices and dendrogram of the UPGMA algorithm</td>
<td>33</td>
</tr>
<tr>
<td>5.4</td>
<td>Topology of ART-2 network</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>An alternative ART-2 network</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Error tolerance against number of epochs</td>
<td>41</td>
</tr>
<tr>
<td>6.2</td>
<td>Adaptive FCM using &quot;feed grade&quot;</td>
<td>46</td>
</tr>
<tr>
<td>6.3</td>
<td>Adaptive FCM using &quot;recovery&quot;</td>
<td>46</td>
</tr>
<tr>
<td>6.4</td>
<td>Adaptive FCM using d500c5.dat</td>
<td>46</td>
</tr>
<tr>
<td>6.5</td>
<td>Adaptive FCM using d1000c5.dat</td>
<td>46</td>
</tr>
<tr>
<td>6.6</td>
<td>Adaptive FCM using d2000c5.dat</td>
<td>46</td>
</tr>
<tr>
<td>6.7</td>
<td>Adaptive FCM using d1000c10.dat</td>
<td>46</td>
</tr>
<tr>
<td>6.8</td>
<td>Performance measures of adaptive KCN</td>
<td>47</td>
</tr>
<tr>
<td>A.1</td>
<td>Column flotation process</td>
<td>53</td>
</tr>
<tr>
<td>A.2</td>
<td>Column flotation general set-up</td>
<td>53</td>
</tr>
</tbody>
</table>
1. Introduction

During the last few years research interest to the combination of fuzzy logic and neural networks has grown [21], [22]. Both can be applied to control systems without specifying a complex mathematical model of the process to be controlled.

Fuzzy Logic Control Systems rely on expert knowledge. This expert knowledge is encoded in terms of a rulebase in which relations between specific process inputs and required controller actions are defined. These process input variables and controller actions are described linguistically, that is to say, rather than being represented by a crisp value, they are fuzzy variables which map crisp inputs to grades of membership in linguistic classes. Each class is defined by a membership function.

The number of membership functions needed to optimally represent each fuzzy variable, the shape and position of these membership functions within the scope (Universe of Discourse) of the fuzzy variable are normally extracted from knowledgeable operators by means of extensive interviews. These membership functions are then iteratively tuned to enhance controller performance. It is desirable to automate the knowledge acquisition process.

Combining fuzzy logic and neural networks will lead to more effective knowledge acquisition; where the knowledge is acquired through the self-learning and adaptive capabilities of neural networks.

A valuable source of training data for neural networks, and sometimes the only available source of information of a process, are measurements made during operation such as temperature, pressure, speed etc. and corresponding controller actions which are logged with respect to time.

A study of data clustering techniques on these logged data extracts information on the number, position and shape of the membership functions needed to optimally represent each process input variable and controller actions. This study will lead to an improved front-end to a neural network based fuzzy logic control and decision system, a fuzzy neural network.
2. Fuzzy Logic

2.1 Introduction

Fuzzy logic is a mathematical theory developed by Prof. Lotfi Zadeh of the University of California at Berkeley. In 1965 he published the article "Fuzzy Sets" in which the basic concepts of fuzzy logic, the fuzzy set theory, were introduced. This article was based upon the work of Jan Lukasiewicz, a Polish logician and philosopher of the University of Warsaw, who invented substructures of fuzzy sets in which values other than true and false were possible in the 1920s and quantum philosopher Max Black, who wrote about proto-fuzzy sets in 1937 [18].

Initially fuzzy logic remained unnoticed by other sciences until in 1970 fuzzy logic found its first application controlling a cement kiln in Denmark. From then the interest in fuzzy logic has increased and nowadays fuzzy logic has been applied to many control problems, mainly in Japan.

Although the word "fuzzy" implies something vague, the theory is sharp and mathematically proven. Professor Zadeh observed that conventional computer logic is not capable of manipulating data representing subjective or vague human ideas. Fuzzy logic makes it possible to determine valid distinctions in data.

Fuzzy logic allows continuous truth values in the range [0,1] which enables vague concepts to be expressed. Reasoning with fuzzy logic stands much closer to human reasoning than to reasoning based on discrete valued logic such as classical bivalent logic.

2.2 Fuzzy Set Theory

Fuzzy logic is based upon fuzzy set theory. With fuzzy set theory it is possible that an item partially belongs to a set whereas in a classical crisp set an item either belongs or doesn't belong to a set. Whereas the boundaries of crisp sets are perpendicular, those of fuzzy sets are gradual. This makes it possible to calculate and to control with linguistic terms.

For instance, describing someone's age with a linguistic quantity like "young", "middle aged" or "old", is difficult using classical crisp sets. Crisp sets are sets in which an element is included or excluded. The boundaries between the different sets are abrupt, see Fig. 2.1, and cause a discontinuity in the transition between "young" and "middle aged". This is arbitrary, "middle aged" is a state which is variable over some years. Making use of fuzzy sets, the crossings are gradual and allow a more gentle transition between the linguistic terms, the human notion is modeled much better, see Fig. 2.2. This figure shows that a subject or item can belong to different classes.

![Fig. 2.1: Crisp sets](image1)

![Fig. 2.2: Fuzzy sets](image2)
Linguistic quantities, also called fuzzy variables, are represented by means of a positive definite, real valued function: the membership function \( \mu(x) \). This function determines the degree of belonging of an element to a fuzzy set and its value is between 0 and 1. At 0, the element has no membership and at 1, it has full membership. Linguistic quantities which describe a linguistic state are called labels. If an element \( x \) belongs to a collection of objects \( X \), the universe of discourse, a fuzzy set \( A \) can be described by its membership function:

\[
A = \{ (x, \mu_A(x)) \mid x \in X \}
\]  

(2.1)

where \( \mu_A(x) \) is the grade of membership of \( x \) in fuzzy set \( A \).

Operations with fuzzy sets are defined via their membership functions. These fuzzy set operations are an extension of operations defined for classical bivalent sets. The following definitions are frequently used:

- The membership function of the intersection \( C \) of two fuzzy sets \( A \) and \( B \), \( C = A \cap B \), is defined by the fuzzy AND operator as:

\[
\mu_C(x) = \min \{ \mu_A(x), \mu_B(x) \} , \ x \in X
\]

(2.2)

- The membership function of the union \( C \) of two fuzzy sets \( A \) and \( B \), \( C = A \cup B \), is defined by the fuzzy OR operator as:

\[
\mu_C(x) = \max \{ \mu_A(x), \mu_B(x) \} , \ x \in X
\]

(2.3)

### 2.3 Fuzzy Logic Control

Fuzzy logic control is the application of fuzzy logic to operate and control processes. Fuzzy logic control can best be applied to non-linear processes. These processes are difficult to control by conventional means.

The fuzzy logic control strategy consist of three steps [8]:

- Fuzzification;
- Fuzzy inference;
- Defuzzification.

#### 2.3.1 Fuzzification

The input and output variables of a process must be formulated in linguistic terms. Therefore it is necessary to transform the input and output quantities into grades of membership in one or more fuzzy sets describing a linguistic state. This is called fuzzification.

Depending on the type and characteristics of the process, the membership functions can have different shapes based on experience or intuition, see Fig. 2.3.

There exists no procedure for determining the optimal shape of the membership function.
The number of membership functions for each input and output variable depends on the system characteristics. The membership functions have to be overlapping for fuzzy logic operations. Fig. 2.4 shows an example of the fuzzification of the variable outside temperature.

![Fig. 2.3: Shapes of membership functions](image)

**2.3.2 Fuzzy inference**

Fuzzy inference is the process that combines the fuzzy input variables with the fuzzy output variables. It is a reasoning method that uses fuzzy theory to express human knowledge in the form of linguistic rules. These rules together form the fuzzy rule base containing the human expert knowledge. These rules are described in a form like:

\[
\text{IF (antecedent } 1) \text{ AND (antecedent } 2) \ldots \text{ AND (antecedent } n) \text{ THEN (consequent } 1) \text{ AND } \ldots \text{ (consequent } m)\]

The antecedents describe one or more conditions and the consequents contain one or more actions. The antecedents of rules correspond directly to grades of membership calculated during the fuzzification. An example of fuzzy inference is visualised in Fig. 2.5.

The grade of the consequents can be determined by selecting the smallest membership grade of the antecedents. The minimum, Eq. (2.2), has to be determined. Selecting the minimum value of the antecedents best reflects the rule's activation strength.

When the conclusions of each rule are derived through fuzzy inference, the fuzzy output is then determined by combining the fuzzy sets of the conclusions for each rule. The maximum, Eq. (2.3), has to be determined: the OR operator. Selecting the maximum allows all of the rules to influence the final outcome.
2.3.3 Defuzzification

The last step of fuzzy logic control is defuzzification. The fuzzy output is converted into a crisp, numerical result. The most common used method for defuzzification is the centre of gravity method (c.o.g.) which calculates the weighted average of the final fuzzy output. With this defuzzification method, the control outputs will depend continuously on the control inputs.

The centre of gravity defuzzification method is given by:

\[
    u_o = \frac{\int u \mu(u) du}{\int \mu(u) du}
\]

(2.4)

where:
- \( u \) : fuzzy output;
- \( u_o \) : crisp, numerical output;
- \( \mu(u) \) : membership function;
- \( U \) : universe.

An other defuzzification method is the maximum height method which uses the maximum fuzzy output as the final result. This method is more often used for classification of patterns. Sometimes singletons are used to simplify the defuzzification process. A singleton is an output membership function represented by a single vertical line.

An example of defuzzification is visualised in Fig. 2.5.

**Fig. 2.5: Fuzzy inference and defuzzification**
Fig. 2.5 shows two rules of a fuzzy rule base. The fuzzy variables X1, X2 and Y are described by three fuzzy sets: small (S), medium (M) and large (L), mapping crisp inputs to grades of membership. Fuzzy inference determines the minimum of the grades of membership of the antecedents in each rule using the AND operator. This is 0.25 in rule 1 and 0.5 in rule 2. The grades of membership of the consequents of each rule are then combined by taking the maximum, using the OR operator. Defuzzification with the centre of gravity defuzzification method determines the numerical output, given by $u_o$.

2.4 Applications

The experience of human operators generally is intrinsically vague (fuzzy). Therefore, it is difficult to set up a control model by conventional means. Fuzzy logic is able to deal with this type of vagueness and models this information in the form of a rule base.

Fuzzy control is best applied to processes that rely heavily on human experience and intuition, and which therefore make it difficult to use conventional methods. Mostly these are highly complex processes often with non-linear relations between input and output variables.

One of the advantages of fuzzy logic control is that one can specify the desired system behaviour without a complex mathematical model. Human expertise is directly used by expressing the knowledge in terms of linguistic rules. The rules are easy to set up and use a natural language. A fuzzy logic controller can also adapt very easily to new situations, but stability is difficult to prove.

Generally the application areas of fuzzy logic can be divided into:

- Expert systems;
- Control;
- Data analysis and pattern recognition.

In all of these applications fuzzy logic works well in tandem with other technologies and may reduce maintenance, power consumption and facilitate man-machine interaction.
3. A Fuzzy Neural Network

3.1 Introduction

Fuzzy logic control systems rely on expert knowledge. This expert knowledge is extracted from knowledgeable operators through extensive interviews and questionnaires [12]. System designers must rely on operator experience. Information gathered in this way may not be optimal. The acquisition of knowledge is a well known problem in the field of knowledge based systems which has attracted much research. This is caused by the following:

- Expert knowledge is often intuitive and therefore difficult to describe;
- Experts can have different control strategies;
- Complex variable interrelationships are difficult to describe;
- Optimal tuning of the membership functions is iterative.

Using neural networks, fuzzy knowledge acquisition can be automated. Combining fuzzy logic and neural networks into an integrated system will lead to more effective knowledge acquisition. Fuzzy knowledge is completely embedded in a neural network, thus creating a system called a fuzzy neural network (FNN). In this network, the human-like reasoning of fuzzy logic control systems is combined with the learning capabilities of a neural network.

3.2 A neural network

Neural networks (NN) are artificial networks based on the present understanding of the human biological nervous system. They are used for cognitive tasks such as learning and optimization. Neural networks are of interest because they offer a computational approach that may prove to be a very effective way of solving certain problems that are difficult to solve by conventional means.

Analogues to the human brain, an artificial neural network forms a connectionist system consisting of many highly connected computational elements or nodes. These computational elements or processing units are called neurons and are often arranged in a layered structure.

The interconnections between the neurons are called synapses. Each synapse is associated with an adaptive weight which can be positive or negative (excitation or inhibition). These weights are adjusted and reach their optimum value after the neural network has been trained.

A biological neuron can easily be modeled by an artificial neuron. An artificial neuron computes a sum of all the weighted input activation signals, subtracts a internal threshold \( \Theta \) and passes the result through a saturating non-linear function. The output activation of a neuron can be described by:

\[
o = f \left( \sum_{k=0}^{n} A_k w_k - \Theta \right)
\]  

(3.1)

A model of a neuron is visualised in Fig. 3.1.
In a neural network the knowledge is distributed over the network nodes and weights. The most important property of neural networks is the ability to learn by adjustment of the synapses. There exist many strategies, or learning algorithms, which iteratively change synaptic weights to encode knowledge in the network [19], [23].

Learning algorithms can be split into two broad categories: supervised learning and unsupervised learning. A neural network based on supervised learning is taught to minimise a certain error function by comparing the actual output of the network with the desired output provided by a supervisor. A neural network based on unsupervised learning constructs internal models that capture regularities in the input variables without additional information.

### 3.3 A fuzzy neural network

There are several ways of embedding fuzzy knowledge in connectionist topologies [11], [17], one such method is described here.

A fuzzy neural network (FNN) is a feedforward multi-layered network in which the basic elements and functions of fuzzy logic control, fuzzy knowledge concepts such as membership functions and a fuzzy rule base, are embedded. It forms a connectionist structure which has the ability to learn. A topology of a fuzzy neural network is given in Fig. 3.2 [17].

The connectionist system consist of five layers. Nodes at layer one are the input nodes representing the input linguistic variables. Each input variable can be described by a certain number of membership functions. These membership functions are represented by layer two nodes, the input MF nodes, according to:

\[
\mu(x) = e^{-\frac{(x-m_j)^2}{2\sigma_j^2}}
\]  

(3.2)

where \(m_j\) and \(\sigma_j\) are the centre and the width of the gaussian membership function \(\mu(x)\). The values of \(m_j\) and \(\sigma_j\) are represented by the synapses between layer one and two.

Nodes at layer three are rule nodes initially describing all possible combinations between the input variable membership functions. They form the fuzzy rule base. The rule nodes perform the fuzzy AND operation, the MIN operator for fuzzy inference, and transmit the result to layer four that performs the fuzzy OR operation, the MAX operator. The link weights between layer...
two and layer three define the rule antecedents and the links between layer three and layer four define the rule consequents. Connections in both these layers have unit weights and the connections between layer three and four are initially fully connected.

Nodes at layer five are output MF nodes representing the linguistic output variable. They determine the desired output signal by performing the centre of gravity defuzzification method (c.o.g.). The output activations of the output MF nodes, $u_i$, are defuzzified into the outputs $o_j$:

$$o_j = \frac{\sum_i (m_{ij} \sigma_{ij} u_i)}{\sum_i \sigma_{ij} u_i}$$

(3.3)

where $m_{ij}$ and $\sigma_{ij}^2$ are, respectively, the centre and the width of the gaussian membership function. The values of $m_{ij}$ and $\sigma_{ij}^2$ are represented by the synapses between layer four and five.

![Diagram of a fuzzy neural network]

**Fig. 3.2: Topology of a fuzzy neural network**

There are two linguistic nodes for each output variable in layer five. One is used for the decision signal (actual output) and the other is used for training data (desired output): reversed mode. During training the nodes of layer four and layer five perform in this reversed mode. In reversed mode layer five acts as an input layer like layer one, only with desired output data as input. The nodes of layer four present the membership functions for each input from layer five. They realise the same function as the nodes of layer two, see Eq. (3.2).

### 3.4 Construction of a fuzzy neural network

The fuzzy neural network is constructed from logged process data in four stages, see Fig. 3.3. Each stage is explained in the following paragraphs.
3.4.1 Stage 1: Data clustering

This stage determines the number, the shape and the position of membership functions needed to optimally represent each fuzzy variable. Independent clustering of each input linguistic variable and output linguistic variable will give the centres $m_i$ and widths $\sigma_i^2$ for each cluster. Those clusters define regions of high activity in the training data sets for each independent variable. An example of a process with one input and one output is presented in Fig. 3.4, showing the key areas. These key areas are also called fuzzy patches [14].

Each cluster will be described by a gaussian membership function covering these areas of high activation as well as possible. Because gaussian membership functions are differentiable and therefore continuous, they enable the use of gradient descent methods such as back propagation. In Fig. 3.3, the data clustering stage will determine the number of nodes and the weights, which are the centres and the variances, between layer 1 and layer 2 and between layer 4 and layer 5.
3.4.2 Stage 2: Rule generation

Using unsupervised competitive learning, the neural network is trained to extract the relevant knowledge rules from logged data. Finding these rules means finding correlations between the input membership functions and the output membership functions. During the rule generation process, layer five and layer four work in reversed mode.

The relevant knowledge rules are determined by competitive learning between the output activations of the rule nodes at layer three, \( o_{i3}(t) \), and the output activations of the nodes at layer four, \( o_{i4}(t) \). The activations at the rule layer are calculated according to the grades of membership of the input variables and are compared with the activations at the output of layer four. If there is a correlation between those two activations, the weight of the link between the \( i \)th rule node of layer three and the \( j \)th output node at layer four is updated according to the strength of the correlations. This is done by a Hebbian-like competitive learning law [19]:

\[
\begin{align*}
    w_{ij}(t+1) &= o_{i4}(t)w_{ij}(t) + o_{i3}(t) \\
                  &= o_{i4}(t)w_{ij}(t) - o_{i3}(t)
\end{align*}
\]  

Initially the network between layer three and four is fully connected which means that every possible rule exists. During the competitive learning process, associations between the output membership functions and the rule nodes are built up. After the competitive learning process the weights of the links between layer three and four will represent strength of correlation.

3.4.3 Stage 3: Rule elimination and rule combination

For each rule node there is a link connected with an output MF node of layer four with a maximum strength. This link is chosen and set to 1, the rest of the links is deleted for that rule node. If all the link weights emanating from a rule node are very small, then the rule node and the corresponding link can be deleted. The result is a collection of rules and the relevant consequences representing the data set.

To reduce the number of rules further, there is a check for redundant rules. The criteria for this rule combination are:

- Rule nodes must have exactly the same consequences (output MF nodes);
- The rules in the selected set all have a common set of input preconditions (input MF nodes);
- The union of the remaining preconditions form a whole set connected to the same input MF nodes.

An example is visualised in Fig. 3.5.

Fig. 3.5 shows the input MF layer, layer two, the rule layer, layer three, and the output MF layer, layer four. The network on the left contains redundant rules. Rule nodes 1,4 and 6 have the same consequents (output MF nodes). They also have the same input MF nodes. And every possible state of the other input MF node \( X_2 \) is represented by the combination of rule nodes. Therefore the three rule nodes can be combined.
3.4.4 Stage 4: Membership fine tuning

The last stage is used to optimally tune the membership functions to improve the performance. Therefore the synapses $m_{ij}$ and $\sigma^2_{ij}$ between layer one and two and between layer four and five are tuned with the backpropagation algorithm. Backpropagation is a supervised learning strategy for multi-layered networks and is an often used training algorithm in neural networks [23].

This algorithm iteratively adapts the synapses between layer one and two and between layer four and five by minimising a system error $E$ which is the sum of the squared differences between the actual and the desired output for all data points. This system error $E$ is given by:

$$E = \frac{1}{2} \sum_{k=1}^{p} \sum_{j=1}^{n} (y_{jk} - d_{jk})^2$$

where $y_{jk}$ : actual output of output node $k$ using data pattern $j$;
$d_{jk}$ : desired output of output node $k$ using data pattern $j$;
$n$ : number of data pattern;
$p$ : number of output nodes at layer five.

A new training set consisting of input data associated with desired output data, forms training pairs. For each training data set the actual output is calculated from layer one to layer five: a forward pass. According to the actual output and a desired output, given by a supervisor, the error function is calculated and this error is back-propagated into the network: a backward pass, allowing the synapses between layer four and five and between layer one and two to be updated. These are the only layers affected by back propagation. After several iterations, this leads to optimal adjustment of the centres $m$ and variances $\sigma^2$ of the input and output membership functions.

3.5 Application strategy and target environment

A fuzzy neural network can be used to ease knowledge acquisition. A global strategy for the application of a fuzzy neural network is visualised in Fig. 3.6.
Fig. 3.6 presents a controller, a process and the proposed fuzzy neural network. The controller can be a human expert, an expert system or a fuzzy controller. Inputs of the controller are a setpoint and process output.

The fuzzy neural network creates a Fuzzy Logic Knowledge Base (FLKB) which then can be used by the fuzzy inference system of a fuzzy controller to control the process. After training the fuzzy neural network can be used to control the process itself. This makes human experts superfluous.

If there already exist some knowledge of the control, this can be used to initialise the fuzzy neural network before training. If no previous knowledge exists the fuzzy neural network can be constructed purely from logged process data.

In the next sections, a target system and a typical application are given of the proposed fuzzy neural network target environment.

3.6 Extension of the Omron FS-30AT fuzzy inference software package

3.6.1 Introduction

To apply fuzzy logic for process control, Omron has developed fuzzy logic control tools which are implemented on a personal computer. It consist of a hardware board FB-30AT and a software package FS-30AT.

The FB-30AT includes an inference board with an Omron digital 12-bit fuzzy processor FP-3000 and a driver software library. This driver software library enables the board to be used by a user program.
The FS-30AT fuzzy inference software package is a support tool for fuzzy inference. With the FS-30AT fuzzy inference software tool it is possible to define the number of membership functions, create the shape of the membership functions and define the fuzzy rules which form the fuzzy rulebase.

Extending the FS-30AT fuzzy inference software tool with a fuzzy neural network will automatically generate the number and the shape of the memberships-functions and create the fuzzy rules. This means automation of the designing stage of the FS-30AT software for the FB-30AT board.

3.6.2 The FS-30AT fuzzy inference software package

The FS-30AT fuzzy inference software package consists of an editor, a simulator, a chipcode translator and a C-header translator. In the FS-30AT editor, the following has to be defined:

- the number of input and output variables;
- the fuzzy rules;
- the shape of the membership functions for each input or output variable.

The FS-30AT editor creates a knowledge file which is translated into an object code file for the FP-3000 processor of the FB-30AT inference board and a C-source code file. Because the FB-30AT board driver functions are written in the C language, a user program in the C language can call them. This makes it easy to call fuzzy logic functions from user programs. For fuzzy logic execution by the FB-30AT, the object code will be down loaded into the FP-3000 processor using the board driver software.

3.7 Separation of minerals in a column flotation pilot plant

A specific control application of the fuzzy neural network is the column flotation ore separation process which is used in mining industries.

At the Centre de Valorização de Recursos Minerais (CVRM) of the Instituto Superior Técnico (IST), the University of Lisbon, Portugal, a feasibility study for ore separation by column flotation has been set up. This study investigates the application of fuzzy logic control for ore separation and is done under the supervision of Prof. M.T.C. Carvalho.

The aim of the column flotation process is to separate minerals in order to create a concentrate, with increased grade and recovery of one or more selected minerals, and a reject (tailing), with the least amount of the selected minerals as possible. A more specific description of the column flotation process can be found in Appendix A.

The column flotation process is an example of a process which is difficult to control. There are cases where the actions of a manipulated variable result in unknown or unquantified changes in process variables. This makes it very difficult, even for an expert, to model the knowledge required to control the process. It is very difficult to find associations between the manipulated variables and the process variables. Therefore the column flotation process might be suitable as a target environment for the proposed fuzzy neural network. The fuzzy neural network will automatically generate the desired membership functions and fuzzy rules to control this process.

From the Instituto Superior Técnico (IST) in Lisbon, preliminary process data are already available and some clustering tests have been carried out.
4. Data Clustering

4.1 Clustering

Clustering is the grouping of similar objects into a number of smaller groups according to likeness or similarity criteria defined for the objects. Other words for clustering are botryology or numerical taxonomy [5], [10].

A standard way of expressing the likeness or similarity is through some dissimilarity measure or index. A common dissimilarity measure between two objects i and j, is the Minkowski metric:

\[ d(i,j) = \sum_{k=1}^{K} [||x_{ik} - x_{jk}||^r]^{1/r}, \quad r \geq 1 \]

(4.1)

where \( d \): distance measure;
\( K \): the number of features for each data point \( x \).

The most important ones are the Manhattan metric, \( r = 1 \) and the Euclidean metric, \( r = 2 \). The Euclidean metric is normally used to compute the distance between different objects. The Euclidean metric is also called the Euclidean distance.

4.2 Classification of clustering methods

There are many clustering methods, which can be classified according to the type of algorithm employed. A global classification of clustering methods is presented in Fig 4.1. The terms are explained below.

<table>
<thead>
<tr>
<th>Clustering methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-exclusive</td>
</tr>
<tr>
<td>(overlapping)</td>
</tr>
<tr>
<td>Extrinsic</td>
</tr>
<tr>
<td>(supervised learning)</td>
</tr>
<tr>
<td>Hierarchical Partitional</td>
</tr>
<tr>
<td>Intrinsic</td>
</tr>
<tr>
<td>(unsupervised)</td>
</tr>
<tr>
<td>Hierarchical Partitional</td>
</tr>
<tr>
<td>Exclusive</td>
</tr>
<tr>
<td>(non-overlapping)</td>
</tr>
<tr>
<td>Extrinsic</td>
</tr>
<tr>
<td>(supervised learning)</td>
</tr>
<tr>
<td>Hierarchical Partitional</td>
</tr>
<tr>
<td>Intrinsic</td>
</tr>
<tr>
<td>(unsupervised)</td>
</tr>
<tr>
<td>Hierarchical Partitional</td>
</tr>
</tbody>
</table>

Fig. 4.1: Classification of clustering methods
Exclusive versus non-exclusive

An item can belong totally to one cluster in an exclusive classification whereas it is possible to belong to many clusters in a non-exclusive classification. The terms nonoverlapping and overlapping are also used respectively for exclusive and non-exclusive.

Extrinsic versus intrinsic

An intrinsic classification groups objects with reference only to the set of objects themselves. While an extrinsic classification utilises external information or labels to form the groups. Intrinsic classification is called unsupervised learning. Extrinsic classification is called supervised learning because the external information or extrinsic labels are considered as information provided by a supervisor.

Hierarchical versus partitional

A hierarchical classification is a clustering method which creates a tree or sequence of groupings from a dissimilarity matrix. The tree representing the objects or items at the leaves is called a dendrogram or phenogram and provides a hierarchical structure. Higher branches represent larger clusters. A partitional classification is a particular partitioning of objects into single groups. It only uses the objects themselves and does not make use of a dissimilarity matrix. The term clustering is associated with a partitional exclusive classification.

All clustering methods are procedures which search through a data set to find clusters in which the data fits well. Usually the algorithm attempts to optimise a numerical criterion or objective function in order to find the right clusters but many algorithms do not explicitly optimise such a criterion.

4.3 Cluster validity

Cluster validity determines the correctness of the clustering method and of the clusters themselves.

Clustering algorithms which use a numerical criterion or objective function to find the clusters assume that the number of clusters existing in the data set is known a priori. The validity of the individual clusters can be based on the number of data points in that particular cluster or based on related topics like variance. When the number of data points in a data set is increased the reliability in the chosen clustering algorithm and in the final clusters will increase.

When the optimal number of clusters is unknown, the clustering algorithm has to be run for different numbers of clusters. The optimal number of clusters is found by optimising a validity (performance) measure. The optimisation of such a validity measure is very difficult, because usually the validity measure is a monotonically decreasing or increasing function of the number of clusters. Therefore the progress of a validity measure has to be examined. This means that a significant point, a certain "knee point" of a performance measure used in the clustering algorithm, has to be identified. That point will determine the optimal number of clusters [15].
4.4 Mean and variance

The two most important quantities describing data belonging to a data set are the mean and variance. The mean presents the average value of the data and is denoted by:

\[ m = \frac{1}{n} \sum_{j=1}^{n} x_j \]  

(4.2)

The variance presents the spread or dispersion of the data and is denoted by:

\[ \sigma^2 = \frac{1}{n} \sum_{j=1}^{n} (x_j - m)^2 \]  

(4.3)

The positive square root of the variance is the standard deviation, denoted by \( \sigma \).
5. Analysis of Clustering Algorithms

5.1 The Fuzzy C-Means algorithm

5.1.1 Introduction

The Fuzzy C-Means algorithm (FCM) is a partitional clustering algorithm. It was initially developed by Dunn [6] and generalized by Bezdek [1]. The Fuzzy C-means algorithm is based on the classical ISODATA algorithm [6] and is also called the Fuzzy Isodata algorithm.

The Fuzzy C-Means algorithm clusters a data set into a number of clusters with similar data according to the grade of membership of each data point to each cluster. The centre, the standard deviation and the number of data points contained can be defined for each cluster.

Assuming that there are c clusters in the data set, the algorithm minimizes an objective function by which the internal distances or intra-cluster distances (between elements in a cluster) are minimized while interdistances or inter-cluster distances (between clusters) are maximised.

5.1.2 Theory

For a finite data set \( x = \{x_1, x_2, x_3, \ldots, x_n\} \) and a predefined number of clusters, \( c \), for this data set, a matrix \( U \), called the Fuzzy C-partition matrix with dimensions \( n \times c \), can be defined. This matrix \( U \) gives the grade of membership of each data point in data set, \( x \), to each of the clusters, \( c \). For each cluster a centre can be defined, these are described by \( V = \{v_1, v_2, v_3, \ldots, v_c\} \).

The grades of membership \( u_{ij} \) of the matrix \( U \) must satisfy the following conditions:

\[
\begin{align*}
    u_{ij} & \in \{0, 1\} ; \quad 1 \leq i \leq c ; \quad 1 \leq j \leq n ; \\
    \sum_{j=1}^{n} u_{ij} & > 0 ; \quad 1 \leq i \leq c ; \\
    \sum_{i=1}^{c} u_{ij} & = 1 ; \quad 1 \leq j \leq n ;
\end{align*}
\]

(5.1) (5.2) (5.3)

The error function which has to be minimized during clustering is given by:

\[
J_m(U, V) = \sum_{i=1}^{c} \sum_{j=1}^{n} (u_{ij})^m (|x_j - v_i|)^2
\]

(5.4)

This squared error criterion is the normalised sum of the distances between each data point of the data set \( x \) and each centre of \( V \).

The weighting coefficient for the grades of membership \( u_{ij}, m \in [1, \infty) \), called the fuzzifier, describes the relevance of the membership grade in the error function. It controls the "fuzziness". If the weighting coefficient \( m=1 \), the Fuzzy C-Means algorithm changes into the Hard C-means algorithm (HCM). For the HCM algorithm the membership grades \( u_{ij} \) are equal to 0 or 1. When \( m \to \infty \), the membership functions are maximally fuzzy. Normally \( m=2 \), which has proven to give good results.
By differentiating the error function $J_m$ with respect to the membership grades $u_{ij}$ while keeping the centres $V$ fixed, the equation for changing the membership grades can be obtained. Similarly by differentiating the error function $J_m$ with respect to the centres $V$ while keeping the membership grades $U$ fixed, the equation for changing the centres can be obtained. The resulting equations are given by:

\[
u_{ij} = \frac{1}{c} \frac{1}{\sum_{k=1}^{c} \left( \frac{1}{\|x_j - v_{ij}\|^2} \right)^{m-1}} \sum_{j=1}^{n} \left( u_{ij} \right)^m x_j
\]

\[
u_j = \frac{1}{\sum_{j=1}^{n} \left( u_{ij} \right)^m} \sum_{j=1}^{n} \left( u_{ij} \right)^m x_j
\]

The derivations of these two equations are given in [6].

The derivations of these two equations are given in [6].

The Fuzzy C-Means algorithm is described in the following sequence:

1. Specify the number of clusters $c$, the weighting coefficient $m$ and the error tolerance $\varepsilon$.
2. Initialise the centres to divide the universe of discourse equally.
3. Initialise the grades of membership.
4. Assign each data point of the data set to the closest cluster centre.
5. Calculate the RMS change in membership grades according to:

\[
\Delta U = \sum_{i=1}^{c} \sum_{j=1}^{n} \left\| u_{ij}^{(t+1)} - u_{ij}^{(t)} \right\|
\]

6. Update cluster centres according to Eq. (5.6).
7. Update membership grades according to Eq. (5.5).
8. Stop when the RMS change in membership grades, defined in step 5, has reached the error tolerance $\varepsilon$, else go to step 4.
9. Determine variance for each cluster.

The variance of each cluster can be calculated according to:

- a cut-off grade of membership which decides to which cluster each data point belongs.
- a minimum distance which decides to which cluster each data point belongs.

5.1.3 Adaptive Fuzzy C-Means clustering algorithm

The determination of the optimal number of clusters is an important issue. For determining the optimal number of clusters a criterion is used. This criterion is presented in Eq. (5.8) [20].

\[
S(c) = \sum_{i=1}^{c} \sum_{j=1}^{n} \left( u_{ij} \right)^m \left( \|x_j - v_{ij}\|^2 - \|v_j - \bar{x}\|^2 \right)
\]
where  
\( n \) : number of data points  
\( c \) : number of clusters  
\( x_j \) : \( j^{th} \) data point  
\( \bar{x} \) : average data point  
\( v_i \) : centre of \( i^{th} \) cluster  
\( u_{ij} \) : grade of \( j^{th} \) data point belonging to \( i^{th} \) cluster  
\( m \) : weight coefficient (usually \( m=2 \))

Eq. (5.8) is an extension of Eq. (5.4). The goal is to minimise the function \( S(c) \) by running the Fuzzy C-Means algorithm for an increasing number of \( c \). For a certain value of \( c \) the function \( S(c) \) will have a local minimum. This minimum determines the optimal number of clusters needed to represent the data set optimally.

In Eq. (5.8) the first term of the right-hand side is the variance of the data in a cluster and the second term is that of the clusters themselves. Therefore optimal clustering is considered to minimise the variance in each cluster and to maximise the variance between the clusters.

### 5.1.4 Restrictions of the Fuzzy C-Means algorithm

The Fuzzy C-means algorithm uses the probabilistic constraint that the grades of membership of a data point sum up to 1, see Eq. (5.3). The membership grades resulting from FCM and its derivatives do not always correspond to the intuitive concept of degree of belonging. The FCM algorithm does not perform well in noisy environments [16].

The following simple example illustrates the problem associated with the probabilistic constraint used in the FCM algorithm as related to the interpretation of the resulting grades of membership. Fig. 5.1 shows a situation with two clusters defining all data points in the universe of discourse.

![Diagram](Fig. 5.1: Example of FCM restriction)

The FCM algorithm would produce different grades of membership for the data points \( a \) and \( b \), even though they have the same distance to the cluster centre \( m_1 \). This is due to the fact that data point \( b \) also has a grade of membership in cluster 2. Similarly, data point \( a \) and \( c \) may have equal grades of membership in cluster 1, even though data point \( c \) is closer to the cluster centre. In other words, in the FCM algorithm the grade of membership of a data point to a cluster is a relative number and depends on the grades of membership of the data point to other clusters.

Thus the grades of membership are unrepresentative of the degree of belonging and do not distinguish between a data point close to the centre of the cluster and a data point far from the centre of the cluster. Noise points, often quite distant from the primary clusters, can drastically influence the clustering due to a relatively high grade of membership. This is caused by the fact that the sum of the membership grades for each point must sum up to 1.
5.2 The Possibilistic C-Means algorithm

5.2.1 Introduction

A way to improve the performance of the FCM algorithm in the presence of noise is to adjust the FCM algorithm to make the Possibilistic C-Means algorithm (PCM). This algorithm places the clustering problem into the framework of possibility theory and does not have the restrictions of the Fuzzy C-Means algorithm, namely the property that the grades of membership of a data point must sum up to 1, described in section 5.1.4. The algorithm describes the data set with possibility distributions instead of membership functions.

5.2.2 Theory

The grades of membership are interpreted as degrees of possibility of the data points belonging to the clusters. The grade of membership \( u_{ij} \) is now defined as the degree of compatibility of the data point \( x_j \) to cluster \( i \) or the possibility of data point \( x_j \) belonging to cluster \( i \). To form the possibilistic partition matrix \( U \), the conditions for the fuzzy partition matrix \( U \) of the FCM algorithm are changed to:

\[
0 < \sum_{j=1}^{n} u_{ij} < 1 ; \quad 1 \leq i \leq c ; \quad 1 \leq j \leq n ;
\]  

\[
0 < \sum_{i=1}^{c} u_{ij} < n ; \quad 1 \leq j \leq n ;
\]  

\[
\sum_{i=1}^{c} u_{ij} > 0 ; \quad 1 \leq j \leq n ;
\]

The aim is to have grades of membership for representative data points to be as high as possible while unrepresentative data points should have low grades of membership in all clusters. An objective function which satisfies these requirements can be formulated as:

\[
J_m(V, U) = \sum_{i=1}^{c} \sum_{j=1}^{n} (u_{ij})^m (\|x_j - v_i\|^2) + \sum_{i=1}^{c} \sum_{j=1}^{n} (1-u_{ij})^m
\]

This function has to be minimised. The first term insinuates that the distances from the data points to the cluster centres be as low as possible, whereas the second term forces the grades of membership \( u_{ij} \) to be as large as possible.

The value of \( \eta_i \) determines the distance at which the grade of membership of a data point in a cluster becomes 0.5 (3 dB point). \( \eta_i \) determines a certain bandwidth for the possibility distribution for each cluster. It also determines the relevance of the second term in the objective function compared with the first term. \( \eta_i \) can be defined as:

\[
\eta_i = \frac{\sum_{j=1}^{n} (u_{ij})^m (\|x_j - v_i\|^2)}{\sum_{j=1}^{n} (u_{ij})^m}
\]
The following rule may also be used:

\[ \eta_i = \frac{K \sum_{x_j \in [\Pi_j]_\alpha} (\|x_j - v_i\|^2) \left\| \frac{\|x_j - v_i\|^2}{\eta_i} \right\|}{| [\Pi_j]_\alpha |} \]  

(5.14)

where \((\Pi_j)_{\alpha}\) is an appropriate \(\alpha\)-cut of \(\Pi_j\), the possibility distribution function. \(K\) is chosen to be 1. In this case, \(\eta_i\) is the average intracluster distance for data points with a grade of membership greater than or equal to \(\alpha\).

The change in grades of membership is given by a possibility distribution, satisfying the conditions defined in Eqs. (5.9), (5.10) and (5.11):

\[ u_{ij} = \frac{1}{1 + \left( \frac{(\|x_j - v_i\|^2)}{\eta_i} \right)^{\frac{1}{m-1}}} \]  

(5.15)

In each iteration, the updated value of the grade of membership \(u_{ij}\) depends only on the distance of the data point \(x_j\) to cluster \(i\).

The grades of membership of a data point to a cluster should be solely determined by the distance of the data point to the cluster centre and should not depend on its location with respect to other cluster centres. This allows optimal grades of membership solutions for the entire universe of discourse and implies that the grades of membership are not restricted to the FCM algorithm rule that the sum of all grades of membership for a single data point should be equal to 1.

Noise points will have low degrees of compatibility in all clusters thus neglecting their influence.

The change of cluster centres will proceed exactly the same as in the FCM algorithm, Eq. (5.6).

The **Possibilistic C-Means algorithm** is described by the following sequence [15]:

-1- Specify the number of clusters, the weighting coefficient and the error tolerance \(\varepsilon\).
-2- Initialise the centres to divide the universe of discourse equally.
-3- Initialise the grades of membership making use of the FCM algorithm.
-4- Estimate approximate values for \(\eta_i\) using Eq. (5.13).
-5- Assign an input to the closest cluster centre.
-6- Calculate the change in membership grades according to:

\[ \Delta U = \sum_{i=1}^{c} \sum_{j=1}^{n} \| u_{ij}^{(t+1)} - u_{ij}^{(t)} \| \]  

(5.16)

-7- Update the cluster centres according to Eq. (5.6).
-8- Update the membership grades, according to Eq. (5.15).
-9- Stop when the change in membership grades, defined in step 6, has reached the error tolerance \(\varepsilon\), else go to step 5.
-10- Re-estimate more accurate values for \(\eta_i\) using Eq. (5.14) and start again at step 5 (second run of the algorithm).
-11- Determine the variance for each cluster.

The second run of the algorithm converges very quickly and is used to find more precise centres.
5.3 The Kohonen Clustering Network

5.3.1 Introduction

The Kohonen Clustering Network (KCN) is an unsupervised, partitional clustering algorithm developed by Prof. Teuvo Kohonen from the Helsinki University of Technology in Finland in 1988. This neural network tries to classify or cluster data points by self-organising the data. Data with features in common are clustered using a competitive learning strategy.

5.3.2 Theory

The Kohonen Clustering Network is a feedforward network consisting of two layers, an input (fan-out) layer and an output (competitive) layer, see Fig. 5.2. The input layer consist of one input node, for one dimensional inputs, and the output layer of c fixed output nodes representing the clusters. Between the two layers there exists a fully connected network with a synaptic weight for each connection. When an input is presented at the input node, it is compared with the weights belonging to the output nodes. The output nodes compete among themselves to determine the "winner", the output node whose synaptic weight is closest to the input. Thus the network decides to which cluster the input belongs most. This competition between the output nodes is called competitive learning. Only the winner updates its weight according to the distance between its weight and the input. This single winner update or "winner-take-all" learning is given by:

$$w_{\text{winner}}^{t+1} = w_{\text{winner}}^t + LR \cdot (x_k - w_{\text{winner}})$$

(5.17)

where

- $LR$: the learning rate;
- $x_k$: input;
- winner: number of winning output node;
- $w_{\text{winner}}$: weight of winning output node.

This process continues until all weights stabilise and converge. The learning is completed and the weights represent the cluster centres. This is called self-organising. Generally it can be said that the Kohonen clustering network tries to organise the weights in such a way, that topologically related nodes are sensitive to inputs that are physically similar.

![Fig. 5.2: The Kohonen Clustering Network](image-url)
The Kohonen clustering algorithm is given in the following sequence [1]:

-1- Define the number of output nodes/clusters $c$ and an error criterion $\varepsilon$.
-2- Initialise the weights (=centres) equally spreading the input space.
-3- Initialise the learning rate $L.R$.
-4- Compute the euclidean distance measure for each output node with all data points.

$$d_i = (x(t) - w_i(t))^2 ; \quad i=1..c$$  \hspace{1cm} (5.18)

-5- For each data point $x$ select the output node with the minimum distance and adapt the weights to this node (single winner) or update the winner and its neighbours, the nodes which are in the neighbourhood of this winning node (multiple winner).

-6- Compute the square of the difference of the weights and the previous weights:

$$E = \sum_{i=0}^{c-1} \| w[i] - w_{old}[i] \|^2$$  \hspace{1cm} (5.19)

-7- If $E \leq \varepsilon$ then go to step 8, else adjust learning rate and start from step 4, another epoch.

-8- Define mean and variance of each output node/cluster.

5.3.3 The learning rate

In Eq. (5.17), the learning rate is used in updating the weights. To force the weights to converge, the learning rate has to be reduced during clustering to produce finer adjustments. The decrease of the learning rate can be linear or exponential. The most used forms are:

$$LR = \eta \times \left(1 - \frac{t}{\alpha}\right)$$  \hspace{1cm} (5.20)

$$LR = \eta \times \exp\left(-\frac{t}{\beta}\right)$$  \hspace{1cm} (5.21)

where $t$ : integer, starting from 0 and increasing after every epoch (step 4-7);
$\eta$ : initial value of learning rate;
$\alpha, \beta$ : constants.

An other possibility for the decrease of the learning rate is to start the decrease after a fixed number of epochs:

$$\begin{cases}
LR = \eta \times \left(1 - \frac{(t-\gamma)}{\alpha}\right) \\
LR = \eta \quad \text{for } t<\gamma
\end{cases}$$  \hspace{1cm} (5.22)

$$\begin{cases}
LR = \eta \times \exp\left(-\frac{t-\gamma}{\beta}\right) \\
LR = \eta \quad \text{for } t<\gamma
\end{cases}$$  \hspace{1cm} (5.23)

where $\gamma$ : integer, determining the number of epochs after which the learning rate will decrease.
5.3.4 Multiple winner update

To prevent some output units never becoming active (because all input patterns are not evenly divided in input space), not only the winning output node updates its weights but also the nodes in the neighbourhood of the winning output node. This multiple winner update rule replaces Eq. (5.17):

\[
\mathbf{w}[i]_{t+1} = \mathbf{w}[i]_t + \mathbf{L} \mathbf{R} \ast \frac{|x_k-w[\text{winner}]|}{|x_k-w[i]_t|} \ast (x_k-w[\text{winner}])
\] (5.24)

Another multiple winner update rule is given by:

\[
\mathbf{w}[i]_{t+1} = \mathbf{w}[i]_t + \mathbf{L} \mathbf{R} \ast \left(1 - \frac{\sum_{i=0}^{c} |x_k-w[i]_t|}{|x_k-w[\text{winner}]|} \right) \ast \frac{|x_k-w[\text{winner}]|}{(x_k-w[i]_t)}
\] (5.25)

5.3.5 Adaptive Kohonen clustering network

To find the optimal number of clusters the Kohonen clustering algorithm should be run for several different values of \(c\). After each clustering, the variance and the number of data points have to be calculated to help decide which number of clusters is best.

Thus the optimal number of clusters can be defined by:

- A given maximum standard deviation \(\sigma\) for every cluster;
- A given minimum number of data points for every cluster;
- A certain mathematical criterion.

Another way of determining the optimal number of clusters is finding a local or global minimum or a "knee point" of a certain validity (performance) measure. Examples of a validity (performance) measure are:

- The number of epochs needed to cluster the data set for every value of \(c\);
- The cumulated distance of each data point to its nearest cluster;
- A certain mathematical criterion.

5.3.6 Drawbacks of the Kohonen clustering network

There are some drawbacks on the Kohonen clustering network:

- The Kohonen clustering network does not optimise a criterion or function, but it finds the cluster centres by using a decreasing learning rate. This means that the algorithm is terminated artificially. So even though the weights reach their final value, it is possible that the weights have not converged close enough to the real final values representing the cluster centres.
- Because the weights are directly updated after every presentation of an input, the final values of the weights, the cluster centres, are dependent on the sequence of this input. This makes the algorithm order dependent.
- The decreasing learning rate function must be varied from one set to another, it is dependent on the data set.
5.4 The UPGMA clustering algorithm

5.4.1 Introduction

The UPGMA algorithm is a sequential, agglomerative, hierarchic, non-overlapping clustering method. UPGMA stands for Unweighted Pair Group using Metric Averages.

It computes the average similarity or dissimilarity between two data points or clusters of data points, weighting each data point or cluster of data points equally.

The UPGMA algorithm is a linkage algorithm which creates a tree of clusters. The visualisation of such a tree is called a phenogram or a dendrogram.

5.4.2 Analysis of the UPGMA clustering algorithm

The algorithm starts with a quantitative dissimilarity matrix \( D \) with the dissimilarities between all data points. The dissimilarity matrix \( D \) is given by:

\[
D = d[(i),(j)] ; \quad 0 < i < n \quad \land \quad 0 < j < n ;
\]  

(5.26)

The smaller the indexes, the smaller the distance between two data points.

Clusters are then created by a merging process. The algorithm produces a nested sequence of clusters and a value or level for each cluster after linking. This value or level increases during the linking of clusters and is a measure for the dissimilarity between the linked clusters.

The UPGMA algorithm is described by the following sequence:

-1- Initially each pattern is assigned to a unique cluster. So initially the number of clusters is equal to the number of data points.

-2- Find the smallest entry in the dissimilarity matrix \( D \).

\[
d(i, j) = \min_{(s, t)} [d(s, t)]
\]  

(5.27)

-3- Merge cluster \((i)\) and \((j)\) into a new cluster to establish clustering.

-4- Updating the dissimilarity matrix \( D \) by deleting the row and column corresponding to one of the designated clusters, e.g. \((i)\). The entries in the row and column corresponding to the other designated cluster, denoted \((i,j)\) are given according to:

\[
d[(k),(i,j)] \leftarrow \alpha_i d[(k),(i)] + \alpha_j d[(k),(j)] ; \quad k \neq i \land k \neq j ;
\]  

(5.28)

\[
\alpha_i = \frac{n_i}{n_i + n_j} ; \quad \alpha_j = \frac{n_j}{n_i + n_j} ;
\]  

(5.29)

The dissimilarity between cluster \((k)\) and the newly formed cluster is a weighted average that depends on the number of data points \(n_i\) and \(n_j\) in the merged clusters \((i)\) and \((j)\).

-5- Repeat linking clusters, which means going to step 2, until the value or level of the linked clusters have reached a predefined cut-level (a maximum dissimilarity). This predefined cut-level determines the maximum value or level for each linked cluster which is a measurement for the dissimilarity between the two merged clusters.
A derivation of the UPGMA algorithm is the WPGMA algorithm which stands for Weighted Pair Group using Metric Averages. It differs from the UPGMA algorithm by weighting the member most recently admitted to a cluster equal with all previous members.

5.4.3 A basic example

In Fig. 5.3 an example of the UPGMA algorithm is given [5]. The sequence of dissimilarity matrices and the corresponding dendrogram is visualised.

Fig. 5.3: Sequence of matrices and dendrogram of the UPGMA algorithm

The first minimum in the dissimilarity matrix D is 0.2 between point 1 and 3. Point 1 and point 3 are merged into a new cluster and the dissimilarity matrix is updated by deleting the row and column corresponding to point 1 and point 3. The new elements, the entries of the new dissimilarity matrix, \( d[(2),(1,3)], d[(1,3),(4)], d[(1,3),(5)] \) and \( d[(1,3),(6)] \) are calculated according to Eqs. (5.26) and (5.27).

As example the new entry \( d[(2),(1,3)] \) is calculated according to:

\[
d[(2),(1,3)] = \frac{n_1}{n_1 + n_3} \cdot d[(2,1)] + \frac{n_2}{n_1 + n_3} \cdot d[(2,3)] = \frac{1}{2} \cdot 0.6 + \frac{1}{2} \cdot 0.7 = 0.65
\]
Defining a cut-level in the dendrogram will determine the number of clusters. In Fig 5.3, a cut-level of the value 4 will result in 2 clusters.

5.4.4 Drawbacks of the UPGMA algorithm

A drawback of the UPGMA algorithm are its memory requirements. The UPGMA algorithm stores $n(n-1)/2$ data points to form the dissimilarity matrix. This will limit the application of the UPGMA algorithm to small data sets.
5.5 The analogue Adaptive Resonance Theory network

5.5.1 Introduction

The analogue Adaptive Resonance Theory network (ART-2) has been introduced by G.A. Carpenter and S. Grossberg in 1987. Both are working at the "Centre for Adaptive Systems" (CAS) of the Boston University.

The ART-2 network is a neural network which classifies or clusters analogue input patterns. The number of categories will be determined during the classification. This network is able to pick out and enhance similar patterns embedded in noisy backgrounds [2], [3].

5.5.2 The network

The ART-2 network consists of a number of interconnected arrays. These arrays are divided into two different fields. A "feature representation" field denoted by F1 representing the arrays $w, x, y, u, p, q$ of dimension $N$, the number of features of an input pattern and a "category representation" field denoted by F2 representing array $y$ of dimension $M$, the number of categories. Both fields are representative for Short Term Memory (STM).

The two fields are connected by means of a bottom-up (BU) and a top-down (TD) filter, representative of Long Term Memory (LTM). The fields F1 and F2, with the bottom-up (BU) and the top-down (TD) filters are called the "attentional subsystem". Besides the attentional subsystem there is a "orienting subsystem" which consists of a reset mechanism for checking the correctness of each classification.

An ART-2 topology is presented in Fig. 5.4.

![Topology of ART-2 network](image)

**Fig. 5.4: Topology of ART-2 network**

A general description of the operation of an ART-2 network is described as follows: An input pattern, $i$, of dimension $N$, is presented at the bottom of the F1 layer. This input pattern will undergo some non-linear transfer functions and normalisations of the F1 layer. These non-linearities and normalisations will enhance salient features and suppress noise.
They are given by the following F1 layer formulas:

\[ w_i = \hat{i}_i + a \cdot u_i \quad (5.30) \]

\[ x_i = \frac{w_i}{e + \|w\|} \quad (5.31) \]

\[ v_i = f(x_i) + b \cdot f(q_i) \quad (5.32) \]

\[ u_i = \frac{v_i}{e + \|x\|} \quad (5.33) \]

\[ p_i = u_i + \sum_{j=1}^{M} g(y_j) \cdot z_{ji} \quad (5.34) \]

\[ q_i = \frac{p_i}{e + \|p\|} \quad (5.35) \]

where:
- \( i \): dimension of an input pattern: \( 1 \leq i \leq N \);
- \( j \): number of output nodes (categories): \( 1 \leq j \leq M \);
- \( a \): constant determining the relationship between \( i \) and \( y \);
- \( b \): constant determining the relevance of maxima in the input pattern;
- \( e \): constant determining the relevance of the normalisation;
- \( z_{ji} \): elements of the TD filter.

The non-linear function \( f(x) \) can be described by:

\[ f(x) = \begin{cases} 
0, & 0 \leq x \leq \Theta \\
x, & x \geq \Theta 
\end{cases} \quad (5.36) \]

The function \( f(x) \) only passes positive values that exceed \( \Theta \). \( \Theta \) influences the non-linearities and determines the contrast enhancement. If \( \Theta = 0 \), there is no non-linearity.

After these non-linearities and normalisations, the resulting vector \( p \) stabilises and will be presented to the BU filter. This filter connects the result, \( p \), of the F1 layer with \( M \) output nodes, \( y \), of the F2 layer representing \( M \) categories. For every output node \( y \), the sum of each feature of array \( p \) multiplied with the corresponding elements of the BU filter, the summed filtered input to the \( j^{th} \) F2 node, \( T_j \), will be calculated:

\[ T_j = \sum_{i} p_i \cdot z_{ij}, \quad \forall j, \quad 0 \leq j \leq M-1 \quad (5.37) \]

where \( z_{ij} \) are the elements of the BU filter.

The \( j^{th} \) output node \( y \), the "winner", which becomes maximally active, will be chosen as the best category for the input pattern:

\[ T_j = \max\{T_j : j = 0 \ldots M-1\} \quad (5.39) \]

From this output node the TD calculation will started.

A reference pattern, a prototype, of that particular category which is stored in the TD weights, \( z_{ji} \),
will be presented to array \( p \) of the F1 layer making use of a function \( g \), defined by:

\[
g(y_j) = \begin{cases} 
    d & \text{if } T_j = \max\{T_i\} \\
    0 & \text{else}
\end{cases}, \quad 0 < d < 1
\]  

(5.39)

where \( d \) is a constant which determines the importance of a reference pattern from the LTM.

This means that equation (5.34) reduces to:

\[
p_i = \begin{cases} 
    u_i & \text{if } F2 \text{ is inactive} \\
    u_i + d_z_j & \text{if } y_J \text{ is active}
\end{cases}
\]  

(5.40)

Again a new input pattern is presented at the F1 layer and after a few iterations a new pattern is presented at array \( p \). This pattern is now influenced by the reference pattern of layer F2. A comparison is made between array \( p \) and \( y \) by a vigilance test [7].

5.5.3 The vigilance test

The vigilance test checks whether a pattern belongs to the last selected category or not. The orienting subsystem performs this vigilance test and generates a RESET vector of dimension \( M \) according to the \( N \) dimensional vector \( r \) of the orienting subsystem. The elements of vector \( r \) are calculated according:

\[
r_j = \frac{u_i + c \cdot p_j}{e + \| u \| + \| c \cdot z \|}
\]  

(5.41)

where \( c \) determines the relevance between \( u \) and \( p \).

The orienting subsystem will determine the \( j^{th} \) element of RESET. RESET\(_j\)=1 if:

\[
\frac{\rho}{e + \| z \|} > 1, \quad 0 \leq \rho \leq 1
\]  

(5.42)

where \( \rho \) is the vigilance parameter.

If category \( J \) is not cancelled (this means that RESET\(_j\)=0), the elements of the BU and TD filter are adjusted. This adjustment is described by learning rules.

For the BU filter to winning node \( J \):

\[
\frac{d}{dt} z_{j,i} = d[p_{j,i} - z_{j,i}] = d(1-d) \left[ \frac{u_i}{1-d} - z_{j,i} \right]
\]  

(5.43)

For the TD filter from winning node \( J \):

\[
\frac{d}{dt} z_{j,i} = d[p_{j,i} - z_{j,i}] = d(1-d) \left[ \frac{u_i}{1-d} - z_{j,i} \right]
\]  

(5.44)

If category \( J \) is cancelled (RESET\(_j\)=1), a category (output node) has to be associated with the input pattern and the vigilance test will be executed again. This continues until an uncommitted node
(category), this is a node which has never been active before, is activated. The node is then said to be committed. After that, a new input pattern can be applied to the Fl layer and the classification starts again.

ART-2 networks are self-organising and classify patterns according to the prototypes stored in the TD filter elements. The weights will be adjusted step by step so that the influence of old patterns does not disappear.

According to [3] there are three criteria for initialising of the ART-2 network.

- The initial values for the elements of the TD filter have to be set equal to:
  \[ z_{ji}(0) = 0 \quad \forall i, j \]  
  (5.45)

- The values of the constants c and d have to satisfy:
  \[ \frac{c \cdot d}{1 - d} \leq 1 \]  
  (5.46)

- The initial values for the elements of the BU filter have to be set to:
  \[ z_{ij}(0) = \frac{1}{(1 - d) \sqrt{N}} \quad \forall i, j \]  
  (5.47)

5.5.4 An alternative network architecture

From the Division of Medical Electrical Engineering of the Department of Electrical Engineering of the Eindhoven University of Technology, a simulation program of an ART-2 network was available. Originally this program was developed by two students of Carpenter & Grossberg in 1990. The simulation program is based on an alternative ART-2 structure [7], given in Fig. 5.5.

In this alternative ART-2 network, the Fl layer is extended with an "preprocessing" field F0. This extra layer serves as a buffer to isolate the input lines from the top-down activity of the network, and to normalize the input pattern. Fl performs further normalisation and integrates the bottom-up (BU) and top-down (TD) influences.

The F0 field equations are the same as the Fl field Eqs. (5.30)-(5.36).

The vector \( \mathbf{r} \) of the orienting system changes into:

\[ \mathbf{r}_i = \frac{\mathbf{p}_{0,i} + c \cdot \mathbf{p}_{1,i}}{\mathbf{e} + \| \mathbf{p}_0 \| + \| c \cdot \mathbf{p}_1 \|} \]  
(5.48)
Fig. 5.5: An alternative ART-2 network
6. Testing of Clustering Algorithms

6.1 Implementation of clustering algorithms

To test the different clustering algorithms, they were implemented in the C language. Listings of the C-source code of implemented clustering algorithms are given in Appendix D.

6.2 Used data sources

The clustering algorithms are tested with data sets for the construction of stage 1 of the proposed fuzzy neural network. The following data sources are used:

- Data of preliminary tests of the pilot column flotation process of the Instituto Superior Técnico of the University of Lisbon, Portugal;
- Data generated by a data generation computer program, which generates data in a specified number of clusters.

6.2.1 Data from the pilot column flotation process

Data are logged from preliminary tests of a pilot column flotation process of the Instituto Superior Técnico (IST) of the University of Lisbon in Portugal. The logged data consist of seven data sets each of 47 data patterns for process variables and manipulated variables in the column flotation process, see Appendix B. Each of these seven data sets has already been clustered using the UPGMA algorithm at the IST. The results can be used to make comparisons between different clustering algorithms implemented for this study.

6.2.2 Data generation program

The logged data sets of the column flotation process have some restrictions. The number of data points in each data set is small and the data sets do not have non-overlapping clusters. Therefore data sets with more data points and overlapping clusters are needed for the validation of the clustering algorithms. These data sets are created with a data generation program. This program generates random data points according to a predefined number of clusters, with for each cluster a given centre, \( m \), and standard deviation, \( \sigma \) [4].

In this program two random numbers \( U_1 \) and \( U_2 \) are chosen randomly, ranging \([0,1]\). An independent normal random number with zero mean and unit variance is then computed according to:

\[
X = (-2lnU_1)^{1/2} \cos(2\pi U_2)
\]  

(6.1)

The set of random data points can be generated according to:

\[
Y = m + \sigma X
\]

(6.2)

where \( m \) and \( \sigma \) have to be defined for each cluster.

The generated data files are ASCII formatted files. The data sets generated by the data generation program are given in Appendix C.
6.3 Clustering with a fixed number of clusters

The described clustering algorithms are tested with the mentioned data sources with a predefined fixed number of clusters $c$. The validation of the results of a clustering algorithm is difficult, it depends on the a priori knowledge of each data set.

In the next paragraphs the word "epoch" refers to the execution of the main loop of the relating clustering algorithm. In every epoch all data points are presented to the algorithm.

6.3.1 Fuzzy C-Means algorithm

The Fuzzy C-Means algorithm is tested with both data sources. To find relations between the error tolerance and the number of epochs for large data sets, the FCM algorithm is run with three generated data sets. The results are given in Table 6.1 and visualised in Fig. 6.1.

<table>
<thead>
<tr>
<th>error tolerance</th>
<th>d500c5.dat</th>
<th>d1000c5.dat</th>
<th>d2000c5.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>0.1</td>
<td>30</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>0.01</td>
<td>58</td>
<td>28</td>
<td>20</td>
</tr>
<tr>
<td>0.001</td>
<td>84</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>0.0001</td>
<td>107</td>
<td>44</td>
<td>34</td>
</tr>
</tbody>
</table>

Fig. 6.1 shows that the Fuzzy C-Means algorithm is dependent on the number of data. For bigger data sets the number of epochs will decrease. This seems logical, in every epoch the adjustment of the centres depends on the number of data points. The more data points the more precise the adjustment of the centres in one epoch and the faster the convergence.
6.3.2 Possibilistic C-Means algorithm

As mentioned in the analysis of the Possibilistic C-Means, the algorithm starts with an initialisation of the grades of membership by running the Fuzzy C-Means algorithm for a number of epochs until a certain error tolerance is reached. The value of this error tolerance is different for each data set and should not be too small because then the possibilistic clustering would be unnecessary.

6.3.3 Kohonen Clustering Network

Two versions of the KCN are implemented: a "single winner update" version and a "multiple winner update" version. If the centres are more or less equally dividing the universe of discourse, the "single winner update" version is recommended, if not, the "multiple winner version" should be used.

The KCN algorithm is influenced by the initial value of the learning rate and the decrease of the learning rate after every epoch. Initial clustering showed that artificial termination is necessary for convergence. This means that the learning rate must decrease after each epoch for a finer adjustment of the centres, else the centres oscillate. The learning rate will not be decreased after each update of the centres because this would mean that each data point is not of equal importance.

To check the dependence of the Kohonen clustering network on data ordering, the algorithm is tested with the data set feed grade in reversed order. The results are compared with the clustering of the normal data set. This is displayed in Table 6.2.

The initial learning rate is set at 0.01 and the decrease of the learning rate is defined by:

\[ LR = 0.01 \cdot \exp(-t/10) \]  

where \( t \) is a counter.

An exponential decrease forces the algorithm to converge faster than a linear decrease. If the decrease is too big, the accuracy will reduce. The update of the KCN will be of the "multiple winner" type.

<table>
<thead>
<tr>
<th>data set</th>
<th>epochs</th>
<th>cluster</th>
<th>centre ( m )</th>
<th>stand. dev. ( \sigma )</th>
<th># data points</th>
</tr>
</thead>
<tbody>
<tr>
<td>feed grade</td>
<td>43</td>
<td>1</td>
<td>5.6536</td>
<td>0.3552</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7.2406</td>
<td>0.2497</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>17.9272</td>
<td>0.4675</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>24.3964</td>
<td>0.4756</td>
<td>8</td>
</tr>
<tr>
<td>feed grade (reversed)</td>
<td>45</td>
<td>1</td>
<td>5.6798</td>
<td>0.3600</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7.2621</td>
<td>0.2461</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>17.9381</td>
<td>0.4671</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>24.3387</td>
<td>0.4846</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 6.2 shows that the Kohonen clustering network is dependent on the sequence of presentation of the data points to network. This is caused by the fact that the centres are adjusted after
presenting a data point. In every epoch the data is presented to the network in the same way. This 
order dependence can be neglected by presenting the data points randomly to the network. This is 
realised by storing the entire data set in an array, from which each data point randomly will be 
chosen. If a data point is chosen, its place in the array will be filled with the last data point of the 
array until all data points are chosen.

6.3.4 Comparison between clustering algorithms

A comparison is made between the five different clustering algorithms, described in part 5. First 
the clustering algorithms are tested with the data sets of the column flotation process. Table 6.2 
gives the clustering results of the data set feed grade.

The results of the UPGMA algorithm came from the IST of the University of Portugal. For the 
other clustering algorithms, the error tolerance is set at 0.0001 and the number of clusters is set at 
4. The PCM algorithm is first initialised with the FCM algorithm with an error tolerance of 0.1.

<table>
<thead>
<tr>
<th>Table 6.3: Results of clustering algorithms with data set &quot;feed grade&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>data set: feed grade</strong></td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td><strong>FCM clustering</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>PCM clustering</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>KCN clustering</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>UPGMA clustering</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Simulations with ART-2 are done with the ART-2 programs of two students of Carpenter and 
Grossberg from the Centre of Adaptive Systems at the Boston University. They appeared not to be 
suitable for clustering one-dimensional data. Because of the normalisation of the input in the F1-
layer all one dimensional inputs were classified into one cluster. ART-2 tries to classify certain 
patterns and has to be used for multi-dimensional data points.

Table 6.3 shows that the results of the four clustering algorithms are all in the same direction. It 
is difficult to say which one is best. The data set feed grade is a small data set consisting of four 
distinct clusters which are non-overlapping. The four algorithms perform almost the same.
The second data set of the column flotation process which is clustered is data set recovery. This data set consist of data more equally spread and shows some overlap. The results are presented in Table 6.4. The number of clusters is fixed at seven. A correct initialisation of the grades of membership for the PCM algorithm seemed not to be possible.

Table 6.4: Results of clustering algorithms with data set "recovery"

<table>
<thead>
<tr>
<th>data set: recovery</th>
<th>cluster</th>
<th>mean m</th>
<th>stand. dev. σ</th>
<th># data points</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCM clustering</td>
<td>1</td>
<td>18.0768</td>
<td>4.5716</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>30.4455</td>
<td>1.8928</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>41.3573</td>
<td>3.1340</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>55.0912</td>
<td>3.7735</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>67.1599</td>
<td>2.8177</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>74.4538</td>
<td>2.4818</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>84.8039</td>
<td>5.3709</td>
<td>12</td>
</tr>
<tr>
<td>KCN clustering</td>
<td>1</td>
<td>18.4232</td>
<td>4.5642</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32.8618</td>
<td>3.7312</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>44.4422</td>
<td>2.7992</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>55.0677</td>
<td>2.8229</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>66.5226</td>
<td>2.7300</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>74.3593</td>
<td>2.4809</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>85.7651</td>
<td>5.3043</td>
<td>12</td>
</tr>
<tr>
<td>UPGMA clustering</td>
<td>1</td>
<td>11.03</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25.485</td>
<td>5.5655</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>39.842</td>
<td>3.1681</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>53.920</td>
<td>3.9295</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>70.8061</td>
<td>4.8510</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>83.422</td>
<td>1.9988</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>96.780</td>
<td>2.1070</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.4 shows that the FCM and the KCN algorithm perform in a similar way. Differences between the two are caused by some overlap in the data set. The UPGMA algorithm shows the tendency to cluster outliers separately.

Further testing is done with the FCM and the KCN algorithm. With the data generation program, data sets with overlapping clusters are generated. The results of the clustering are displayed in Table 6.5 and Table 6.6.

The update of the KCN algorithm is of the "single winner update" type and the learning rate function is defined as in Eq. (6.3).

The possibilistic C-Means algorithm is not further tested. For large data sets the FCM initialisation was already very accurate and no real improvement was noticed.
Table 6.5: Results of clustering with data set d500c5.dat

<table>
<thead>
<tr>
<th>cluster</th>
<th>centre m</th>
<th>stand. dev. σ</th>
<th>data points</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCM clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10.7892</td>
<td>4.4911</td>
<td>104</td>
</tr>
<tr>
<td>2</td>
<td>25.7817</td>
<td>4.1082</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>40.4119</td>
<td>4.3756</td>
<td>95</td>
</tr>
<tr>
<td>4</td>
<td>54.8529</td>
<td>4.2443</td>
<td>109</td>
</tr>
<tr>
<td>5</td>
<td>70.3909</td>
<td>3.9780</td>
<td>105</td>
</tr>
<tr>
<td>KCN clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11.6518</td>
<td>4.6568</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>27.1978</td>
<td>4.0344</td>
<td>89</td>
</tr>
<tr>
<td>3</td>
<td>41.8362</td>
<td>4.2563</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>55.5194</td>
<td>3.9150</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>70.1962</td>
<td>3.9215</td>
<td>104</td>
</tr>
</tbody>
</table>

Table 6.6: Results of clustering with data set d1000c5.dat

<table>
<thead>
<tr>
<th>cluster</th>
<th>centre m</th>
<th>stand. dev. σ</th>
<th>data points</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCM clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8.7540</td>
<td>3.9659</td>
<td>189</td>
</tr>
<tr>
<td>2</td>
<td>23.2385</td>
<td>3.9840</td>
<td>195</td>
</tr>
<tr>
<td>3</td>
<td>38.8073</td>
<td>4.0165</td>
<td>237</td>
</tr>
<tr>
<td>4</td>
<td>54.4313</td>
<td>4.1428</td>
<td>202</td>
</tr>
<tr>
<td>5</td>
<td>69.4362</td>
<td>4.2109</td>
<td>177</td>
</tr>
<tr>
<td>KCN clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9.0837</td>
<td>4.0505</td>
<td>193</td>
</tr>
<tr>
<td>2</td>
<td>23.2817</td>
<td>3.8901</td>
<td>191</td>
</tr>
<tr>
<td>3</td>
<td>38.5959</td>
<td>3.9438</td>
<td>234</td>
</tr>
<tr>
<td>4</td>
<td>53.6813</td>
<td>4.0389</td>
<td>199</td>
</tr>
<tr>
<td>5</td>
<td>68.8502</td>
<td>4.3428</td>
<td>183</td>
</tr>
</tbody>
</table>

Both tables show resemblance between the FCM algorithm and the KCN algorithm. It is difficult to verify or validate each algorithm. According to the generated data sets, the FCM and KCN algorithm cluster the data set well. The resulted standard deviations are smaller.

For large data sets, the FCM algorithm becomes slow because the number of membership grades which have to be updated after every epoch, becomes very large.

6.4 Adaptive clustering

6.4.1 Adaptive FCM

According to criterion of Eq. (5.8), the optimal number of clusters c is determined. The results of tests with data from the column flotation process are given in Fig. 6.2 and Fig. 6.3. The criterion function S(c) of clustering the generated data sets is visualised in Fig. 6.4 - Fig. 6.7.
For all six tests an error tolerance of 0.0001 is reached.

The minimum of criterion $S(c)$ determines the optimal number of clusters. The optimal number of cluster is circled. For the data sets feed grade and recovery an optimal number of clusters is found. The algorithm also finds the correct number of clusters of the generated data sets.
6.4.2 Adaptive KCN

To determine the optimal number of clusters with the KCN algorithm, the algorithm has to be run for several different values of $c$. For each clustering, analysis of variances or the number of data points can decide the optimal number of clusters $c$. Another way to determine the optimal clusters is to find a minimum or a "knee point" of a certain validity (performance) measures. The following performance measures are investigated for the generated data sets. The investigated performance measures for determining the number of clusters are:

- The cumulated distances of each data point to the nearest cluster centre:

$$ S_1(c) = \sum_{i=1}^{c} \sum_{j=1}^{n} \| x_j - v_{\text{winner}} \|^2 $$

(6.4)

- The cumulated distances of each data point to all the cluster centres:

$$ S_2(c) = \sum_{i=1}^{c} \sum_{j=1}^{n} \| x_j - v_i \|^2 $$

(6.5)

- The criterion used for the adaptive FCM algorithm:

$$ S_3(c) = \sum_{i=1}^{c} \sum_{j=1}^{n} (\| x_j - v_{\text{winner}} \|^2 - \| v_{\text{winner}} - x \|^2) $$

(6.7)

These performance measures are calculated to determine the optimal number of clusters by finding a minimum or a clear "knee point". The measures are visualised in Fig. 6.8. All are normalised on one scale. The four performance measures all monotonically decrease or increase and do not show local minima or maxima or a certain "knee point".

![Fig. 6.8: Performance measures of adaptive KCN](image)
7. Conclusions

In this thesis a fuzzy neural network is described that automates knowledge acquisition for knowledge base systems. The knowledge acquisition is automated by extraction of logged data from a process instead of extraction of knowledge from experts by means of extensive interviews.

Different clustering algorithms have been analysed and tested for the construction of stage 1 of the fuzzy neural network, the determination of the optimal number, shape and position of membership functions.

**Clustering with a fixed number of clusters**

- **Fuzzy C-Means algorithm:**
  The number of epochs in the FCM algorithm decreases with increasing number of data points.

- **Possibilistic C-Means algorithm:**
  This algorithm is an extension of the FCM algorithm. The FCM initialisation of the grades of membership depend on the data set used.

- **Kohonen Clustering Network:**
  The KCN algorithm is shown to be dependent on the sequence of the data points. This is cancelled by presenting the data points randomly to the network. The choices of initial learning rate and the decay of the learning rate depend on the data set.

- **UPGMA algorithm:**
  The UPGMA algorithm is useful for small non-overlapping data sets. For large data sets the algorithm requires much memory and the number of computations for updating the entries of the dissimilarity matrix will be huge.

- **ART-2 network:**
  The ART-2 network seems not to be applicable for clustering one-dimensional data points.

In case of data sets with distinct clusters which are non-overlapping the FCM, PCM, KCN and the UPGMA algorithm perform equivalently. In this case the KCN is the fastest algorithm.

In case of data sets with overlapping clusters the FCM and the KCN algorithm perform similarly. For large data sets like d500c5.dat and d1000c5.dat, the FCM algorithm becomes slow. The KCN algorithm can be adjusted by changing the learning rate. This makes a faster convergence possible but this change depends on the data set.

**Adaptive clustering**

A major improvement of the data clustering stage is automatic determination of the number of clusters without prior knowledge of the number of membership functions needed to represent each fuzzy variable.

The implemented adaptive FCM algorithm is able to find the optimal number of clusters in data sets with non-overlapping and overlapping clusters by minimising an objective function. This objective function minimises the variance in each cluster and maximises the variance between the clusters. The position and shape are represented by the centres $m$ and the variances $\sigma^2$ of the clusters.
For the KCN algorithm different validity (performance) measures are investigated to determine the optimal number of clusters. Because the KCN is not based on optimising any model or objective function or criterion, the investigated performance measures show no minima or "knee points". Therefore the KCN algorithm is not able to dynamically determine the optimal number of clusters.

The adaptive Fuzzy C-Means algorithm was chosen to be implemented as the front-end to the fuzzy neural network described in section 3. Although the Fuzzy C-Means algorithm is relatively slow for larger data sets, the nature of this type of learning means that for any given process the algorithm need only be executed once to generate the membership functions. The extra time required is therefore a small price to pay for the accuracy and the intelligent choice of an optimal number of membership functions that the adaptive FCM algorithm provides.

8. Future work

After this study the following suggestions can be made:

- Other intelligent clustering methods such as the Leader clustering algorithm should be investigated;

- The FCM and the PCM algorithm should be tested in noisy environments;

- The ART-2 network may be useful for stage 2: rule generation;

- Integration and testing on real processes;

- Better validation techniques should be investigated;
Bibliography

Fuzzy Kohonen clustering networks.

Clustering characterization of adaptive resonance.


On the optimal choice of parameters in a Fuzzy C-Means algorithm.

Clustering methodologies in exploratory data analysis.

A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters.

The adaptive resonance theory network: (clustering-) behaviour in relation with brainstem auditory evoked potential patterns.

[8] Gier, W.A.
Fuzzy control.

[9] Grant, B.W.
An implementation of the Fuzzy C-Means clustering algorithm.

[10] Hartigan, J.A.
Clustering algorithms.

Composition methods of fuzzy neural networks.
A prototype of neuro-fuzzy cooperation system.

Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence.

Fuzzy logic.

[15] Krishnapuram, R,
Fuzzy clustering methods in computer vision.

A possibilistic approach to clustering.

Neural-network-based fuzzy logic control and decision system.

Fuzzy Logic: The discovery of a revolutionary computer technology.

[19] Simpson, P.K.
Artificial Neural Systems.

Fuzzy-logic-based approach to qualitative modelling.

[21] Takagi, H.

[22] Zadeh, L.

[23] Zurada, J.M.
Introduction to artificial neural systems.
Appendix A: The column flotation process

After grinding and chemical conditioning the ore is feed into a flotation column at about 2/3 of the total height of the column as a pulp. This column is about six metres high. In the pre-conditioning phase the minerals required are coated to make them hydrophobic (rejected by water) and the reject minerals are made hydrophillic (attracted to water).

Air bubbles are introduced at the base of the column which the hydrophobic minerals use for transportation to the top of the column. The hydrophillic minerals are attracted to water and sink to the bottom of the column and are rejected. At the top of the column a water spray operates to free the small amount of hydrophillic minerals which have been adhered to the air bubbles or have been trapped in the froth of air and hydrophobic minerals. They will fall down to the bottom of the column.

So inside the column two zones can be distinguished: the collection zone and the cleaning or froth zone. In the collection zone the particles descending will collide with the air bubbles and hydrophobic particles will adhere to the air bubbles and clime to the froth zone where they constitute the concentrate.

The column flotation process is presented in Fig. A.1. Fig. A.2. shows the general set-up of the column.

In the column flotation process two types of variables can be distinguished:

**Process variables:**

- Feed grade
- Froth level
- Air flow
- Recovery
- Concentrate grade

**Manipulated variables:**

- Feed rate
- Froth dosage
- Percentage solids in concentrate
- Tailings rate
- Air rate
- Wash water rate
Fig. A.1: Column flotation process

Fig. A.2: Column flotation general set-up
## Appendix B: Logged data of column flotation process

<table>
<thead>
<tr>
<th>number</th>
<th>Feed grade</th>
<th>Froth level</th>
<th>Feed rate</th>
<th>Air rate</th>
<th>Recovery grade</th>
<th>Concentrate Weight</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17.89</td>
<td>20</td>
<td>0.3</td>
<td>0.16</td>
<td>69.82</td>
<td>22.67</td>
<td>55.12</td>
</tr>
<tr>
<td>2</td>
<td>17.58</td>
<td>20</td>
<td>0.3</td>
<td>0.42</td>
<td>41.84</td>
<td>22.55</td>
<td>32.43</td>
</tr>
<tr>
<td>3</td>
<td>18.08</td>
<td>30</td>
<td>0.64</td>
<td>0.16</td>
<td>73.28</td>
<td>22.13</td>
<td>59.61</td>
</tr>
<tr>
<td>4</td>
<td>18.29</td>
<td>30</td>
<td>0.64</td>
<td>0.03</td>
<td>78</td>
<td>22.71</td>
<td>62.74</td>
</tr>
<tr>
<td>5</td>
<td>18.83</td>
<td>40</td>
<td>1</td>
<td>0.09</td>
<td>84.72</td>
<td>21.22</td>
<td>74.97</td>
</tr>
<tr>
<td>6</td>
<td>17.56</td>
<td>40</td>
<td>1</td>
<td>0.3</td>
<td>73.13</td>
<td>20.48</td>
<td>62.97</td>
</tr>
<tr>
<td>7</td>
<td>18.15</td>
<td>50</td>
<td>1.38</td>
<td>0.09</td>
<td>85.81</td>
<td>20.54</td>
<td>76.36</td>
</tr>
<tr>
<td>8</td>
<td>17.24</td>
<td>50</td>
<td>1.38</td>
<td>0.3</td>
<td>74.35</td>
<td>20.61</td>
<td>62.85</td>
</tr>
<tr>
<td>9</td>
<td>23.64</td>
<td>20</td>
<td>0.3</td>
<td>0.09</td>
<td>58.2</td>
<td>28.75</td>
<td>47.92</td>
</tr>
<tr>
<td>10</td>
<td>24.14</td>
<td>20</td>
<td>0.3</td>
<td>0.23</td>
<td>53.56</td>
<td>28.95</td>
<td>44.85</td>
</tr>
<tr>
<td>11</td>
<td>24.31</td>
<td>30</td>
<td>0.64</td>
<td>0.09</td>
<td>69.6</td>
<td>28.6</td>
<td>59.18</td>
</tr>
<tr>
<td>12</td>
<td>23.95</td>
<td>30</td>
<td>0.64</td>
<td>0.23</td>
<td>48.34</td>
<td>28.63</td>
<td>40.73</td>
</tr>
<tr>
<td>13</td>
<td>24.7</td>
<td>40</td>
<td>1</td>
<td>0.09</td>
<td>71.91</td>
<td>28.1</td>
<td>63.55</td>
</tr>
<tr>
<td>14</td>
<td>24.86</td>
<td>40</td>
<td>1</td>
<td>0.23</td>
<td>62.54</td>
<td>27.47</td>
<td>56.1</td>
</tr>
<tr>
<td>15</td>
<td>24.97</td>
<td>50</td>
<td>1.37</td>
<td>0.09</td>
<td>72.83</td>
<td>28.19</td>
<td>64.08</td>
</tr>
<tr>
<td>16</td>
<td>24.97</td>
<td>50</td>
<td>1.37</td>
<td>0.23</td>
<td>67.17</td>
<td>28.27</td>
<td>59.03</td>
</tr>
<tr>
<td>17</td>
<td>5.75</td>
<td>30</td>
<td>0.3</td>
<td>0.16</td>
<td>79.06</td>
<td>23.07</td>
<td>19.71</td>
</tr>
<tr>
<td>18</td>
<td>5.64</td>
<td>30</td>
<td>0.64</td>
<td>0.16</td>
<td>82.32</td>
<td>20.61</td>
<td>22.45</td>
</tr>
<tr>
<td>19</td>
<td>5.63</td>
<td>30</td>
<td>1</td>
<td>0.16</td>
<td>81.54</td>
<td>20.73</td>
<td>22.08</td>
</tr>
<tr>
<td>20</td>
<td>5.68</td>
<td>30</td>
<td>1.38</td>
<td>0.16</td>
<td>83.41</td>
<td>14.65</td>
<td>32.45</td>
</tr>
<tr>
<td>21</td>
<td>5.65</td>
<td>30</td>
<td>0.3</td>
<td>0.16</td>
<td>86.82</td>
<td>18.58</td>
<td>26.38</td>
</tr>
<tr>
<td>22</td>
<td>5.54</td>
<td>30</td>
<td>0.64</td>
<td>0.16</td>
<td>98.27</td>
<td>8.5</td>
<td>64.37</td>
</tr>
<tr>
<td>23</td>
<td>5.86</td>
<td>50</td>
<td>1</td>
<td>0.16</td>
<td>95.29</td>
<td>9.16</td>
<td>60.37</td>
</tr>
<tr>
<td>24</td>
<td>6.02</td>
<td>20</td>
<td>0.64</td>
<td>0.09</td>
<td>67.22</td>
<td>19.92</td>
<td>20.3</td>
</tr>
<tr>
<td>25</td>
<td>5.78</td>
<td>20</td>
<td>0.64</td>
<td>0.09</td>
<td>68.38</td>
<td>20.69</td>
<td>19.35</td>
</tr>
<tr>
<td>26</td>
<td>5.75</td>
<td>30</td>
<td>0.64</td>
<td>0.09</td>
<td>63.97</td>
<td>22.43</td>
<td>16.27</td>
</tr>
<tr>
<td>27</td>
<td>5.48</td>
<td>30</td>
<td>0.64</td>
<td>0.23</td>
<td>54.84</td>
<td>25.86</td>
<td>11.62</td>
</tr>
<tr>
<td>28</td>
<td>5.37</td>
<td>40</td>
<td>0.64</td>
<td>0.23</td>
<td>50.65</td>
<td>25.82</td>
<td>10.53</td>
</tr>
<tr>
<td>29</td>
<td>6.14</td>
<td>40</td>
<td>0.64</td>
<td>0.23</td>
<td>37.67</td>
<td>25.13</td>
<td>9.17</td>
</tr>
<tr>
<td>30</td>
<td>4.67</td>
<td>50</td>
<td>1.38</td>
<td>0.23</td>
<td>43.2</td>
<td>18.65</td>
<td>10.85</td>
</tr>
<tr>
<td>31</td>
<td>5.06</td>
<td>50</td>
<td>1</td>
<td>0.09</td>
<td>62.98</td>
<td>19.37</td>
<td>16.4</td>
</tr>
<tr>
<td>32</td>
<td>7.29</td>
<td>20</td>
<td>0.47</td>
<td>0.16</td>
<td>36.66</td>
<td>29.59</td>
<td>9.03</td>
</tr>
<tr>
<td>33</td>
<td>7.44</td>
<td>20</td>
<td>0.47</td>
<td>0.16</td>
<td>23.25</td>
<td>29.53</td>
<td>6.01</td>
</tr>
<tr>
<td>34</td>
<td>7.51</td>
<td>40</td>
<td>0.82</td>
<td>0.16</td>
<td>31.33</td>
<td>28.41</td>
<td>8.28</td>
</tr>
<tr>
<td>35</td>
<td>7.52</td>
<td>40</td>
<td>0.82</td>
<td>0.43</td>
<td>20.74</td>
<td>31.33</td>
<td>4.98</td>
</tr>
<tr>
<td>36</td>
<td>7.87</td>
<td>40</td>
<td>1.19</td>
<td>0.16</td>
<td>27.54</td>
<td>28.22</td>
<td>7.68</td>
</tr>
<tr>
<td>37</td>
<td>7.34</td>
<td>40</td>
<td>1.19</td>
<td>0.43</td>
<td>18.37</td>
<td>31.25</td>
<td>4.31</td>
</tr>
<tr>
<td>38</td>
<td>7.24</td>
<td>40</td>
<td>1.58</td>
<td>0.16</td>
<td>11.03</td>
<td>30.64</td>
<td>2.61</td>
</tr>
<tr>
<td>39</td>
<td>7.53</td>
<td>40</td>
<td>1.58</td>
<td>0.43</td>
<td>31.68</td>
<td>28.8</td>
<td>8.28</td>
</tr>
<tr>
<td>40</td>
<td>7.16</td>
<td>20</td>
<td>0.47</td>
<td>0.16</td>
<td>57.93</td>
<td>13.2</td>
<td>31.44</td>
</tr>
<tr>
<td>41</td>
<td>7.18</td>
<td>20</td>
<td>0.47</td>
<td>0.03</td>
<td>70.92</td>
<td>11.42</td>
<td>44.01</td>
</tr>
<tr>
<td>42</td>
<td>7.41</td>
<td>30</td>
<td>0.82</td>
<td>0.03</td>
<td>74.16</td>
<td>11.5</td>
<td>46.63</td>
</tr>
<tr>
<td>43</td>
<td>7.08</td>
<td>30</td>
<td>0.82</td>
<td>0.16</td>
<td>76.18</td>
<td>11.1</td>
<td>48.35</td>
</tr>
<tr>
<td>44</td>
<td>6.93</td>
<td>50</td>
<td>1.37</td>
<td>0.03</td>
<td>80.8</td>
<td>10.23</td>
<td>54.07</td>
</tr>
<tr>
<td>45</td>
<td>7.28</td>
<td>50</td>
<td>1.37</td>
<td>0.16</td>
<td>84.08</td>
<td>9.39</td>
<td>65.2</td>
</tr>
<tr>
<td>46</td>
<td>7.03</td>
<td>60</td>
<td>1.58</td>
<td>0.16</td>
<td>81.2</td>
<td>8.88</td>
<td>64.17</td>
</tr>
<tr>
<td>47</td>
<td>6.91</td>
<td>60</td>
<td>1.58</td>
<td>0.03</td>
<td>83.59</td>
<td>9.04</td>
<td>63.97</td>
</tr>
</tbody>
</table>
Appendix C: Generated data sets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Data Points</th>
<th>Number of Clusters</th>
<th>Centres</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>d500c5.dat</td>
<td>500</td>
<td>5</td>
<td>$m_1=10$, $m_2=25$, $m_3=40$, $m_4=55$, $m_5=70$</td>
<td>$\sigma_1=5$, $\sigma_2=5$, $\sigma_3=5$, $\sigma_4=5$, $\sigma_5=5$</td>
</tr>
<tr>
<td>d1000c5.dat</td>
<td>1000</td>
<td>5</td>
<td>$m_1=10$, $m_2=25$, $m_3=40$, $m_4=55$, $m_5=70$</td>
<td>$\sigma_1=5$, $\sigma_2=5$, $\sigma_3=5$, $\sigma_4=5$, $\sigma_5=5$</td>
</tr>
<tr>
<td>d2000c5.dat</td>
<td>2000</td>
<td>5</td>
<td>$m_1=10$, $m_2=25$, $m_3=40$, $m_4=55$, $m_5=70$</td>
<td>$\sigma_1=5$, $\sigma_2=5$, $\sigma_3=5$, $\sigma_4=5$, $\sigma_5=5$</td>
</tr>
<tr>
<td>d1000c10.dat</td>
<td>1000</td>
<td>10</td>
<td>$m=5, 8, 11, 14, 17, 20, 23, 26, 29, 32$</td>
<td>$\sigma=1$ for $m=1..10$</td>
</tr>
</tbody>
</table>

These 3 data sets are generated separately and are therefore not dependent (data set d1000c5.dat is not two times data set d500c5.dat).
DESCRIPTION: Generating of random data for a given number of clusters with predefined centres and sigmas. This data can be used for testing data clustering algorithms.

AUTHOR: Bart C. Veenendaal

LANGUAGE: Borland C++ version 3.1

DATE: 03-06-93

COPYRIGHT: OMRON-ETC-NL 's Hertogenbosch

#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <alloc.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

float

#define SUCCESS 0 /* return on successful completion */
#define FAILURE -1 /* return on unsuccessful completion */
#define PI 3.14159265359 /* the number pi */
#define FNAMESIZE 40 /* maximum characters of filename */

void Header(void);
void InputData(void);

void Header()
{
    clrscr();
    printf("RANDOM DATA GENERATOR \n");
    printf("------------------------------------------ \n");
}

void InputData()
{
    int i; /* loop counter */

    printf("input # of clusters: \t");
    scanf("%d", &c);
    fflush(stdin);
    printf("input # of datapoints: \t");
    scanf("%d", &d);
    fflush(stdin);
    printf("\n");

    if ((centre = calloc(c, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in < MAIN >\n");
        exit(FAILURE);
    }

    if ((sigma = calloc(c, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in < MAIN >\n");
        exit(FAILURE);
    }

    for (i=0;i<c;i++)
    {
        printf("input the centre and sigma for cluster \%d \t\n", i);
        scanf("%f", &centre[i]);
        scanf("%f", &sigma[i]);
        fflush(stdin);
    }
}
```c
/* -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* MAIN */
/* -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
int main()
{
    FILE *datafile; /* created datafile */
    char fname[FNAMESIZE]; /* name of output file */
    int j, /* loopcounter */
    clusternumber; /* number of chosen cluster */
    float u1,u2, /* uniform random numbers */
    x, /* normal random number with zero mean and unit variance */
    y; /* created datapoint */

    // Header();
    printf("Input name of output file, including extension: \t");
    scanf("%s", fname);
    fflush(stdin);

    // InputData();
    if ((datafile = fopen(fname, "w")) == NULL)
    {
        printf("Error opening datafile \n");
        exit(FAILURE);
    }
    // randomize();
    for (j = 0; j < dj + 1)
    {
        u1 = (float)random(RAND_MAX) / RAND_MAX;
        u2 = (float)random(RAND_MAX) / RAND_MAX;
        x = sqrt(2*log(u1))*cos(2*PI*u2); /* calculation of random number */
        clusternumber = random(c);
        y = centre[clusternumber] + sigma[clusternumber]*x;
        fprintf(datafile, "%f
", y);
    }
    fclose(datafile);
    return(SUCCESS);
}```
FILE: AdaptFCM.c


AUTHOR: B.C. Veenendaal

LANGUAGE: Borland C++ version 3.1

DATE: 09-07-93

COPYRIGHT: OMRON ETC's Hertogenbosch

09-07-93 Criterion calculation for determination of the optimal number of clusters

11-06-93 Adjustment of variance calculation

25-11-92 First working version of FCM Clustering algorithm by BWG

---

/* Header Files */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <alloc.h>
#include <time.h>
#include <math.h>
#include <values.h>

/* Global Constants */
#define SUCCESS 0
#define FAILURE -1
#define DOSFNAMESIZE 40

/* return on successful completion */
/* return on unsuccessful completion */
/* maximum chars in DOS filename */

/* Global variables */
int c; /* number of clusters */
float m, /* weighting exponent */
err; /* error tolerance */

/* Function Prototypes */
void Header(void);
void GetVars(void);
FILE *GetDataFile(int, char *);

FILE *GetLogFile(int, char *);
int CountPatterns(FILE *);
float CalculateMean(FILE *, int);
void InitCentres(FILE *, int);
void InitMembers(float, int);
float MemberChange(float, int);
void NewCentres(FILE *, float *, int);
void UpdateMembers(FILE *, float *, float *, float *, int);
float GetVariances(FILE *, float *, float *, float *, int);
float Criterion(FILE *, float *, float *, float *, int);
float AllocateSpace(int);

---

/* HEADER */
/* Function which creates a header */
void Header()
{
    clrscr();
    printf("Adaptive Fuzzy C-Means clustering algorithm \n");
    printf(" ------------------------------ \n \\
    \n");
}

/* GETVARS */
/* Function which prompts for user defined parameters to be used in the fuzzy c-means algorithm */
void GetVars()
{
    Header();
    printf("Input weighting coefficient \n");
    scanf("%f", &m);
    printf("Input error tolerance \n");
    scanf("%f", &err);
}

/* GETFILE */
/* Function which opens the data file. The file name may be supplied at the command line or it may be prompted for. */
FILE *GetDataFile(int argc, char **argv)
{
    char dname[DOSFNAMESIZE];
}
FILE *data;

Header();
if (argc > 3)
{
    printf("Usage: %s <data file> <log file>
\n", argv[0]);
    exit(FAILURE);
} else if (argc == 3 || (argc == 2))
{
    strcpy(dname, argv[1]);
} else
{
    printf("Input name of data file, including extension : ");
    scanf("%s", dname);
    fflush(stdin);
}
if ((data = fopen(dname, "r") == NULL)
{
    printf("File Error: Can't open <\%s> for reading\n", dname);
    exit(FAILURE);
}
return(data);

FILE *GetLogFile(int argc, char **argv)
{
    char lname[DOSFNAMESIZE];
    FILE *log;
    if (argc > 3)
    {
        printf("Usage: %s <data file> <log file>
\n", argv[0]);
        exit(FAILURE);
    } else if (argc == 3)
    {
        strcpy(lname, argv[2]);
    } else
    {
        printf("Input name of log file, including extension : ");
        scanf("%s", lname);
        fflush(stdin);
    }
    if ((log = fopen(lname, "w") == NULL)
    {
        printf("File Error: Can't open <\%s> for writing\n", lname);
        exit(FAILURE);
    }
    return(log);
}

int CountPatterns(FILE *data)
{
    int n = -1; /* number of patterns */
    float dummy; /* data point from data file */
    fseek(data, 0L, SEEK_SET); /* reset data file back to the beginning */
    while (!feof(data)) /* count the number of floats until eof */
    {
        fscanf(data, "%f", &dummy);
        n++;
    }
    return(n);
}

float CalculateMean(FILE *data, int n)
{
    float datapoint, 
    cum=0.0, /* data point from a datafile */
    mean; /* cumulated values of data points */
    mean; /* mean of all data points */
fseek(data, 0L, SEEK_SET); /* reset data file to the beginning */
while (!feof(data)) {
    fscanf(data, "%f", &datapoint);
    cum += datapoint;
}
mean = (float)(cum/n);
return(mean);

void InitCentres(FILE *data, float *V)
{
    float min, /* Minimum value in data file */
    max, /* Maximum value in data file */
    fit; /* Float read from data file */
    int i; /* Loop counter */
    if (!feof(data)) /* Get initial values for min and max */
    {
        fscanf(data, "%f", &flt);
        min = fit;
        max = fit;
    }
    fseek(data, 0L, SEEK_SET); /* Reset data file to start */
    while (!feof(data)) /* Read the rest of the file to update min/max */
    {
        fscanf(data, "%f", &flt);
        if (flt < min)
            min = flt;
        else if (flt > max)
            max = flt;
    }
    for (i = 0; i < c; i++)
        V[i] = min + (i+1)*(max-min)/(c+1);
}

void InitMembers(float **U, int n)
{
    int i, j; /* Loop counters */
    float *memsums; /* Sum of membership for each data point */
    /* allocate memory for memsums */
    if ((memsums = calloc(n, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in < InitMembers> \n");
        exit(FAILURE);
    }
    randomize(); /* start the random number generator randomly */
    for (j = 0; j < n; j++) /* Fill Membership grades with random values */
    {
        for (i = 0; i < c; i++)
        {
            U[i][j] = (float)rand() / RAND_MAX;
            memsums[j] += U[i][j];
        }
    }
    /* Normalise membership values to meet conditions */
    for (i = 0; i < c; i++)
    {
        U[i][j] /= memsums[j];
    }
    free(memsums);
}

float MemberChange(float **U, float **oldU, int n)
{
    float sqerr = 0; /* normalised error */
    /* function which calculates the normalised error between the new membership grades and the previous membership grades. */
int i, j; /* Loop counters */

for (i = 0; i < c; i++)
{
    for (j = 0; j < n; j++)
    {sqerr += (float)pow((U[i][j] - oldU[i][j]), 2);
    }
    sqerr = (float)sqrt(sqerr) * 0.5;
    return(sqerr);
}

void NewCentres(FILE *data, float **U, float *V, int n)
{
    for (i = 0; i < c; i++)
    {
        fseek(data, 0L, SEEK_SET);
        num = 0.0;
        den = 0.0;
        for (j = 0; j < n; j++)
        {
            fscanf(data, "%f", &flt);
            u = (float)pow(U[i][j], m);
            num += u * flt;
            den += u;
        }  // j
        V[i] = num/den;
    }  // i
}

void UpdateMembers(FILE *data, float **U, float **oldU, float *V, int n)
{
    for (i = 0; i < c; i++)
    {
        for (j = 0; j < n; j++)
        {oldU[i][j] = U[i][j];
        }
    }  // i
    for (j = 0; j < c; j++)
    {
        fseek(data, 0L, SEEK_SET);
        for (k = 0; k < n; k++)
        {
            fscanf(data, "%f", &flt);
            num = (float)pow((flt - V[i]), 2);
            for (j = 0; j < c; j++)
            {
                den = (float)pow((flt - V[j]), 2);
                inter += (float)pow(num/den, (1/(m-1)));
            }  // j
            U[i][k] = 1/inter;
        }  // k
    }  // j
}

void GetVariances(FILE *data, float *prevV, float *variance, float *clusterpoints, int n)
{
    for (i = 0; i < c; i++)
    {
        winner = winner;
        datapoint = datapoint;
        minimum = minimum;
        *clusterdistance = *clusterdistance;
    }  // i
}  // End of the program
totaldistance; /* summed cluster distances */

clusterdistance = AllocateSpace(c); /* Allocate space */
totaldistance = AllocateSpace(c);

fseek(data, 0L, SEEK_SET);
for (i = 0; i < n; i++)
{
    fscanf(data, "%f", &datapoint);
    for (j = 0; j < c; j++)
    {
        clusterdistance[i] = (float)pow(datapoint - prevV[i], 2);
    }
    minimum = clusterdistance[0];
    winner = 0;
    for (i = 0; i < c; i++)
    {
        if (clusterdistance[i] < minimum)
        {
            minimum = clusterdistance[i];
            winner = i;
        }
    }
    totaldistance[winner] += minimum;
    clusterpoints[winner] += 1;
}
for (i = 0; i < c; i++)
{
    if (clusterpoints[i] == 0)
        variance[i] = 0;
    else
        variance[i] = totaldistance[i] / clusterpoints[i];
}
free(clusterdistance);
free(totaldistance);

CRITERION
Function which determines the # clusters by minimizing criterion2
-------------------------------------------------------------
float Criterion(FILE *data, float *U, float *V, float mean, int n)
{
    int i, j;
    float datapoint;
    float S = 0;
    fseek(data, 0L, SEEK_SET);
    for (j = 0; j < n; j++)
    {
        fscanf(data, "%f", &datapoint);
        for (i = 0; i < c; i++)
        {
            S += pow((float) U[i][j], (float) V[j] - mean, (float) 2);
            ...
        }
    }
    return(S);
}

ALLOCATE SPACE
Function which allocates space for pointers
-------------------------------------------------------------
float *AllocateSpace(int c)
{
    float *dummy;
    if ((dummy = calloc(c, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in \"MAIN\"\n");
        exit(FAILURE);
    }
    return(dummy);
}

MAIN-BODY
/* Number of patterns of data */
/* Counts number of iterations */
/* Loop counter */
FILE *data_file,
     *log_file; /* File of data to be clustered */
     /* Log of sqerr */
float °V,
"prevV,
°oldU,
°U,
sqerr,
mean,
S,
oldS=MAXFLOAT,
°variance,
°clusterpoints;

data file = GetDataFile(argc, argv);
log file = GetLogFile(argc, argv);
GetVars();

n = CountPatterns(data file);
mean = CalculateMean(data_file, n);
c=1;

break1:
V = AllocateSpace(c);

v = AllocateSpace(c);

c1usterpoints = AllocateSpace(c);

if ((U = calloc(c, sizeof(float *))) == NULL)
{
    printf("Memory Allocation Error in <MAIN>\n");
    exit(FAILURE);
}

for (i=0; i<c; i++)
{
    if (U[i] = calloc(n, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in <MAIN>\n");
        exit(FAILURE);
    }
}

if ((oldU = calloc(c, sizeof(float *))) == NULL)
{
    printf("Memory Allocation Error in <MAIN>\n");
    exit(FAILURE);
}

for (i=0; i<c; i++)
{
    if (oldU[i] = calloc(n, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in <MAIN>\n");
        exit(FAILURE);
    }
}

InitCentres(data file, V);
InitMembers(U, n);

Header();
printf("No. of clusters: \n", c);
printf("Weighting coefficient : %f\n", m);
printf("Threshold : %f\n", err);
count = 0;

while ((sqerr = MemberChange(U, oldU, n)) > err)
{
    gotoxy(1,8);
    printf("Epoch: %d\n", count);
    printf("Clustering error : %f\n", sqerr);
    for (i=0; i<c; i++)
    {
        printf("Centre %d : %f\n", i, V[i]);
    }
}

S = Criterion(data file, U, V, n);
UpdateMembers(data file, U, oldU, V, n);
count ++;

GetVariances(data file, prevV, variance, clusterpoints, n);
}

break2:
/* optimal number of clusters is found */
c=1;
GetVariances(data file, prevV, variance, clusterpoints, n);

gotoxy(1,9);
printf("Centre : %f	 
\sigma : %f	 
\#Data Points: %d\n", i, prevv[i], sqrt(variance[i]));

for (i=0; i<n; i++)
{
    printf("i, prevV[i], sqrt(variance[i]), (int)clusterpoints[i]);
}

printf("Optimal number of clusters: %d\n", c);
printf("Total number of data points in data set: %d\n", n);
printf("Clustering Error within specified threshold: %f\n", sqerr);

getch();

return(SUCCESS);

free(V);
free(prevV);
free(variance);
free(clusterpoints);
for (i=0; i<n; i++)
{
    free(U[i]);
    free(oldU[i]);
}
free(U);
free(oldU);
fclose(data_file);
fclose(log_file);
return(SUCCESS);
FILE: Kohonen1.c
DESCRIPTION: Implementation of the Kohonen Clustering Network
AUTHOR: Bart C. Veenendaal
LANGUAGE: Borland C++ version 3.1
DATE: 18/5/93
COMPANY: OMRON-ETC-NL 's Hertogenbosch

29-09-93 Second working version: Random data point selection
01-06-93 First working version: single winner update
26-05-93 Second development version
18-05-93 Development version

Header Files
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <alloc.h>
#include <time.h>
#include <math.h>

Global Constants
#define SUCCESS 0
#define FAILURE -1
#define DOSFNAMESIZE 40
#define LEARNINGRATE 0.05

Global variables
int c; /* number of clusters */
float err; /* error tolerance */

Function Prototypes
void Header(void);
void GetVars(void);
FILE *GetDataFile(int argc, char **argv);
FILE *GetLogFile(int argc, char **argv);
int CountPatterns(FILE *);
void StoreData(FILE *, float *, int);
void InitCentres(FILE *, float *);
int WinningCluster(float, float *);
float CentreChange(float, float *);
void UpdateCentres(float, float *, int, int);
void CalculateVariance(FILE *, float *, float *, float *, int);
float *AllocateSpace(int);

/*---------------------------------------------*/
/* HEADER */
/* Function which creates a header */
void Header()
{
  clrscr();
  printf("KOHONEN CLUSTERING NETWORK ALGORITHM: single winner \n");
  printf("\n\n");
}

/*------------------*/
/* GETVARS */
/* Function which prompts for user defined parameters to be used */
/* in the kohonen clustering algorithm */
void GetVars()
{
  Header();
  printf("Input # of clusters \n");
  scanf("%d", &c);
  fflush(stdin);
  printf("Input error tolerance \n");
  scanf("%f", &err);
  fflush(stdin);
}

/*------------------*/
/* GETFILE */
/* Function which opens the data file. The file name may be */
/* supplied at the command line or it may be prompted for. */
FILE *GetDataFile(int argc, char **argv)
{
  char dname[DOSFNAMESIZE];
  FILE *data;
  Header();
  if (argc > 3)
  {
### GETFILE

Function which opens the log file. The file name may be supplied at the command line or it may be prompted for.

```c
FILE *GetLogFile(int argc, char **argv) {
    char "name[DOSFNAMESIZE];
    FILE "log;
    if (argc > 3) {
        printf("Usage: %s < data file > < log file > \n", argv[0]);
        exit(FAILURE);
    } else if (argc == 3) {
        strcpy("name, argv[2]); /* log file name */
    } else {
        printf("Input name of log file, including extension : ");
        scanf("%s", "name);
        fflush(stdin);
    }
    if ((log = fopen("name, "w")) == NULL) {
        printf("File Error: Can't open <%s> for writing\n", "name);
        exit(FAILURE);
    }
    return(log);
}
```

### COUNTPATTERNS

Function which returns the number of datapoints in the data set.

```c
int CountPatterns(FILE *data) {
    int n = -1;
    float dummy;
    while (!feof(data)) /* count number of floats until eof */
    {
        fscanf(data, "%f", &dummy);
        n++;
    }
    fseek(data, 0L, SEEK_SET); /* reset data file back to the beginning */
    return(n);
}
```

### STOREDATA

Function which stores the entire data set in an array.

```c
void StoreData(FILE *data, float *dataset, int n) {
    int j; /* loop counter */
    float flt; /* float */
    for (j=0; j<n; j++) {
        fscanf(data, "%f", &flt);
        dataset[j] = flt;
    }
    fseek(data, 0L, SEEK_SET); /* reset data file back to the beginning */
}
```
/* INITCENTRES */
Function which initialises the centres equally dividing the input space

void InitCentres(FILE *data, float *centre)
{
    float min, max, fit;
    int i;
    if (!feof(data)) /* Get initial values for min and max */
    {
        fscanf(data, "%f", &flt);
        min = fit;
        max = fit;
    }
    while (!feof(data)) /* read the rest of the file to update min/max */
    {
        fscanf(data, "%f", &flt);
        if (flt < min)
            min = fit;
        else if (flt > max)
            max = fit;
    }
    for (i = 0; i < c; i++)
        centre[i] = min + (i + 1) * (max - min) / (c + 1);
    fseek(data, 0L, SEEK_SET); /* Reset data file to start */
}

/* WINNINGCLUSTER */
Function which determines the winning cluster

int WinningCluster(float datapoint, float *centre)
{
    int winner, i;
    float minimum, *clusterdistance;
    clusterdistance = AllocateSpace(c);
    for (i = 0; i < c; i++)
    {
        clusterdistance[i] = (float)pow(datapoint - centre[i], 2);
    }
    minimum = clusterdistance[0];
    winner = 0;
    for (i = 1; i < c; i++)
    {
        if (clusterdistance[i] < minimum)
        {
            minimum = clusterdistance[i];
            winner = i;
        }
    }
    free(clusterdistance);
    return(winner);
}

/* UPDATECENTRES */
Function which updates the centres according to the Kohonen clustering algorithm: single winner update

void UpdateCentres(float datapoint, float *centre, int winner, int counter)
{
    float LR;
    int i;
    LR = LEARNINGRATE * exp(-counter / 10);
    centre[winner] += LR * (datapoint - centre[winner]);
}

/* CENTRECHANGE */
Function which calculates the normalised squared error between the new centres and the previous centres

float CentreChange(float *centre, float *oldcentre)
{
    float sqerr = 0;
    int i;
    for (i = 0; i < c; i++)
    {
        sqerr += pow(centre[i] - oldcentre[i], 2);
    }
    return(sqrt(sqerr / c));
}
{ sqerr += (float)pow(centre[i]-oldecentre[i], 2); }

sqerr = 0.5*(float)sqrt(sqerr);
return(sqerr);

/* CALCULATE VARIANCE */
/* Function which calculates the variance of each cluster */

void CalculateVariance(FILE *data, float *centre, float *variance, float *clusterpoints, int n)
{
    int i,j, winner; /* loop counters */
    float datapoint, minimum, *clusterdistance, *totaldistance; /* cluster with minimum distance */
    datapoint, minimum, *clusterdistance, *totaldistance; /* summed cluster distances */

    clusterdistance = AllocateSpace(c);
    totaldistance = AllocateSpace(c);

    fseek(data, 0L, SEEK_SET);
    for (i=0; i<n; i++) {
        fscanf(data, "%f", &datapoint);
        for (i=0; i<c; i++)
            clusterdistance[i] = (float)pow(datapoint-centre[i], 2);
        minimum = clusterdistance[0];
        winner = 0;
        for (i=1; i<c; i++)
            if (clusterdistance[i] < minimum)
                { minimum = clusterdistance[i]; winner = i; }
        totaldistance[winner] += minimum;
        clusterpoints[winner] += 1;
    }
    for (i=0; i<c; i++)
        if (clusterpoints[i] == 0) variance[i] = 0;
        else variance[i] = totaldistance[i]/clusterpoints[i];

    free(clusterdistance);
    free(totaldistance);
}

/* ALLOCATE SPACE */
/* Function which allocates space for pointers */

float *AllocateSpace(int c)
{
    float *dummy;
    if ((dummy = calloc(c, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in <MAIN>
        exit(FAILURE);
    }
    return(dummy);
}

/* MAIN PROGRAM */
/* MAIN PROGRAM */
int main(int argc, char **argv)
{
    int n, counter=0, i,j, runner, winner;
    FILE *data file, *log file; /* Number of patterns of data */
    float *dataset, *centre, *oldecentre, sqerr, datapoint, *variance, /* Counts the number of epochs */
        /* loop counters */
        /* pointer in data/Array */
        /* cluster with minimum distance */
        /* File of data to be clustered */
        /* Log of sqerr */
        /* Stored entire data set */
        /* Vector of centres */
        /* Matrix of previous centres */
        /* RMS error */
        /* datapoint red from data file */
        /* variance of cluster */

    /* */
/* clusterpoints; */
data_file = GetDataFile(argc, argv);
log_file = GetLogFile(argc, argv);
GetVar();
n = CountPatterns(data_file);

centre = AllocateSpace(c); /* Allocate space */
oldcentre = AllocateSpace(c);
variance = AllocateSpace(c);
c1usterpoints = AllocateSpace(c);
if ((dataset = calloc(n, sizeof(float))) == NULL)
{
    printf("Memory Allocation Error in <MAIN>\n");
    exit(FAILURE);
}

InitCentres(data_file, centre);
Header();
printf("%d cluster(s)\n", c);
printf("Threshold : %f\n", err);
for (i=0; i<=c; i++)
{
    oldcentre[i]=0;
}
while ((sqerr = CentreChange(centre, oldcentre)) > err)
{
    for (i=0; i<=c; i++)
    {
        oldcentre[i]=centre[i];
    }
    lseek(data_file, 0L, SEEK_SET);
    StoreData(data_file, dataset, n);
    randomize();
    for (j=0; j<n; j++)
    {
        gotoxy(1,7);
        printf("Epoch: %d\n", counter);
        print("Clustering error : %f\n", sqerr);
        for (i=0; i<=c; i++)
        {
            printf("Centre %d : %f, i, centre[i]);
        }
        runner = random(n-j); /* random selection of data point */
datapoint = dataset[runner];
if (runner!=n-j-1) /* if last data point is selected */
dataset[runner] = dataset[n-j-1]; /* the dataset should not be adjusted */
winner = WinningCluster(datapoint, centre);
UpdateCentres(datapoint, centre, winner, counter);
    }
    counter++;
    CalculateVariance(data_file, centre, variance, c1usterpoints, n);
    for (i=0; i<=c; i++)
    {
        fprintf(log_file, "%f\n", centre[i], sqrt(variance[i]));
    }
    gotoxy(1,8);
    printf("Clustering error : %f\n", sqerr);
    for (i=0; i<=c; i++)
    {
        printf("centre %d : %f\n", i, centre[i], sqrt(variance[i]));
    }
    print("\n");
    print("Total number of DataPoints in data set:\n")
    print("Clustering Error within specified threshold\n");
gotoxy(5,
getch();
free(dataset);
free(centre);
free(oldcentre);
free(variance);
free(c1usterpoints);
free(data_file);
fclose(log_file);
return(SUCCESS);
}
FILE: PCM.c

DESCRIPTION: Implementation of Possibilistic C-Means clustering alg.

AUTHOR: Bart C. Veenendaal

LANGUAGE: Borland C++ version 3.1

DATE: 23-09-93

COPYRIGHT: OMRON European Technical Centre, 's Hertogenbosch

23-09-93 Possibilistic C-Means clustering in which the possibility
distributions are initialised by the Fuzzy C-Means alg.

11-06-93 Updated version of Fuzzy C-Means algorithm

25-11-92 First version of Fuzzy C-Means algorithm by BWG

FILE: PCM.c

DESCRIPTION: Implementation of Possibilistic C-Means clustering alg.

AUTHOR: Bart C. Veenendaal

LANGUAGE: Borland C

+ +

version 3.1

DATE: 23-09-93

COPYRIGHT: OMRON European Technical Centre, 's Hertogenbosch

23-09-93 Possibilistic C-Means clustering in which the possibility
distributions are initialised by the Fuzzy C-Means alg.

11-06-93 Updated version of Fuzzy C-Means algorithm

25-11-92 First version of Fuzzy C-Means algorithm by BWG

/* Header Files */
#define SUCCESS 0
#define FAILURE -1
#define DOSFNAMESIZE 40

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <alloc.h>
#include <time.h>
#include <math.h>

/* Global Constants */
#endif

/* Global variables */
int c;
float m,
FCMerr,
PCMerr;

/* Function Prototypes */
void Header1(void);
void Header2(void);
void GetVars(void);
FILE *GetDataFile(int, char **);
FILE *GetLogFile(int, char **);
int CountPatterns(FILE *);
void InitCentres(FILE *, float *);
void InitMembers(float **, int);
float MemberChange(float **, float **, int);
void CalculateEta(FILE *, float *, float *, float *, float *, float *, int);
void NewEta(FILE *, float **, float *, float *, float *, float *, int);
void NewCentres(FILE *, float *, float *, float *, int);
void UpdateMembers1(FILE *, float **, float *, float *, float *, float *, int);
void UpdateMembers2(FILE *, float **, float *, float *, float *, int);
float *AllocateSpace(int);

void Header1()
{
    clrscr();
    printf("Fuzzy C-Means clustering algorithm for initialisation \n\n");
}

void Header2()
{
    clrscr();
    printf("Possibilistic C-Means clustering algorithm \n\n");
}

void GetVars()
{
    Header1();
    printf("Input # of clusters: \t\n");
    scanf("%d", &c);
    fflush(stdin);
}
printf("Input weighting coefficient m:\t");
scanf("%f", &m);
fflush(stdin);
printf("Input FCM error tolerance :\t");
scanf("%f", &FCMerr);
fflush(stdin);
printf("Input PCM error tolerance :\t");
scanf("%f", &PCMerr);
fflush(stdin);

FILE *GetDataFile(int argc, char **argv) {
    char dname[DOSFNAMESIZE];
    FILE *data;
    Header1();
    if (argc > 3) {
        printf("Usage: %s < data file > < log file >\n", argv[0]);
        exit(FAILURE);
    }
    else if ((argc == 3) || (argc == 2)) {
        strcpy(dname, argv[1]);
    }
    else {
        printf("Input name of data file, including extension : ");
        scanf("%s", dname);
        fflush(stdin);
        if ((data = fopen(dname, "r")) == NULL) {
            printf("File Error: Can't open <%s> for reading\n", dname);
            exit(FAILURE);
        }
        return(data);
    }
}

FILE *GetLogFile(int argc, char **argv) {
    char lname[DOSFNAMESIZE];
    FILE *log;
    if (argc > 3) {
        printf("Usage: %s < data file > < log file >\n", argv[0]);
        exit(FAILURE);
    }
    else if (argc == 3) {
        strcpy(lname, argv[2]);
    }
    else {
        printf("Input name of log file, including extension : ");
        scanf("%s", lname);
        fflush(stdin);
        if ((log = fopen(lname, "w")) == NULL) {
            printf("File Error: Can't open <%s> for writing\n", lname);
            exit(FAILURE);
        }
        return(log);
    }
}

int CountPatterns(FILE *data) {
    int n = -1;
    float dummy;
}
while (!feof(data)) /* count the number of floats until the eof */  
{  
   fscanf(data, "%f", &dummy);  
n++;  
}  
seek(data, 0L, SEEK_SET); /* reset data file back to the beginning  
return(n);  
  
void InitCentres(FILE *data, float *V)  
{  
float min, /* Minimum value in data file */  
max, /* Maximum value in data file */  
flt; /* Float read from data file */  
int i; /* Loop counter */  
  
if (!feof(data)) /* Get initial values for min and max */  
{  
   fscanf(data, "%f", &flt);  
   min = flt;  
   max = flt;  
}  
while (!feof(data)) /* read the rest of the file to update min/max */  
{  
   fscanf(data, "%f", &flt);  
   if (flt < min)  
      min = flt;  
   else if (flt > max)  
      max = flt;  
}  
for (i = 0; i < c; i++)  
V[i] = min + (i+1)*(max-min)/(c+1);  
seek(data, 0L, SEEK_SET); /* Reset data file to start */  
  
void InitMembers(float **U, int n)  
{  
int i,j; /* Loop counters */  
float *memsums; /* Sum of membership for each data point */  
  
if ((memsums = calloc(n, sizeof(float))) == NULL) /* allocate memory */  
{  
   printf("Memory Allocation Error in <InitMembers> \n");  
   exit(FAILURE);  
}  
randomize(); /* start the random number generator randomly */  
for (j = 0; j < n; j++) /* Fill Membership grades with random values */  
{  
   for (i = 0; i < c; i++)  
   {  
      U[i][j] = (float)rand()/RAND_MAX;  
      memsums[j] += U[i][j];  
   }  
}  
for (j = 0; j < n; j++) /* Normalise membership values to meet conditions */  
{  
   for (i = 0; i < c; i++)  
   {  
      U[i][j] /= memsums[j];  
   }  
}  
  
float MemberChange(float **U, float **oldU, int n)  
{  
float sqerr = 0; /* normalised error */  
int i,j; /* Loop counters */  
  
for (i = 0; i < c; i++)  
{  
   for (j = 0; j < n; j++)  
   {  
      U[i][j] -= oldU[i][j];  
      sqerr += pow(U[i][j], 2);  
   }  
}  
  
return(sqrt(sqerr/n));  
}  
  
/*---------------------------*/  
/* INIT CENTRES */  
/* Function which initialises the centres in such a way that the */  
/* input space is equally divided */  
/*---------------------------*/  
  
/* INIT MEMBERS */  
/* Function to randomly initialise the membership grades of each data */  
/* point to each cluster so that the condition that the sum of the */  
/* memberships for any one data point equals 1 and the sum for any one */  
/* cluster lies between zero and the total number of data points. This */  
/* is true because each membership grade exists on the set [0,1]. */  
/*---------------------------*/  
  
/* MEMBER CHANGE */  
/* Function which calculates the normalised error between the new */  
/* membership grades and the previous membership grades. */  
/*---------------------------*/
for (j=0; j<n; j++)
    sqerr += (float)pow((U[j][i]-oldU[i][j]), 2);
}
sqerr = (float)sqrt(sqerr) * 0.5;
return(sqerr);

/*----------------------*/
/* CALCULATE-ETA */
/* Function which calculates the value of eta for each cluster i */
void CalculateEta(FILE *data, float **U, float *V, float *eta, int n)
{
    int i,j; /* Loop counters */
    float num, /* Numerator of calculation */
    den, /* Denominator of calculation */
    u, /* Intermediate calculation */
    flt; /* Float read in from data file */
    for (i=0; i<c; i++)
    {
        num = 0.0;
        den = 0.0;
        for (j=0; j<n; j++)
        {
            fscanf(data, "%f", &flt);
            u = (float)pow(U[j][i], m);
            num += u * (float)pow(flt-V[i], 2);
            den += u;
        }
        eta[i] = num/den;
        fseek(data, 0L, SEEK_SET);
    }
}

/*----------------------*/
/* NEW-ETA */
/* Function which calculates the new eta's for each cluster i, */
/* according to an alpha-cut */
void NewEta(FILE *data, float **U, float *V, float *eta, float *variance, int n)
{
    int i,j;
    float alpha = 0.2, /* alpha for alpha-cut */
            flt, /* data point */
            distance, /* distance between flt and V */
            *possdistr; /* possibility distribution */
    possdistr = AllocateSpace(c); /* calculation of possibility distributions */
    for (i=0;i<c;i++)
    {
        possdistr[i] = 2*sqrt(variance[i])*sqrt(-2*log(alpha));
    }
    for (i=0;i<c;i++)
    {
        for (j=0;j<n;j++)
        {
            distance = 0.0;
            fscanf(data, "%f", &flt);
            if ((U[i][j])>alpha) /* calculation of alpha-cut */
                {
                    distance += (float)pow(flt-V[i], 2);
                }
        }
    }
    if (possdistr[i]!=0)
        eta[i]=distance/possdistr[i];
    fseek(data, 0L, SEEK_SET);
}

/*----------------------*/
/* NEWCENTRES */
/* Function which updates the centres */
void NewCentres(FILE *data, float **U, float *V, int n)
{
    int i,j; /* Loop counters */
    float num, /* Numerator of calculation */
    den, /* Denominator of calculation */
    u, /* Intermediate calculation */
    flt; /* Float read in from data file */
    for (i=0; i<c; i++)
    {
        num = 0.0;
        den = 0.0;
        for (j=0; j<n; j++)
        {
            num += u * (float)pow(flt-V[i], 2);
            den += u;
        }
        if (possdistr[i]!=0)
            eta[i]=distance/possdistr[i];
        fseek(data, 0L, SEEK_SET);
    }
}

73
for (j = 0; j < n; j++)
{
    fscanf(data, "%f", &flt);
    u = (float)pow(U[i][j], m);
    num += u * fit;
    den += u;
}
V[i] = num/den;
fseek(data, 0L, SEEK_SET);
}

UPDATEMEMBERS1
Function which updates the grades of membership in FCM algorithm

void UpdateMembers1(FILE *data, float **U, float **oldU, float *V, int n)
{
    int i, j, k; /* Loop counters */
    float num, /* Numerator of calculation */
    den, /* Denominator of calculation */
    inter, /* Intermediate result */
    fit; /* data read from file */

    for (i = 0; i < c; i++)
    {
        for (j = 0; j < n; j++)
        {
            oldU[i][j] = U[i][j];
        }
    }
    for (i = 0; i < c; i++)
    {
        for (k = 0; k < n; k++)
        {
            inter = 0.0;
            fscanf(data, "%f", &flt);
            num = (float)pow((flt - V[i]), 2);
            for (j = 0; j < c; j++)
            {
                den = (float)pow((flt - V[j]), 2);
                inter += (float)pow(num/den, (1/(m-1)));
            }
            U[i][k] = 1/inter;
        }
    }
    fseek(data, 0L, SEEK_SET);
}

UPDATEMEMBERS2
Function which updates the possibility distributions in PCM algorithm

void UpdateMembers2(FILE *data, float **U, float **oldU, float *V, float *eta, int n)
{
    int i, j;
    float distance, /* Distance between float and centre */
    fit; /* data read from file */

    for (i = 0; i < c; i++)
    {
        for (j = 0; j < n; j++)
        {
            oldU[i][j] = U[i][j];
        }
    }
    for (j = 0; j < n; j++)
    {
        fscanf(data, "%f", &flt);
        distance = (float)pow((flt - V[i]), 2);
        U[i][j] = 1/(1 + (float)pow(distance/eta[i], (1/(m-1))));
    }
    fseek(data, 0L, SEEK_SET);
}

GETVARIANCES
Calculates the variances for each cluster after clustering

void GetVariances(FILE *data, float *V, float *variance, float *clusterpoints, int n)
{
    int i, j;
    float winner; /* cluster with minimum distance */
    float datapoint, /* datapoint red from datafile */
    }
minimum; /* minimum distance */
*clusterdistance; /* distance from datapoint to cluster */
*totaldistance; /* summed cluster distances */

clusterdistance = AllocateSpace(c); /* Allocate space */
totaldistance = AllocateSpace(c);

for (j = 0; j < n; j++)
{
    fscanf(data, "%f", &datapoint);
    for (i = 0; i < c; i++)
    {
        clusterdistance[i] = (float)pow(datapoint-V[i], 2);
    }
    minimum = clusterdistance[0];
    winner = 0;
    for (i = 0; i < c; i++)
    {
        if (clusterdistance[i] < minimum)
        {
            minimum = clusterdistance[i];
            winner = i;
        }
    }
    totaldistance[winner] += minimum;
    clusterpoints[winner] += 1;
}
seek(data, 0L, SEEK_SET);
for (i = 0; i < c; i++)
{
    if (clusterpoints[i] == 0)
        variance[i] = 0;
    else
        variance[i] = totaldistance[i]/clusterpoints[i];
}
free(clusterdistance);
free(totaldistance);

/* ALLOCATE SPACE */
/* Function which allocates space for pointers */
/*----------------------------------------------------------------------------*/
float *AllocateSpace(int c)
{
    float *dummy;
    if ((dummy = calloc(c, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in <MAIN>\n");
        exit(FAILURE);
    }
    return(dummy);
}

/* MAIN-BODY */
/*----------------------------------------------------------------------------*/
int main(int argc, char **argv)
{
    int n, /* Number of patterns of data */
    count, /* Counts number of iterations */
    i, j; /* Loop counters */
    FILE *data_file, /* File of data to be clustered */
    *log_file; /* Log of sqerr */
    float *V, /* Vector of centres */
    **oldU, /* Matrix of previous membership grades */
    **L, /* Matrix of membership grades */
    *eta, /* Array of eta for every cluster */
    *sqerr, /* RMS error */
    *variance, /* Variance of cluster */
    *clusterpoints; /* Number of points in cluster */

data file = GetDataFile(argc, argv);
log file = GetLogFile(argc, argv);
GetVars();
n = CountPatterns(data file);
V = AllocateSpace(c);
variance = AllocateSpace(c);
clusterpoints = AllocateSpace(c);
if ((U = calloc(c, sizeof(float *)) == NULL)
{
    printf("Memory Allocation Error in <MAIN>\n");
    exit(FAILURE);
}
for (i=0; i<n; i++)
{
    if ((U[i] = calloc(n, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in \nMAIN\n");
        exit(FAILVRE);
    }
}
if (oldU = calloc(n, sizeof(float *))) == NULL)
    {
        printf("Memory Allocation Error in \nMAIN\n");
        exit(FAILVRE);
    }
for (i=0; i<n; i++)
{
    if ((oldU[i] = calloc(n, sizeof(float))) == NULL)
    {
        printf("Memory Allocation Error in \nMAIN\n");
        exit(FAILVRE);
    }
}
InitCentres(data_file, V); /* Begin of FCM algorithm */
InitMembers(U, n);
Header1();
printf("%d cluster(s)\n", c);
printf("Weighting coefficient : \%f\n", m);
printf("Threshold FCM: \%f, FCMerr\);
printf("Threshold PCM: \%f, PCMerr\);
count = 0;
while ((sqerr = MemberChange(U, oldU, n)) > FCMerr)
{
    gotoxy(1,9);
    printf("Epoch: %d\n", count);
    printf("Clustering error : \%f\n", sqerr);
    NewCentres(data_file, U, V, n);
    for (i=0; i<n; i++)
    {
        printf("Centre %d : \%f\n", i, V[i]);
        print("Centre %d : \%f\n", i, V[i]);
    }
    UpdateMembers1(data_file, U, oldU, V, n);
    count++;
}
eta = AllocatedSpace(c); /* Begin of PCM algorithm */
CalculateEta(data_file, U, V, eta, n);
Header2();
printf("%d cluster(s)\n", c);
printf("Weighting coefficient : \%f\n", m);
printf("Threshold FCM: \%f, FCMerr\);
printf("Threshold PCM: \%f, PCMerr\);
count = 0;
while ((sqerr = MemberChange(U, oldU, n)) > PCMerr)
{
    gotoxy(1,9);
    printf("Epoch: %d\n", count);
    printf("Clustering error : \%f\n\n", sqerr);
    for (i=0; i<n; i++)
    {
        printf("Centre %d : \%f\n", i, V[i]);
        print("Centre %d : \%f\n", i, V[i]);
    }
    NewCentres(data_file, U, V, n);
    UpdateMembers2(data_file, U, oldU, V, eta, n);
    count++;
}
GetVariances(data_file, V, variance, clusterpoints, n);
/* second time run of the algorithm */
NewEta(data_file, U, V, eta, variance, n);
free(clusterpoints);
clusterpoints = AllocatedSpace(c);
count = 0;
Header2();
printf("%d cluster(s)\n", c);
printf("Weighting coefficient : \%f\n", m);
printf("Threshold FCM: \%f, FCMerr\);
printf("Threshold PCM: \%f, PCMerr\);
count = 0;
while ((sqerr = MemberChange(U, oldU, n)) > PCMerr)
{
    gotoxy(1,9);
    printf("Epoch: %d\n", count);
    printf("Clustering error : \%f\n", sqerr);
    for (i=0; i<n; i++)
    {  

76
{ printf("Centre %d: %f\n", i, V[i]);
    newCentres(data_file, U, V, n);
    UpdateMembers2(data_file, U, oldU, V, eta, n);
    count++;
    GetVariances(data_file, V, variance, clusterpoints, n);
gotoxy(1,10);
    printf("Clustering error: %f\n", sqerr);
    for (i=0; i<c; i++)
        
        printf("Centre %d: %f\n", i, V[i]);
        fprintf(log_file,"%f\t%f\t%f\n", -i, V[i], sqrt(variance[i]), (int)clusterpoints[i]);
    printf("Total\n#datapoints: %d\n", n);
    printf("Clustering Error within specified threshold\n"); getch();

    free(V); /* Free all allocated space */
    for (i=0; i<c; i++)
        free(U[i]);
    free(U);
    free(oldU);
    free(eta);
    free(variance);
    free(clusterpoints);
    fclose(data_file); /* close files */
    fclose(log_file);
    return(SUCCESS);
}