# ID3: Incorporation of Fuzziness and Generation of Network Architecture

a dissertation submitted in partial fulfillment of the
requirements for the *M. Tech. (Computer Science)*
degree of the Indian Statistical Institute

By

**Pawan Kumar Singal**

*Under the Supervision of*

**Dr. Sushmita Mitra**
Associate Professor

**Prof. Sankar. K. Pal**
Distinguished Scientist

*Machine Intelligence Unit,*

INDIAN STATISTICAL INSTITUTE
203, Barrackpore Trunk Road
Calcutta - 700 035

## Certificate of Approval

This is to certify that the thesis titled *ID3: Incorporation of Fuzziness and Generation of Network Architecture* submitted by **Pawan Kumar Singal** towards partial fulfillment of the requirements for the degree of *M. Tech. (Computer Science)* at the Indian Statistical Institute, Calcutta, embodies the work done under our supervision. His work is satisfactory.

S Mitra 28.7.99 | 28.07.99

**Dr. Sushmita Mitra** | **Prof. Sankar. K. Pal**

# Acknowledgements

At the outset, I express my sincere gratitude to Dr. Susmita Mitra and Prof. Sankar. K. Pal for their invaluable guidance, innovative suggestions and encouragement. At every stage of this work - from the initial planning of work upto the final preparation of report, they made serious efforts to improve the quality of the work.

In doing this project, a number of individuals have rendered their helpful suggestions, comments and encouragement; for them I would like to express my grateful thanks.

First, I am thankful to Dr. D. P. Mandal, for his help and encouragement through out. Special thanks are due to Mr. B. Uma Shankar for helping with many technical details. I would like to thank all staff members of MIU for supporting me during this work.

I am also thankful to Mr. Sounak Mishra and Mr. Pabitra Mitra for their help. I also like to offer thanks to Mr. Hare Krushna Patel and Mr. Rajiv Verma, my friends and seniors for their valuable suggestions when I was new to this course.

My sincere thanks are due to Mr. Parveen Gupta, my friend, whose lot of time has been killed by me during my stay in ISI.

Finally, I wish to pay my regards to my parents for their patience and devotion.

Pawan Kumar Singal.

# Contents

# List of Figures

# Abstract

A new method of generating fuzzy knowledge-based network is described. Crude domain knowledge is extracted using the *ID3* algorithm. The *ID3* approach to classification consists of a procedure for synthesizing an efficient discriminatory tree for classifying patterns that have non numeric values. One of the problems with ID3 is that it cannot deal with numeric (continuous) data, which most practical problems have. In our work a method to use *ID3* in continuous attribute case is proposed. The rules are generated in linguistic form. They are mapped to a fuzzy neural network. Topology of the layered network is automatically determined and the net is finally pruned to generate an optimal architecture. The frequency of samples, representative of a rule, is also taken into consideration.Fuzzy membership concept is incorporated at the sample level to handle uncertainty. This involves changing the decision function at the node level. A novel approach to map confidence factors of unresolved/ ambiguous nodes directly into a fuzzy neural network is also described.

The effectiveness of the algorithm is demonstrated on a vowel recognition problem.

# Chapter 1

# Introduction

Today, in the mass storage era, the knowledge acquisition process represents a major knowledge engineering bottleneck. Computer programs extracting knowledge from data successfully attempt to lessen this problem. Among such systems, inducing decision trees for classification is very popular. The resulting knowledge in the form of decision trees are found to be quite comprehensible. Quinlan popularized the concept of decision trees by inducing ID3 [1], which stands for *Interactive Dichotomizer 3*. Systems based on this approach use an information theoretic measure of entropy for assessing the discriminatory power of each attribute. ID3 is a popular and efficient method of making decision for classification of *symbolic* data. A decision tree assigns symbolic design to new samples. This makes it inapplicable in cases where a numeric decision in needed. As a matter of fact, most of the real life problems deal with non symbolic (numeric, continuous) data. Recognition problem with continuous-valued attribute must, therefore, be discretized prior to attribute selection. Another problem with ID3 is, it can not provide any information in the intersection region when the classes are highly overlapping.

In recent years, neural networks have become equally popular due to their relative ease of application and ability to provide gradual responses. The back propagation algorithm [2, 3] is central to much current work on learning in neural network. There are many theoretic questions concerning what a multi-layered perceptron [2, 3] can do and can not do; how many layers are needed for a given task; how many units per layer are necessary; and so on. It has been proved that with at most two hidden layers, arbitrary accuracy is obtainable given enough units per layer. It has also been proved that only one hidden layer is enough to approximate any continuous function. The utility of these results depends on how many hidden units are necessary and this is in general unknown. In many cases it may grow exponentially with the number of input units.

In this work we propose a scheme for discretization of continuous attribute so that the ID3 algorithm can be used in continuous attribute cases. Fuzziness is incorporated in the classical algorithm such that in the crisp cases when the classes are not overlapping it will boil down to the classical ID3. In the case of overlapping classes it provides extra information regarding the overlapped region. Keeping the decision tree generated by ID3 in the background we propose a scheme to generate a neural network architecture and a method for initial weight encoding. The method enables automatic generation of the appropriate number of hidden nodes and layer of the network. This is unlike the use of random initial weights in the conventional neural network.

The report consist of five chapters. In Chapter 2 a brief introduction to ID3 and related work is given. The scheme to generate rules from the decision tree is also explained. Method to map the generated rules on to the generated neural network architecture is discussed. In Chapter 3 the way of incorporation of fuzziness in ID3 is discussed. This is used to generate modified ID3 rules that are mapped onto the generated layered network. Chapter 4 gives a brief idea of results on Indian Telugu Vowel sounds. Conclusion and discussions are included in Chapter 5.

# Chapter 2

# ID3: Rule Generation and Network Architecture

In this chapter we discuss about ID3, Neural Network (in particular feed forward network), back-propagation algorithm, rule generation from ID3 and mapping of these rules onto the neural net whose topology is automatically defined. The chapter consist of four sections. Section 1 describes the decision tree, ID3 and some related works. In Section 2, a brief introduction on neural networks, particularly on MLP, is given. Section 3 deals with the proposed method of discretization of continuous attributes so that ID3 can be applied in such cases also. Rule generation from decision and mapping these rules onto the net in the form of weights is discussed in Section 4.

## 2.1 Decision Trees

Decision trees have extensively been used as classifiers in pattern recognition [4]. By applying the decision tree methodology, one difficult decisions can be broken into a sequence of less difficult decision. Because of the tree structure, only some of all possible questions are asked in the process of making the final decision. The final decision is made at the terminal node, which is reached by traversing the tree, starting from the root as indicated by decision made at the internal nodes. The design is performed in top-down fashion. The nodes are split during the design procedure according to some criteria. Each node in the decision tree is either a leaf node (decision node) or an internal node (a testing node). Each leaf node represents a class (unique). If any data point reaches this node after traversing from root of the tree, we conclude that the class of the data point is that represented by the leaf node. On the other hand, each internal node will represent a test with respect to a feature. There are many algorithms in the literature to generate classification or the decision trees. Among them ID3 [1] and CART [5] are the two most important discriminative algorithms working by recursive partitioning the sample space. The basic idea is the same: partitioning the sample space in data-driven manner and representing the partition as the tree. An important property of these algorithms is that they attempt to minimizes the size of the tree simultaneously they attempt to optimize some quality measure. Since in any pattern recognition problem [6] any object is characterized by a set of features, it is better to discuss about the domain type of the features before going into the detail of the said algorithm .

### 2.1.1 Domain Types

In general there are two different kinds of domains for features : *discrete and continuous*. In the discrete case each feature can have a number of values called attribute values, and an object or an data point is characterized by the values of these attributes. Continuous values of features have a proximity relation between them. In general a discrete domain may be unordered, partially ordered or totally ordered whereas the continuous domain is always totally ordered.

ID3 assumes discrete domains with small cardinalities. This is a great advantage as this increases the comprehensibility of the induced knowledge. But this algorithm requires priori partitioning. On the other hand, CART algorithm does not require prior partitioning. The conditions on the tree are based on thresholds (on continuous domain) which are dynamically computed. Because of that, a condition on a path can use a feature a number of times with different thresholds and the thresholds on the different paths are very likely to be different. This idea often increases the quality of the tree but at the cost of comprehensibility. It is to be noted here that CART can be used in case of continuous attribute value but ID3 can not.

Our objective is high comprehensibility along with processing of continuous value. This is accomplished with the help of the ID3 algorithm. So before proceeding further we introduce the ID3 algorithm [7, 1]. This is followed by a brief discussion on the related work of handling continuous valued attributes.

### 2.1.2 Algorithm ID3

In the training set we have $N$ patterns partitioned into sets of pattern belonging to class $C_i, i = 1, 2, ...l$. The population in class $C_i$ is $N_i$, and each pattern has $n$ attributes.
input: Data file $D_1$ with $n$-dimensional attribute, labeled data.
output : Decision tree.
method :

1. *Calculate initial value of entropy.*

   For the training set, the class belonging of each pattern point is known. The initial entropy for the system consisting of $N$ labeled pattern is :-

   $Ent(I) = \sum_{k=1}^{l} -(\frac{N_k}{N}) \log_2 (\frac{N_k}{N})$

   $= \sum_{k=1}^{l} -p_i \log_2 p_i,$
   where $p_k$ is the *a priori* probability of the $k^{th}$ class.

2. *Select a feature to serve as the root node of the decision tree*

(a) for each feature $F_i, i = 1, 2, ..., n$, partition the original population into two sub partitions according to the values $a_{ij}$ (j=0 or 1, stands for the feature value 0 or 1) of the feature $F_i$. There are $n_{ij}$ patterns in $a_{ij}$ branch, but these patterns are not necessarily of any single class.

(b) for any branch population $n_{ij}$, the number of patterns belonging to class $C_k$ is $n_{ij}(k)$. Evaluate the entropy of the branch using

$$Ent(I, F_i, j) = \sum_{k=1}^{l} -(\frac{n_{ij(k)}}{n_{ij}}) \log_2 (\frac{n_{ij(k)}}{n_{ij}}).$$

The entropy of the system after testing on attribute $F_i$ is then $Ent(I, F_i) = \sum_{j=1}^{2} \frac{n_{ij}}{\sum_j n_{ij}} * Ent(I, F_i, j)$.

(c) The decrease in entropy as a result of testing feature $F_i$ is $\Delta Ent(i) = Ent(I) - Ent(I, F_i)$.

(d) Select a feature $F_k$ that yields the greatest decrease in entropy, that is for which $\Delta Ent(k) > \Delta Ent(i)$, for all $i$ = 1,2,..,l , $i \neq k$.

(e) The feature $F_k$ is then the root of the decision tree.

3. *Build the next level of the decision tree.*

   Select a feature $F_{k'}$ to serve as the level 1 node such that after testing on $F_{k'}$ on all branches we obtain the maximum decrease in entropy.

4. *Repeat step 1 through 3*

   Continue the process until all subpopulations reaching to a leaf node are of any single class or the decrease in entropy, *i,e,* $\Delta$ Ent is zero.

5. Prune the leaf node which contains no pattern point.

### 2.1.3 Handling Continuous Attribute

In order to treat numeric (continuous) attribute value many investigations have been done. Among them includes Fuzzy ID3 [8], RID3 [9], Neuro-fuzzy ID3 [8]. In Fuzzy ID3 the numeric range of the attribute is divided into several fuzzy intervals. The information gain $\Delta Ent$ is calculated by incorporating membership function of fuzzy sets. In that work the *a priori* probability is replaced by relative frequency. The relative frequency of class $C_k$ at node $b$ is defined by:-

$$p_k^b = \frac{|D_k^b|}{|D^b|},$$

where :

$D_k^b$is the data set whose elements belong to class $C_k$ at node $b$,

$D^b$is the data set at node $b$,

$| D_k^b |= \sum_{x \epsilon D_k} \prod_{(ij) \epsilon Q} \mu_{i,j}(x).$

$| D^b |= \sum_{x \epsilon D} \prod_{(i,j) \epsilon Q} \mu_{ij}(x)$

$\mu_{i,j}$ denotes membership value of the $i^{th}$ attribute to the $j^{th}$ fuzzy set.

$Q$ is the set of pair $(i, j)$ along the branch from root node to node $b$.

$D$ is the set of all data elements.

$D_k$ is the set of all data element belong to class $C_k$.

The entropy $Ent$, gain in entropy $\Delta Ent$, etc are calculated in the same manner as in original ID3.

In RID3 [9], first of all features are extracted using the fuzzy c-means (FCM) algorithm [10]. Then feature ranking is done with the principle that *good features should not show much variation within a class but should have sig-*

*nificantly different values for different classes.* After this, the sample space is partitioned to form a decision tree.

Ichihashi *et. al.* [8] proposed a slightly revised version of ID3. They find out the membership of each sampled data belonging to (all possible) pattern classes using a B-spline membership function of degree 3. Then they apply the fuzzy ID3 algorithm followed the ID3 algorithm with the modification that the decision will be taken after solving a nonlinear programming problem:-

$$\begin{gathered} max \ \ \sum_{k=1}^{l} -p_k \log_2 p_k \\ \text{subject to } \sum_{c \subset A_i} m(c) \leq \sum_{k \epsilon A_i} p_k (i \in I) \\ \text{and } \sum_{k \epsilon A_i} p_k = 1, p_k \geq 0, \end{gathered}$$

where $A_i$ is any subset of the universal set $X$ and $m(c)$ is the probability assignment to c. In one of his works Janikow [11] discussed a way to incorporate fuzziness to the decision tree to handle the continuous attribute case. His way of tree building procedure is the same as that of ID3. The only difference is that a training example can be found in a node to any degree. Wang *et.al* [12] in a very recent work proposed a scheme to handle the continuous case by discretizing the attribute. A continuous valued attribute is discretized during decision tree generation by partitioning the range into two intervals with the help of a threshold value T. A threshold value T for the continuous attribute $A$ is determined and the set $(A \leq T)$ is assigned to the left branch while the set $(A > T)$ is assigned to the right branch. This threshold is selected among several possible ones subject to the condition that the class entropy of the partition induced by the chosen point attains a minimum among all class entropies of different threshold choices. One disadvantage of this scheme is that in the process one attribute may make more than one decision in a path from the root to a leaf.

## 2.2 Artificial Neural Network

There was an explosive growth in the field of neural networks in the last two decades, because it is one of the best and highly useful techniques for

machine learning. An artificial neural network is represented by a labeled directed graph, where nodes perform some simple computation and each connection or link conveys a signal from one node to another, labeled by a number called the strength or weight of the link indicating the extent to which a signal is amplified or diminished by the link.
Before the neural net is able to give any decision, it must be trained with the training samples. In general there are two different learning paradigms [2, 3], namely supervised and unsupervised. In supervised learning the network has its output compared with known answers and receives a feedback about any error. On the other hand in the unsupervised learning scheme the network must discover for itself interesting categories or features in the input data. In our work we shall be dealing only with supervised learning mechanism. Among different network architectures the feed-forward network [2, 3] is the most important. The back-propagation algorithm [2, 3] is central to much current work on learning in neural networks. The algorithm gives a prescription for changing the weights $w_{pq}$ (weight on the link from node $p$ to node $q$) in any feed-forward network to learn a training set of input-output pairs. The basis of the algorithm is simply *gradient descent* which suggests changing each weight $w_{ik}$ by an amount $\Delta w_{ik}$ proportional to gradient of $E$ (the error) at the present location, *i,e,*

$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}},$$

where $\eta$ is a parameter called the *learning rate* of the network and its value lies between 0 to 1.

The weight $w_{ik}$ will be updated by the following equation:

$$w_{ik}^{new} = w_{ik}^{old} + \Delta w_{ik}.$$

The computation done at each node can be expressed by the sigmoid function.

$$g(h) \;\; = \frac{1}{1+exp(-2\beta h)},$$

where $\beta$ is called the *steepness factor* and is often set to 1 or $\frac{1}{2}$.

To avoid oscillation near some local minima some *inertia* or *momentum* is given to each connection weight. This scheme is implemented by giving a contribution from the previous time step to each weight change:

$$\Delta w_{ik}(t+1) = -\eta \frac{\partial E}{\partial w_{ik}} + \alpha \Delta w_{ik}(t),$$

where $\alpha$ is called *momentum parameter* and its value lies between 0 to 1.

### 2.2.1 Network Pruning

The above obtained network is pruned by doing the sensitivity analysis [13]. In this method we check which are the node active at any particular time. The nodes which are not active are pruned. The measure of relevance $\rho$ of any node is calculated by inducing a gating term $\chi$ for each unit such that

$$o_j = g(\textstyle\sum_i w_{ji} \chi_i o_i)$$

where $o_j$ is the activity of the node $j$. The important of the unit is then approximated by the derivative

$$\rho_i = -(\frac{\partial E^l}{\partial \chi_i}) \mid_{\chi_i = 1}$$

where $E^l = \sum(\mid t_{pj} - o_{pj} \mid)$
which is computed by back propagation.

When $\rho_i$ falls below certain threshold the unit can be deleted.
To suppress the fluctuation we use

$$\rho_i(t+1) = 0.8\rho_i(t) + 0.2\frac{\partial E^l}{\partial \chi_i}.$$

## 2.3 Algorithm for Discretizing Continuous Attribute

As mentioned in Section 1, ID3 algorithm for generating decision tree requires discrete valued attribute. If the pattern recognition problem characterized by continuous valued attribute, then the attribute must be discretized appropriately. Here in this section we describe a new way of discretizing the continuous attribute. Linguistic terms like low, medium and high are used in the process. This is done in two steps. In first step an n-dimensional feature space is translated to a 3n-dimensional feature by using three equidistant $\Pi$ membership functions [14, 15]. In step 2, 3n-dimensional features with continuous domain is translated to 3n-dimensional binary valued features by using appropriate threshold on each dimension. The detailed algorithm of the above mentioned steps are given below:
Given an n-dimensional pattern $F = [F_1, F_2, ...., F_n]$ , it is transformed into a 3n-dimensional vector as [14, 15]

$$F = [\mu_{low(F_1)}, \mu_{med(F_1)}, .....\mu_{hig(F_n)}] = [y_1, y_1, ...., y_{3n}]$$

Where $\mu$ value indicates the membership functions of the corresponding linguistic $\Pi$-sets low, medium, high along each feature axis. The algorithm to calculate the membership of the pattern point is:-

### calcInputMembership()

input : A data file $D_1$ consist of n-dimensional attribute, labeled data.
output : A data file $D_2$ consist of 3n-dimensional attribute, labeled data.

method :

for each attribute $F_j$ do
begin
find $F_{jmin} = min\{F_{1j}, F_{2j}, ...F_{kj}\}$

$F_{jmax} = max\{F_{1j}, F_{2j}, ...F_{kj}\}$,

where $k$ is the total number of samples in the input data file.

find mean $m_j$ of $\{F_{1j}, F_{2j}, ...F_{kj}\}$;

find mean $m_{jl}$ of the attribute whose value lies in $[F_{jmin}, m_j)$;

find mean $m_{jh}$ of the attribute whose value lies in $[m_j, F_{jmax})$;

Centers $C_{jm} = m_j, C_{jl} = m_{jl}, C_{jh} = m_{jh}$,

Radius $\lambda_{jl} = 2 * (C_m - C_l)$

$\lambda_{jh} = 2 * (C_h - C_m)$

$\lambda_{jm} = \frac{\lambda_l * (F_{jmax} - C_m) + \lambda_h * (C_m - F_{jmin})}{F_{jmax} - F_{jmin}}$,

$\Pi(F_j, C_{ji}, \lambda_{ji}) = 2 * (1 - \frac{||F_j - C_{ji}||}{\lambda_{ji}})^2$ ;for $\frac{\lambda_{ji}}{2} \leq ||F_j - C_{ji}|| < \lambda_{ji}$

$= 1 - 2 * (\frac{||F_j - C_{ji}||}{\lambda_{ji}})^2$; for $0 \leq ||F_j - C_{ji}|| < \frac{\lambda_{ji}}{2}$

$= 0$ otherwise.

(for $i$ = low, medium,high.)

$\mu_{i(F_j)} = \Pi(F_j, C_{ji}, \lambda_{ji})$.

end

## discretized()

input : data file $D_1$with n-dimensional attribute labeled data.

$[T_1, T_2, ...., T_{3n}]$3n-dimensional threshold vector.

output : data file $D_2$ with 3n-dimensional attribute with binary value.

method :

Get data file $D_3$ containing 3n-dimensional membership value by using algorithm *calcMembership()* with input file $D_1$.

for $i = 1$ to $k$ do (where $k$ is the total number of pattern points in data file $D_3$)

begin

for $j = 1$ to $3n$ do

begin

if $F_{ij} < T_j$

then write $F_{ij} = 0$ in file $D_2$

else write $F_{ij} = 1$ in file $D_2$

end

end

## 2.4 Rule Generation and Mapping onto the Neural Network

Now we are in a position to form a decision tree for any kind of data. Once the decision tree is ready, rules from the tree can be generated in *conjunctive normal form* by using the proposed algorithm. Before going into further detail let us describe related literature on *neural trees.*

### 2.4.1 Neural Trees

Both decision trees and neural networks are most commonly used tools for pattern classification. In recent years enormous work has been done in an attempt to combine the advantage of neural networks and decision trees. The new architecture so obtained is called a *neural tree.* The neural tree architecture reported in literature can be grouped according to the learning paradigm employed for their training. Most of the existing neural tree architecture were directly or indirectly related to feed-forward neural networks. In fact, the characterization neural tree was indistinguishably used to describe approaches using feed-forward neural network as a building element in order to improve the decision of decision tree, along with approaches employing decision tree as a tool for building and training feed-forward network.

In the first family of approaches, one attempts to develop a tree structure containing a feed-forward neural network in the nodes. Some of the remarkable work in this area are Sankar and Mammone's *neural tree network* (NTN) [16] and *competitive neural trees* by (CNeT) Behnke *et. al* [17]. The architecture of NTN consists of single layered neural network connected in the form of tree. On the other hand CNeT has a structured architecture and performs hierarchical clustering by employing unsupervised learning at node level.

In the second family of approaches attempt is made to build neural networks either by developing tree structured neural network or by mapping decision trees to multi layer neural network. Sethi [18] proposed a procedure for mapping a decision tree into a multi layered feed-forward neural network with two hidden nodes. The mapping rules described there can be stated as follows:

> The number of neurons in the first hidden layer equals to the number of internal nodes in the tree. Each of these neurons implements one of the decision function of the internal nodes.
>
> The number of neurons in the second hidden layer equals to the number of leaf nodes in the tree.
>
> The number of neurons in the output layer equals to the number of distinct classes.

This method of mapping the tree onto the neural network produces a net with very high complexity.

### 2.4.2 Proposed Mapping Scheme

The algorithm used for rule generation is as follows:

**Algorithm ruleGeneration()**
input : Decision tree.
output : Set of rules.
method:

for each path from root to leaf do
begin

```
rule = φ; current_node =root_node;
dowhile{current_node ≠ leaf_node}
  begin
     if the leaf node lies in the left subtree to the node having decisive feature F_i
        rule = rule∧ ¬F_i
     else rule =rule∧ F_i
  end
  assign the decision of the rule by the representative class of the node
  to which it takes the sample point.
  frequency of the rule is the number of pattern points reaching to the node
  in training data assign the frequency to the rule.
end.
```

Now we will illustrate our scheme of mapping the decision tree into the neural network by the following example.

Let the training set consist of 10 sample points, to be classified into two classes according to two continuous valued feature $F_1$ and $F_2$. These features must be discretized before being fed to the algorithm. By using the algorithm *discretized()* we get 6-dimensional features. $L_1, M_1, H_1, L_2, M_2, H_2$. With these feature let us generate a sample decision tree shown in Fig. 2.1.

The rule corresponding to the decision trees are:-

1. $\overline{L_1} \rightarrow C_1;\ 4$
2. $L_1 \wedge M_1 \wedge \overline{M_2} \rightarrow C_1;\ 2$
3. $L_1 \wedge \overline{M_1} \rightarrow C_2;\ 3$
4. $L_1 \wedge M_1 \wedge M_2 \rightarrow C_2;\ 1$

Note that the number to the right of a rule indicates the number of pattern points satisfying that rule. Each rule corresponds to one hidden node.

The corresponding mapped neural network is given in Fig.2.2:

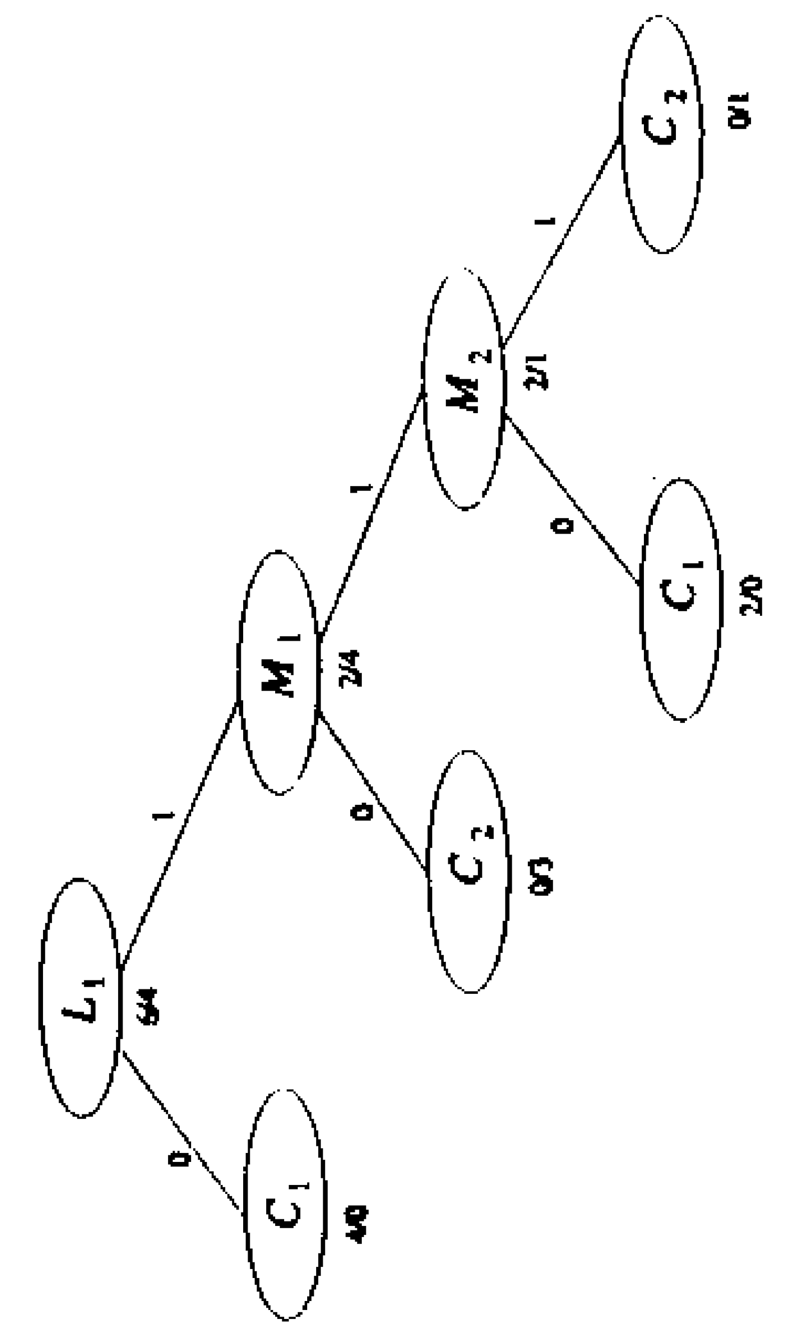

Figure 2.1: Decision tree

As class $C_1$ has two rules with frequencies 4 and 2, we choose 2 hidden nodes with initial output layer weights respectively,

$$\frac{4}{4+2} = \frac{2}{3}, \quad \frac{2}{4+2} = \frac{1}{3},$$

These percolate down to the weights in the input layer in proportion to the number of inputs attribute connected to that hidden node. Hence rule 1, with $\overline{L_1}$, provides a weight of $\frac{\frac{2}{3}}{-1}$ (taking account of the negative attribute $L_1$). Similarly the second rule of $C_1$ has 3 attribute. Hence the weights are $\frac{\frac{1}{3}}{3} = \frac{1}{9}, \frac{\frac{1}{3}}{3} = \frac{1}{9}, \frac{\frac{1}{3}}{-3} = -\frac{1}{9}$ respectively. The same holds for the class $C_2$. This corresponds to the initial weight encoding. Backpropagation is used to train the network further. Note that the remaining links not (specified by the rules) are initiated by very small random numbers. ♣

In the proposed algorithm we handle continuous valued attribute using linguistic labels. These are mapped onto a layered network. The weight encoding procedure has been explained with an example. However, some issues like handling of overlapping classes could not be tackled. To circumvent this problem we designed the fuzzy version of the algorithm, which is discussed in the next chapter.

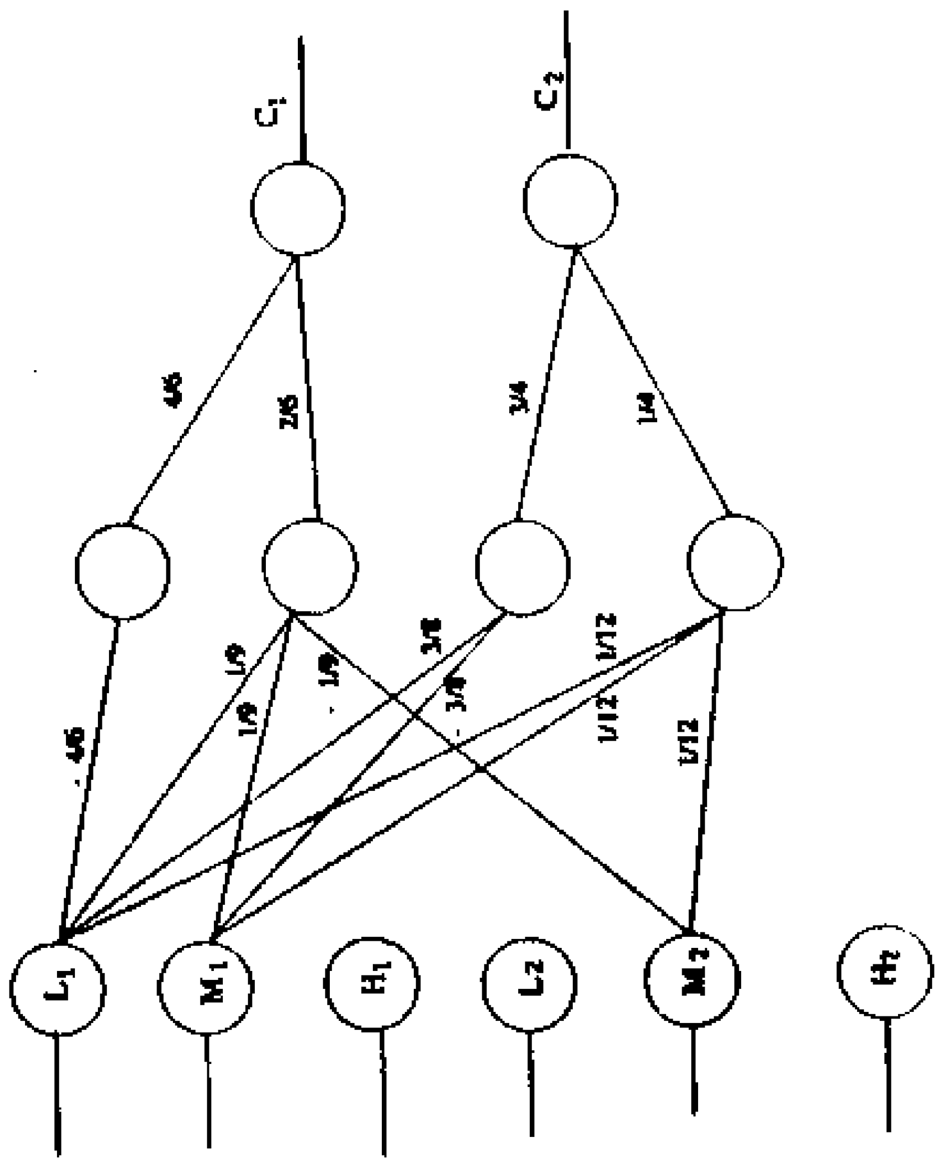


Figure 2.2: Mapped network

# Chapter 3

# ID3: Incorporation of Fuzziness

In Chapter 2 we have shown how the ID3 algorithm can be used in the continuous domain case by applying our proposed scheme. Another stringent requirement for ID3 is that, the classes must not be overlap. In case of overlapping, ID3 fails to give any decision. In this chapter we propose a revised version of the ID3 algorithm by incorporating fuzziness at the sample level and at the node level, such that the proposed algorithm will boil down to classical ID3 in the crisp case. Moreover when the classical ID3 fails to give any decision, our algorithm will provide more information about the overlapping area, this is useful while mapping the tree onto the neural network. This chapter consist of two sections. Section 1 deals with the incorporation of fuzziness into the ID3. Section 2 we describe the rule generation method. Section 3 provides in brief the method used to map the revised algorithm to a fuzzy MLP.

## 3.1 Incorporation of fuzziness

Let us concentrate on why ID3 fails to give any information when There are overlapping pattern classes. In ID3 algorithm we partition the sample space in the form of a tree by using attribute values only. When two sample points from two different overlapping classes lie in the intersection region, the corresponding features for these samples are expected to be the same. Which implies they will travel through the same path in the decision tree and finally land onto a same node. We cannot further split the node because for such a node the gain in entropy $\Delta Ent$ will be zero, which is one of the stopping criteria during tree building. Thus in the overlapped region an attribute value fails to provide any decision about the leaf node. In order to get more information we have to dig the data further. Intuition tells us that the pattern points of any particular class must be clustered around some characteristic prototype or cluster center. We want to exploit the fact that the points nearer to the cluster center have high proximity with the cluster center as compared to the points further from it. Once we know the cluster center, we can compute the distance of any sample point from that particular center. Then conclude this less is the distance, the more is the chance of the point belonging to that cluster. Taking these facts into account we compute the membership of each pattern into any of the class using the following algorithm [14, 15]

## calcClassMembership()

input: data file $D_1$ with n-dimensional attribute labeled data,
    number of classes : l
output : data file $D_2$ containing class membership of each pattern.
method :

for each class $C_k$ do
  begin
    Calculate $n$-dimensional vector $O_k = [O_{k1}, O_{k2}, .., O_{kn}]$ and $V_k = [V_{k1}, V_{k2}, .., V_{kn}]$
      where $O_{ki}$ is the mean of features i belonging to class $C_k$
      $V_{ki}$ is the standard deviation of features i belonging to class $C_k$
    Calculate the weighted distances of the training pattern $F_i$ from the class $C_k$ as:

$$Z_{ik} = \sqrt{\sum_{j=1}^{n} \left[\frac{F_{ij} - O_{kj}}{V_{kj}}\right]^2}$$

Calculate the membership of the $i^{th}$ pattern in the $k^{th}$ class as

$$\mu_k(F_i) = \frac{1}{1+(\frac{z_{ik}}{f_d})^{f_e}}$$

end

Once the membership is calculated we are in a position to decide whether we can make any decision or not. Mandal *et. al.* [19] provides a scheme to calculate the *confidence factor* (CF) and rule base to infer whether the sample point belongs to a particular class or the other. We adopt this notion of CF as:-

$$CF = \frac{1}{2}\left[\mu' + \frac{1}{l-1}\sum_{k=1}^{l}\{\mu' - \mu_k\}\right]$$

where $\{\mu_k\}$ are the class membership of the pattern to $k^{th}$ class, and $\mu'$ is the highest membership value. Note that we compute $CF_1$ and $CF_2$ (wlhen second choice is necessary), such that $\mu'$ refers to first and second membership values respectively.

Rulebase:- $CF_k$ denotes the CF corresponding to class $C_k$

1. If $0.8 \leq CF_k \leq 1.0$ then *very likely* class $C_k$ and there is no second choice.

2. If $0.6 \leq CF_k < 0.8$ then *likely* class $C_k$ and there is second choice.

3. If $0.4 \leq CF_k < 0.6$ then *more or less* likely class $C_k$ and there is second choice.

4. If $0.1 \leq CF_k 0,4$ then *not unlikely* class $C_k$ and there is no second choice.

5. If $CF_k < 0.1$ then *unable to recognize* class $C_k$ and there is no second choice.

In our case we calculate the CF corresponding to the class with highest and second highest membership values. In case of single choice (when rule 1 fires) we update the confidence factor to one and an aggregation has been made at the node level. This is given in the following algorithm. Fuzziness is incorporated at the node level by changing the decision function from classical

entropy to a *fuzzy measure* which is defined in the algorithm itself. Property of this function is that it gives more weightage to that attribute which has a higher discriminating power. The function is designed in such a way that when the class memberships are zero or one it will boil down to classical entropy. The detailed algorithm is given below:
In the training set we have $N$ pattern from $l$ classes $C_i, i = 1, 2, \ldots l$. The population in class $C_i$ is $N_i$, each pattern has $n$ attributes.

## rev_ID3()

input : Data file $D_1$ with $n$-dimensional attribute, labeled data.
output : Decision tree.
method :

1. Generate data file $D_2$ with discrete $3n$-dimensional attribute by using algorithm *Discretize()* with input file $D_1$.

2. Generate data file $D_3$ having class membership to each class by using algorithm *calcClassMembership()* .

3. *Calculate initial value of Fuzziness measure (FM)*

   After step 2 for the training set, class membership is known for all the patterns, also the class belonging of each pattern point is known. The initial FM for the system consisting of $N$ labeled pattern is :-

   $$FM(I) = \sum_{k=1}^{l} \left( \frac{1}{N} \sum_{m=1\&c=k}^{N} min(\mu_{mc}, 1-\mu_{mc}) - (\tfrac{N_k}{N}) \log_2 (\tfrac{N_k}{N}) \right)$$

   $$= \sum_{k=1}^{l} \left( \frac{1}{N} \sum_{m=1\&c=k}^{N} min(\mu_{mc}, 1-\mu_{mc}) - p_i \log_2 p_i \right)$$

   where $p_k$ is the *a prior* probability of the $k^{th}$ class and $\mu_{mc}$ denotes the membership of the $m^{th}$ pattern to $c^{th}$ class .

4. *Select a feature to serve as the root node of the decision tree*

(a) for each feature $F_i, i = 1, 2, ..., n$, partition the original population into two sub partitions according to the values $a_{ij}$($j$=0 or 1, stands for the feature value 0 or 1) of the feature $F_i$. There are $n_{ij}$ patterns in $a_{ij}$ branch, but these patterns are not necessarily of any single class.

(b) for any branch population $n_{ij}$, the number of pattern belonging to class $C_k$ is $n_{ij}(k)$. $\mu_{mkj}$ denotes the membership of the $m^{th}$ pattern in the $k^{th}$ class. Evaluate the FM of the branch using $FM(I, F_i, j) =$

$\sum_{k=1}^{l}\left(\frac{1}{n_{ij}}\sum_{m=1\&c=k}^{n_{ij}} min(\mu_{mcj}, 1-\mu_{mcj}) - (\frac{n_{ij}(k)}{n_{ij}})\log_2(\frac{n_{ij}(k)}{n_{ij}})\right)$ The FM of the system after testing on attribute $F_i$ is then $FM(I, F_i) = \sum_{j=1}^{2}\frac{n_{ij}}{\sum_j n_{ij}} * FM(I, F_i, j)$

(c) The decrease in FM, as a result of testing feature, $F_i$ is $\Delta FM(i) = FM(I) - FM(I, F_i)$

(d) Select a feature $F_k$ that yields greatest decrease in FM, that is for which $\Delta FM(k) > \Delta FM(i)$, for all $i$ = 1,2,..,l , $i \neq k$.

(e) The feature $F_k$ is then root of the decision tree.

5. *Build the next level of the decision tree.*

   Select a feature $F_{k'}$ to serve as the level 1 node such that after testing on $F_{k'}$ on all branches we obtain the maximum decrease in FM.

6. *Repeat step 3 through 5*

   Continue the process until all sub populations reaching to a leaf node are of any single class or decrease in FM, *i,e,* $\Delta$ FM is zero. Mark the node which have more than one class pattern point and which is a leaf node as a *unresolved* node.

7. (a) for each unresolved node calculate the confidence factors [19] $CF_1$ and $CF_2$ as:-

   for each pattern reaching to that node calculate
   $CF = \frac{1}{2}\left[\mu' + \frac{1}{l-1}\sum_{k=1}^{l}\{\mu' - \mu_k\}\right]$
   where $\{\mu_k\}$ are the class membership of the pattern to $k_{th}$ class.

   if $\mu' = max\{\mu_k\}$ then CF $= CF_1$ if $\mu' =$ second maximum of $\{\mu_k\}$ then CF $= CF_2$

   (b) Identify the class corresponding to which there is at least one $CF_1$ or $CF_2$ in the node.

   (c) for each pattern point in the node

      i. if $CF_1 \geq 0.8$ then put $CF_1 = 1, CF_2 = 0$
      and consider the class wise summation of th CF value of the classes found above and take the average.

      ii. Mark the class getting maximum and second maximum CF value.

      iii. Declare the node as the representative of the classes found in the above step with membership corresponding to the CF values.

8. Prune the leaf node which contains no pattern points.

## 3.2 Rule Generation

The algorithm ruleGeneration() described in Chapter 2 is modified for generating rules from the decision tree obtained by the algorithm rev_ID3.

### Algorithm rev_ruleGeneration()

input : Decision tree.

**output** : Set of rules.

**method: for** each path from root to leaf node dp

begin

*rule* = $\phi$; current_node =root_node;

dowhile{current_node $\neq$ leaf_node}

begin

if the leaf node lies in the left subtree to the node having decisive feature $F_i$

rule = rule$\wedge \overline{F_i}$

else rule =rule$\wedge F_i$

end

in training data, assign the frequency to the rule.

/* Frequency of the rule is the number of pattern points reaching to the node */

For the rule which takes the sample point to the leaf node and is

not marked in step 6 of the algorithm *rev_ID3()*,

assign the confidence of the rule as 1 and decision of

the rule by the representative class of the node .

For the rule which takes the sample point to the leaf node which

is marked in step 6 of the algorithm *rev_ID3()*,

assign the confidence of the rule by $CF_1$ and $CF_2$ and

the decision of the rule by the classes having these

confidence factors. Also assign the frequency of the

corresponding satisfying this rule.

end.

## 3.3 Mapping of Rules

Now we will illustrate our scheme of mapping the decision tree obtained from rev_ID3() into the neural network by the following example.

Let the training set consist of 15 sample points, to be classified into three classes according to two continuous valued features $F_1$ and $F_2$. These features must be discretized before being fed to the algorithm. By using the algorithm ***discretized()*** described in Chapter 2 we get 6-dimensional features: $L_1, M_1, H_1, L_2, M_2, H_2$. With these features let us generate a sample decision tree shown in Fig. 3.1.

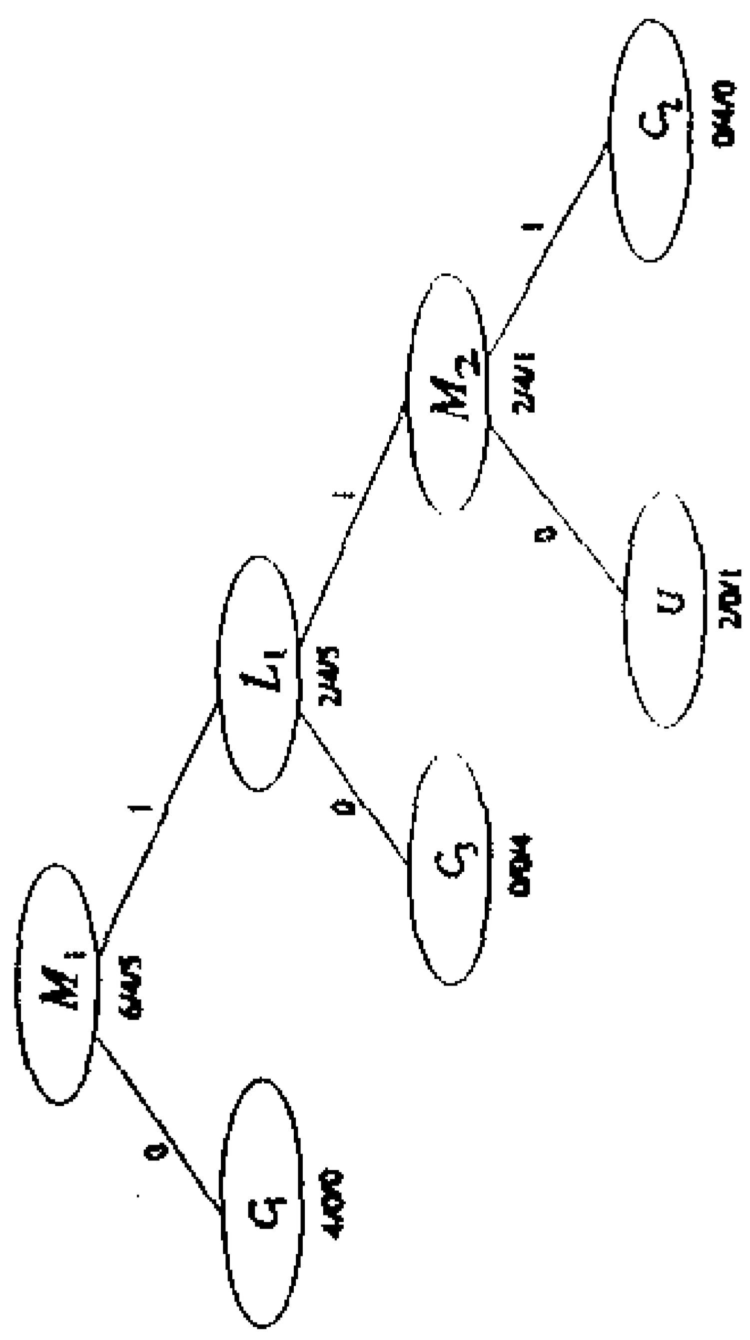


Figure 3.1: Decision tree generated by rev_ID3

Let, the data points in the unresolved node represented by "U" be $d_1, d_2, d_3$. The following table indicates the membership of these data points in the three different classes:

| | $C_1$ | $C_2$ | $C_3$ |
|---|---|---|---|
| $d_1$ | 0.80 | 0.60 | 0.02 |
| $d_1$ | 0.70 | 0.50 | 0.13 |
| $d_1$ | 0.16 | 0.90 | 0.10 |

The $CF_1$ and $CF_2$ (sample level) of $d_1, d_2, d_3$ corresponding to the classes with highest and second highest frequencies are given in the table below:

| | $CF_1$ | $CF_2$ |
|---|---|---|
| $d_1$ | 0.645 ($C_1$) | 0.395 ($C_3$) |
| $d_2$ | 0.54 ($C_1$) | 0.291 ($C_3$) |
| $d_2$ | 0.835 ($C_3$) | -0.09 ($C_1$) |

As row 3 has $CF_1 = 0.835$ ($\geq 0.8$ by step 7(c) of algorithm rev_ID3), hence $CF_1 = 1$ and $CF_2 = 0$ Aggregated CF for class $C_1$ is $(0.645 + 0.54 + 0)/3 = 0.365$ while that for class $C_3$ is $(0.395 + 0.291 + 1)/3 = 0.562$. As class $C_3$ has a higher membership we put $CF_1$ for node "U" as 0.562 for class $C_3$ and $CF_2 = 0.365$ for class $C_1$.

The rules corresponding to the decision trees obtained from rev_ID3 algorithm are:-

1. $\overline{M_1} \rightarrow C_1$; 4, 1 .
2. $M_1 \wedge L_1 \wedge M_2 \rightarrow C_2$; 4, 1.
3. $M_1 \wedge \overline{L_1} \rightarrow C_3$; 4, 1.
4. $M_1 \wedge L_1 \wedge \overline{M_2} \rightarrow$ 3, 0.562, 0.365, $C_3$, $C_1$, 1, 2.

Note that:

a) the two numbers to the right of a rules (1-3) indicate the number of pattern points satisfying that rule and the confidence of the rule respectively

b) in rule number 4 the entities after " $\rightarrow$ " indicate respectively the frequency of the rule, first and second confidence factors, classes corresponding to these confidence factors and the frequency of the sample belonging to these classes which satisfying this rule respectively.

The corresponding mapped neural network is given in Fig.3.2:

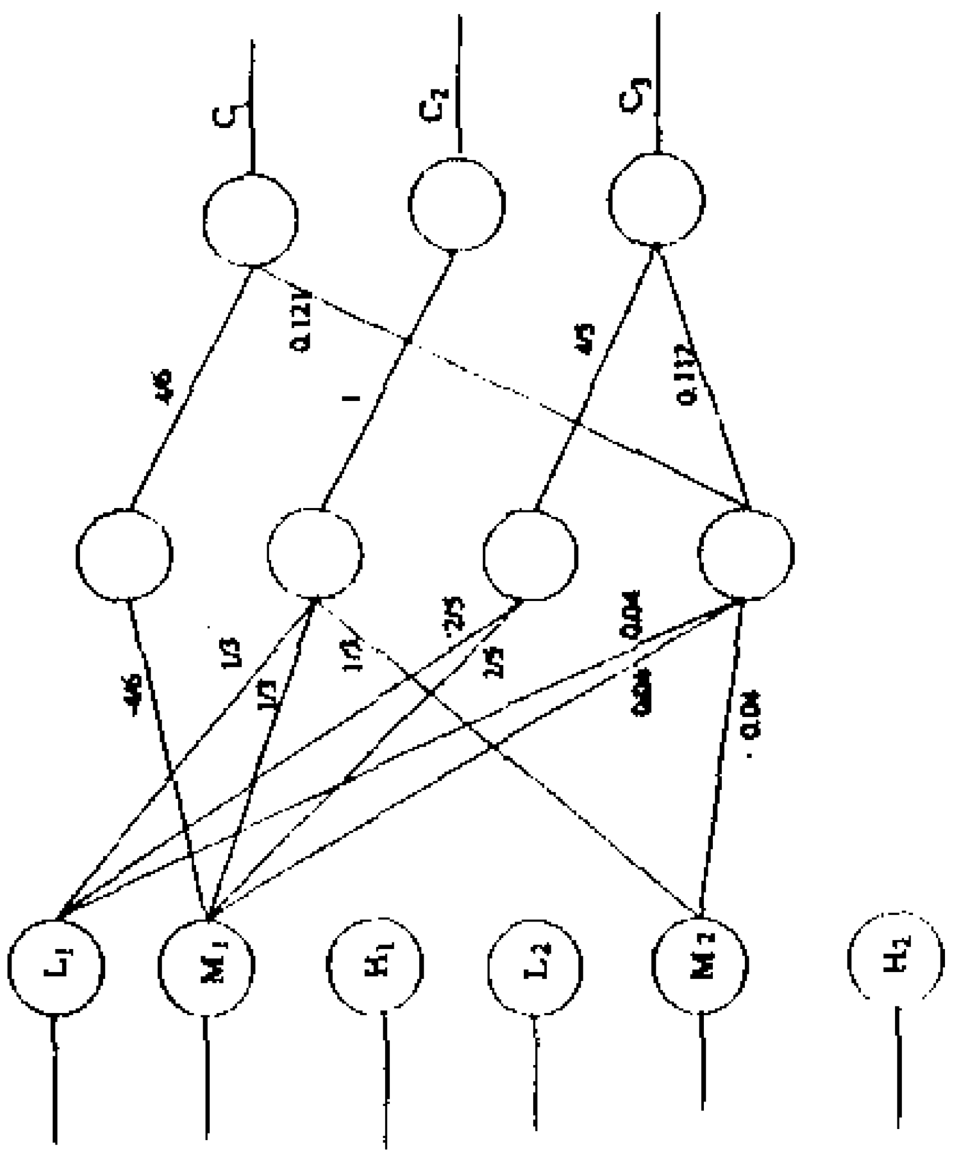

Figure 3.2: Mapped network

Since there are four rules generated in total, we have four hidden nodes. As class $C_1$ has two rules with frequency 4 & confidence 1 and frequency 2 & confidence 0.365, the output node will be connected to 2 hidden nodes with initial output layer weights

$$\frac{4}{4+2} * 1 \;= \frac{2}{3}, \;\; \frac{2}{4+2} * 0.365 \;= 0.121 \text{ respectively.}$$

If there is only one output node connected with any hidden node, the weight on the hidden-output link will percolate down to the weights in the input layer in proportion to the number of input attributes connected to that hidden node. Hence rule 1, with $\overline{L_1}$, provides a weight of $\frac{\frac{2}{3}}{-1}$ (taking into account of the negative attribute $\overline{L_1}$).

When there are more than one output nodes connected to a single hidden unit (corresponding to unresolved nodes) then the maximum of all the weights on the links from that node to all connected output nodes is taken. This will percolate down to the weights in the input layer in proportion to the number of inputs attribute connected to that hidden node. For example rule 4 shows that two output nodes denoting class $C_1$ and class $C_3$ will have a connection with one common hidden node with weights 0.121 and 0.112. Maximum of these is 0.121. Hence the weights are: $\frac{0.121}{3} = 0.04$, $\frac{0.121}{3} = 0.04$, $\frac{0.121}{-3} = -0.04$ respectively.

Now we train the MLP for the class membership of the pattern. Inferencing is done with the principle that the node which fires with higher confidence is the winner. ♣

In the proposed algorithm we are able to handle the cases overlapping classes by incorporating fuzziness to ID3. Rules are generated from the new decision tree. These are mapped onto a fuzzy MLP. The weight encoding procedure has been explained with an example. The following two chapters deal with the result and the concluding remarks.

# Chapter 4

# Results

Here we present some results demonstrating the effectiveness of the algorithm on a set of 871 Indian Telugu vowel sounds [14, 15]. As a comparison, the performance of the conventional MLP has also been provided.

The vowel sounds, collected by trained personnel, were uttered by three male speakers in the age group of 30 to 35 years, in a Consonant-Vowel-Consonant context. The details of the method are available in [14]. The data set has three features; $F_1$, $F_2$ and $F_3$ corresponding to the first, second and third vowel format frequencies obtained through spectrum analysis of the speech data. Note that the boundaries of the classes in the given data set are seen to be ill-defined (fuzzy). Fig. 4.1 shows a 2D projection of the 3D feature space of the six vowel classes $(a, i, u, e, o)$ in the $F_1 - F_2$ plane, for ease of depiction.

Tables 1-3 provide the performance of the networks initially encoded using ID3 and rev_ID3 (involving fuzziness). Comparison is made with the conventional MLP, initialized using random weights. Different training set sizes 10%, 20%, 30%, 40%, 50% are used. Recognition scores (classwise and overall) for both training and test set (100% -training set) are provided. The mean suqare error (mse) and number of training sweeps are also included. Classes 1-6 indicate the six vowel classes mentioned above. The mse is used

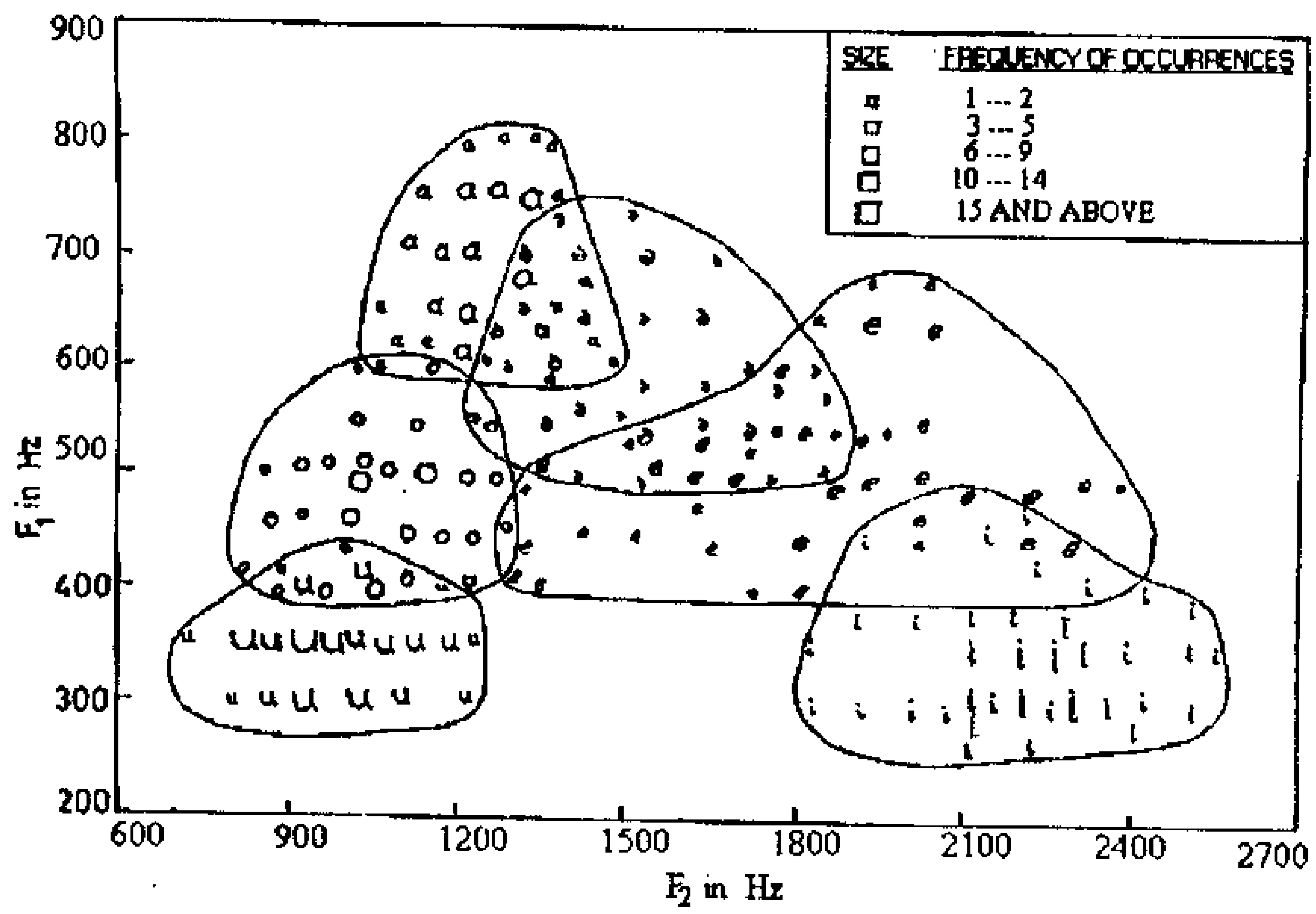


Figure 4.1: Vowel diagram in $F_1 - F_2$ plane

as the stopping criterion.

Tables 1 and 2 involve algorithm ID3 with no fuzziness. However, the frequency of the training pattern is taken into account while computing the initial weights. Table 1 considers 9 rules for the six classes. This is mapped to 9 hidden nodes. Table 2, on the other hand, considers 15 rules. Note that we consider only high frequency rule in Table 1. It is observed from Table 1 that the proposed algorithm gives better results, both in terms of recognition scores and number of training sweeps. This is because the encoding of prior knowledge results in a faster convergence. The gain in terms of recognition score is however not that explicit in Table 2 involving 15 hidden nodes.

Table 3 demonstrates the performance of the network using rev_ID3, that incorporates fuzziness in the decision tree. The use of confidence factor resulted in the generation of 25 rules. Here unresolved nodes of the decision tree were also taken into account. The performance is always better in terms of the number of training sweeps. The gain in terms of recognition score is not as evident as in Table 1. This leads to the conclusion that the smaller the network, more noticeable is the improvement of the proposed algorithm.

Table 4 shows the result using pruning. A training set size of 20% was used. It is observed that pruning results in smaller network when one uses knowledge encoding. However, the performance is not always better. More studies need to be made in this regard.

**Table 1:** Comparative performance with 9 hidden nodes, using knowledge encoding with ID3 and conventional MLP

| Weight Encoding | Training Set (%) | Training Scores(%) 1 | 2 | 3 | 4 | 5 | 6 | TOTAL | mse | # sweeps | Testing Scores(%) 1 | 2 | 3 | 4 | 5 | 6 | TOTAL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 10 | 50.0 | 85.7 | 93.7 | 85.7 | 68.4 | 100.0 | 83.5 | 0.0014 | 2550 | 59.0 | 81.7 | 85.9 | 92.0 | 71.2 | 86.5 | 80.9 |
| | 20 | 61.5 | 93.7 | 84.8 | 86.2 | 67.5 | 74.2 | 77.7 | 0.0014 | 939 | 74.6 | 78.1 | 84.9 | 91.0 | 74.8 | 79.3 | 80.8 |
| D | 30 | 40.0 | 88.0 | 78.0 | 86.3 | 75.4 | 73.5 | 75.9 | 0.0014 | 507 | 71.15 | 76.56 | 87.0 | 92.5 | 76.7 | 75.6 | 80.7 |
| | 40 | 40.7 | 76.4 | 82.0 | 84.7 | 76.5 | 80.2 | 77.0 | 0.0014 | 780 | 68.9 | 83.6 | 86.7 | 91.3 | 76.1 | 78.0 | 81.4 |
| 3 | 50 | 48.5 | 81.4 | 84.7 | 90.5 | 69.6 | 82.0 | 78.2 | 0.0014 | 451 | 67.6 | 84.8 | 85.1 | 92.2 | 81.9 | 75.8 | 82.2 |
| R | 10 | 50.0 | 85.7 | 87.5 | 85.7 | 73.7 | 94.1 | 82.2 | 0.0014 | 2632 | 56.0 | 85.3 | 84.6 | 91.2 | 74.4 | 84.0 | 80.9 |
| A | 20 | 53.8 | 93.7 | 84.8 | 86.2 | 67.5 | 71.4 | 76.5 | 0.0014 | 1771 | 57.6 | 79.4 | 84.9 | 91.8 | 72.4 | 75.7 | 78.4 |
| N | 30 | 40.0 | 88.0 | 78.0 | 84.1 | 70.5 | 77.3 | 75.1 | 0.0014 | 2491 | 59.6 | 79.7 | 86.1 | 90.6 | 76.0 | 74.8 | 79.3 |
| D | 40 | 48.1 | 79.4 | 82.1 | 83.0 | 75.3 | 80.3 | 77.3 | 0.0014 | 2560 | 75.6 | 83.6 | 86.7 | 89.1 | 74.6 | 78.0 | 81.2 |
| O | 50 | 51.4 | 81.4 | 84.7 | 90.5 | 67.6 | 80.9 | 77.8 | 0.0017 | 32300 | 64.6 | 84.8 | 85.0 | 87.0 | 78.1 | 75.8 | 80.1 |

**Table 2:** Comparative performance with 15 hidden nodes, using knowledge encoding with ID3 and conventional MLP

| Weight Encoding | Training | | | | | | | | | | Testing | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Training Set (%) | Scores(%) | | | | | | | mse | # sweeps | Scores(%) | | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 | TOTAL | | | 1 | 2 | 3 | 4 | 5 | 6 | TOTAL |
| I | 10 | 50.0 | 85.7 | 87.5 | 85.7 | 68.4 | 100.0 | 82.2 | 0.0014 | 970 | 66.7 | 85.3 | 84.6 | 89.0 | 70.7 | 83.4 | 80.4 |
| | 20 | 61.5 | 87.5 | 84.8 | 86.2 | 72.5 | 74.3 | 78.3 | 0.0014 | 554 | 61.0 | 79.4 | 84.1 | 91.8 | 73.0 | 80.0 | 79.6 |
| D | 30 | 40.0 | 92.0 | 78.0 | 84.1 | 73.8 | 77.3 | 76.3 | 0.0014 | 434 | 63.5 | 79.7 | 86.1 | 90.6 | 74 | 79.5 | 80.0 |
| | 40 | 44.4 | 79.4 | 82.1 | 81.3 | 75.3 | 80.2 | 76.7 | 0.0014 | 377 | 64.4 | 85.4 | 86.7 | 90.2 | 74.6 | 79.8 | 81.0 |
| 3 | 50 | 48.6 | 79.1 | 82.3 | 89.2 | 70.6 | 83.1 | 77.8 | 0.0014 | 267 | 64.9 | 84.8 | 85.1 | 89.6 | 79.0 | 75.8 | 80.8 |
| R | 10 | 50.0 | 85.7 | 87.5 | 85.7 | 68.4 | 100.0 | 82.2 | 0.0014 | 1256 | 59.1 | 82.9 | 84.6 | 89.0 | 74.5 | 85.9 | 80.9 |
| A | 20 | 61.5 | 93.7 | 84.8 | 86.2 | 75.0 | 74.0 | 79.5 | 0.0014 | 644 | 62.7 | 79.4 | 84.2 | 91.0 | 74.8 | 75.2 | 79.0 |
| N | 30 | 50.0 | 92.0 | 80.0 | 81.8 | 75.4 | 73.6 | 76.7 | 0.0014 | 542 | 69.2 | 78.1 | 86.9 | 90.6 | 74.0 | 78.7 | 80.4 |
| D | 40 | 40.7 | 79.4 | 82.0 | 84.7 | 72.8 | 80.2 | 76.4 | 0.0014 | 575 | 66.7 | 85.4 | 87.6 | 90.2 | 73.0 | 78.9 | 80.8 |
| O | 50 | 62.9 | 81.4 | 84.7 | 90.5 | 69.6 | 80.9 | 79.2 | 0.0014 | 444 | 75.6 | 84.8 | 86.2 | 89.6 | 76.1 | 75.8 | 81.2 |

**Table 3:** Comparative performance with 25 hidden nodes, using knowledge encoding with rev_ID3 and conventional MLP

| Weight | Training | Training | | | | | | | | | Testing | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Scores(%) | | | | | | | mse | # sweeps | Scores(%) | | | | | | |
| Encoding | Set (%) | 1 | 2 | 3 | 4 | 5 | 6 | TOTAL | | | 1 | 2 | 3 | 4 | 5 | 6 | TOTAL |
| R | 10 | 50.0 | 85.7 | 93.7 | 85.7 | 68.4 | 100.0 | 83.5 | 0.0014 | 739 | 65.1 | 84.1 | 85.3 | 91.2 | 75.0 | 77.9 | 80.6 |
| E | 20 | 67.8 | 80.8 | 83.4 | 92.6 | 77.2 | 76.5 | 80.5 | 0.0014 | 389 | 69.5 | 78.1 | 84.2 | 91.0 | 77.8 | 77.9 | 80.7 |
| V | 30 | 40.0 | 88.0 | 80.0 | 86.3 | 75.4 | 73.6 | 76.2 | 0.0014 | 291 | 67.3 | 79.7 | 86.9 | 92.5 | 76.7 | 75.6 | 80.7 |
| | 40 | 40.7 | 76.4 | 82.1 | 83.0 | 77.8 | 80.2 | 76.99 | 0.0014 | 251 | 71.1 | 85.4 | 85.7 | 90.2 | 77.0 | 78.0 | 81.6 |
| ID3 | 50 | 51.4 | 81.4 | 84.7 | 89.1 | 72.5 | 83.1 | 79.2 | 0.0014 | 180 | 67.6 | 84.8 | 83.9 | 88.3 | 83.8 | 75.8 | 81.7 |
| R | 10 | 50.0 | 85.7 | 87.5 | 85.7 | 68.4 | 100.0 | 82.2 | 0.0014 | 833 | 60.6 | 87.8 | 84.6 | 91.2 | 76.6 | 77.3 | 80.7 |
| A | 20 | 61.5 | 93.7 | 84.8 | 86.2 | 72.5 | 74.3 | 79.6 | 0.0014 | 445 | 61.0 | 78.0 | 85.6 | 91.8 | 74.2 | 77.9 | 79.5 |
| N | 30 | 40.0 | 88.0 | 80.0 | 84.1 | 75.4 | 77.4 | 76.7 | 0.0014 | 341 | 71.1 | 78.1 | 86.9 | 90.6 | 71.9 | 77.9 | 79.9 |
| D | 40 | 40.7 | 76.5 | 82.1 | 83.0 | 74.1 | 80.3 | 76.1 | 0.0014 | 267 | 71.1 | 85.4 | 88.6 | 92.4 | 73.0 | 78.0 | 81.5 |
| O | 50 | 57.1 | 83.7 | 84.7 | 90.5 | 69.6 | 80.9 | 79.0 | 0.0014 | 207 | 67.6 | 84.8 | 85.1 | 90.9 | 75.2 | 75.8 | 80.3 |

**Table 4:** Comparative performance using pruning

| Initial # Hidden Nodes | Weight Encoding | Final # Hidden Nodes | Training | | | | | | | | Testing | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Scores(%) | | | | | | | # sweeps | Scores(%) | | | | | | |
| | | | 1 | 2 | 3 | 4 | 5 | 6 | TOTAL | | 1 | 2 | 3 | 4 | 5 | 6 | TOTAL |
| 9 | **ID3** | 6 | 53.8 | 87.5 | 84.8 | 82.8 | 70.0 | 74.3 | 76.5 | 1348 | 59.3 | 69.9 | 84.9 | 88.5 | 75.4 | 78.6 | 78.3 |
| | **Random** | 9 | 53.8 | 93.7 | 84.8 | 86.2 | 67.5 | 74.3 | 77.1 | 286 | 52.5 | 79.4 | 85.6 | 92.6 | 76.6 | 75.2 | 79.1 |
| 15 | **ID3** | 7 | 53.8 | 93.7 | 84.8 | 89.7 | 72.5 | 74.3 | 78.9 | 1139 | 50.8 | 82.2 | 84.9 | 91.8 | 74.8 | 75.9 | 78.7 |
| | **Ramdom** | 7 | 53.8 | 93.7 | 84.8 | 82.8 | 77.5 | 74.3 | 78.9 | 3254 | 50.8 | 79.4 | 84.9 | 88.5 | 73.0 | 77.93 | 77.8 |
| 25 | **ID3** | 9 | 53.8 | 87.5 | 84.8 | 82.8 | 70.0 | 74.3 | 76.5 | 939 | 59.3 | 76.7 | 84.9 | 89.3 | 71.7 | 77.2 | 78.0 |
| | **Random** | 10 | 53.8 | 93.7 | 84.8 | 82.8 | 80.0 | 74.3 | 79.5 | 973 | 54.2 | 79.4 | 84.1 | 89.3 | 79.0 | 79.3 | 79.8 |

# Chapter 5

# Conclusions and Discussion

A novel method of using the ID3 algorithm to initially encode the connection weights of an MLP has been described. Continuous valued attributes are handled by the method. Crude rules are extracted from the data set using the decision tree-based approach. The rules are generated in linguistic terms, enabling a more natural representation. The frequency of the sample points, representative of a rule, is taken into account while mapping the rule onto the neural network.

Fuzziness is incorporated at the node level to tackle unresolved nodes. This novel method helps one to model real life ambiguous data involving uncertainty in the region of overlapping classes. The concept of confidence is used to arrive at a suitable decision. This scheme is directly mapped onto a fuzzy neural network architecture. Each rule corresponds to a separate hidden node.

Lucid examples are used to illustrate the mapping process. It is observed that the effectiveness of the algorithm becomes more evident in the smaller network. In other words, the fewer the number of hidden nodes, greater is the improvement shown by our algorithm as compared to that of conventional MLP involving random initial weights. This fact has also been established earlier by Banerjee *et. al.* [20]. In general, the performance is found to improve in term of both recognition scores and as the number of training cycles.

Using the fuzzy version of ID3 resulted in a larger network. Hence the improvement is less marked. However, that the mapping procedure used can be modified in the future to generate a more compact network architecture. This would help in highlighting the utility of this fuzzy version.

# Bibliography

[1] J. R. Quinlan, "Induction on decision trees," *Mach. Learning*, vol. 1, pp. 81–106, 1986.

[2] J. Hertz, A. Krough, and R. G. Palmer, *Introduction to the theory of Neural Computation*. New York: Addision-Wesley Publishing Company, 1991.

[3] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*. Massachusets: A Bradford Books, The MIT Press, 1997.

[4] B. D. Ripley, *Pattern Recognition and Neural Networks*. New York: Cambridge University Press, 1996.

[5] L. Breiman, J. H. Friedman, R. Olsen, and C. J. Stone, *Classification and Regression Trees*. CA: Wadsworth: Belmont, 1984.

[6] J. Tou and R. C. Gonzalez, *Pattern Recognition Principle*. Reading, Masschusets: Addision-Wesley Publishing Company, 1974.

[7] Y. H. Pao, *Addaptive Pattern Recognition and Neural Networks*. Reading, MA,: Addision-Wesley, 1989.

[8] H. Ichihashi, T. Shirai, K. Nagasaka, and T. Miyoshi, "Neuro fuzzy ID3: A method of inducing fuzzy decision trees with linear programming for maximizing enrtopy and algebraic methods," *Fuzzy Sets and Systems*, vol. 81, no. 1, pp. 157–167, 1996.

[9] N. R. Pal, S. Chakraborty, and A. Bagchi, "RID3: An ID3 like algorithm for real data," *Information Sciences*, vol. 96, pp. 271–290, 1997.

[10] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* New York: Plenum Press, 1981.

[11] C. Z. Janikow, "Fuzzy decision trees: Issues and methods," *IEEE. Trans. on Systems Man and Cybernetics*, vol. 28, no. 1, pp. 1–14, 1998.

[12] W. Xizhao and J. Hong, "On handling fuzziness for attribute in decision tree generation," *Fuzzy Sets and Systems*, vol. 99, no. 2, pp. 283 290, 1998.

[13] R. Reed, "Pruning algorithm - a survey," *IEEE. Trans. on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[14] S. K. Pal and D. Dutta Majumder, *Fuzzy Mathematical Approach to Pattern Recognition.* New York: John Wiley (Halsted Press), 1986.

[15] S. K. Pal and S. Mitra, "Multi-layer perceptron, fuzzy sets and classification," *IEEE Transactions on Neural Networks*, vol. 3, pp. 683–697, 1992.

[16] A. Sankar and R. J. Mammone, "Growing and pruning neural tree networks," *IEEE Trans. on Computers*, vol. 42, pp. 291–299, 1993.

[17] S. Behnke and N. B. Karayiannis, "Competetive neural trees for pattern classification," *IEEE. Trans. on Neural Networks*, vol. 9, no. 6, pp. 1352–1369, 1998.

[18] I. K. Sethi, "Entropy nets: From decision trees to neural networks," *Proc. of the IEEE.*, vol. 78, no. 10, pp. 1605–1613, 1990.

[19] D. P. Mandal, C. A. Murthy, and S. K. Pal, "Formulation of multivalued recognition systems," *IEEE. Trans. on Systems Man and Cybernetics*, vol. 22, no. 4, pp. 607–620, 1992.

[20] M. Banerjee, S. Mitra, and S. K. Pal, "Rough fuzzy MLP: Knowledge encoding and classification," *IEEE. Trans. on Neural Networks*, vol. xx, no. y, pp. 1–20, 1999.