A Novel Fuzzy First-Order Logic Learning System

TSE, Ming Fun

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

Supervised by:
Prof. LEUNG, Kwong Sak, Prof. KING, Kuo Chin Irwin

December 2001

Abstract

Conventional inductive learning methods cannot handle fuzzy information and noisy data. Several algorithms were developed in recent years to learn concept descriptions in fuzzy logic. These systems use a fuzzy propositional attribute-value language for describing entities and classification rules. The simplicity of this formalism allows such systems to deal with large volumes of data; however, it becomes inefficient when faced with complex objects and concepts. On the other hand, learning algorithms accepting descriptions of complex, structured entities and generating classification rules expressed in first-order logic were developed.

This thesis describes a novel learning system, code named FF01, that learns both fuzzy and first-order logic concepts from data. FF01 builds on the ideas from both fuzzy set theory and first-order logic. Object relationships are described using fuzzy relations based on which FF01 generates classification rules expressed in a special form of fuzzy first-order logic. The proposed learning system is an expansion to the FOIL system, which is notable in learning non-fuzzy first order logical concepts. The thesis first describes the fuzzy first-order logic language used in FF01, it includes the discussions of different literal forms and the fuzzyification of continuous variables. Then, we outline the system architecture. We there details the input data format, the Horn-clause formation loop, the preprocessing and the postprocessing procedures.

We start the research on the comparison of fuzzy sets. In particular, a novel learning algorithm, code named FF99 is developed to learn zeroth-order fuzzy

concepts. Some interesting observations are obtained when two fuzzy sets are compared through a method called the error function. The observations are summarized to be the nodal characteristics of fuzzy set comparisons. The nodal characteristics are then quantified by the continuous information theory to become a heuristics to guide the search of literals in FF99. The whole FF99 acts as one of the options of the learning algorithms in FF01, but it works only if we are learning zeroth-order fuzzy concepts.

In order to fully utilized the representation language in FF01, we continue to explore other methods that enable us to learn fuzzy first-order concepts. FF99 fails in first-order learning because every literal is evaluated globally. A partial evaluation method of literals is needed to distinguish the samples into the "covered" and the "uncovered" set. The distinction enables us to prune the covered samples in the next step of learning. We will show that an effective pruning is crucial in first-order learning. FF01 first employ the α-covering as the partial evaluation method. Then we enhance it into a new adaptive α-covering method.

We test FF99 with several databases from UCI Machine Learning Repository. For instances, in the Iris classification experiment, FF99 gets an average accuracy of 97%. It also achieves classification accuracy of 88% in the Credit Approval experiment, which is higher than that got from C4.5. FF01 also works fine in relational databases as mentioned in FOIL. Finally, we construct some examples to demonstrate the unique fuzzy first-order learning power of our system.

In conclusion, FF01 induces fuzzy linguistic concepts that are comprehensive to human beings. Also, it provides a satisfactory way to handle fuzzy relational data. The learning algorithm is robust and efficient. Finally, the experiment results have shown that the novel learning system is suitable for learning knowledge in fuzzy first-order logic from both continuous and discrete variables, and is also very tolerant to noisy data.

傳統的歸納學習不能處理模糊及混亂資料。近年來，有多個演算法被發展去學習模糊邏輯的概念。這些系統使用模糊命題邏輯來形容個身及分類規則。這個簡單的方法容許它們處理大數量的資料。另一方面，有一些學習演算法可以接受複雜，結構法的個體來產生一階邏輯的分類規則。

但是，從來沒有一個學習系統可以使用模糊一階邏輯來形容概念。本論文發展了一個系統，名叫 FF01，來學習模糊一階邏輯。在發展 FF01 之前，我們先發展了一個名爲 FF99 的系統，用來學習零階的模糊概念。當我們試用一個名爲錯誤函數的方法來比較兩個模糊集，我們獲得一些有趣的觀察。這些觀察被總結爲模糊集比較的節點特徵。使用連續信息理論去量化這些節點特徵後會得出一個啓發以用來引導 FF99 對字面的搜尋。

爲了盡量利用 FF01 豐富的知識代表，我們繼續去發掘其他有效的方法。當中最重要就是如何把布爾的覆蓋方法模糊化。我們嘗試了三個方法：α 覆蓋，調整性 α 覆蓋和或然率覆蓋。使用這些方法容許我們處理一階邏輯的大量資料並且保持高度的學習精確率。

我們使用了多個數據隻來測試 FF99 及 FF01 的精確度，結果証明它們比 C4.5 更準確。同時，它能獲得與 FOIL 一樣的一階邏輯結果。我們再引入了一個美國職業籃球的數據隻來驗證 FF01，實驗証明它在現實的資料發掘也能派上用場。

總括來說，FF01 是一個有效及穩健的學習系統，它能夠處理零階及一階的資料，加上對模糊邏輯的支持，它是一套創新及實用的學習系統。

Acknowledgements

Thanks for the guidance of my supervisors. Especially, I thank Prof. K.S.Leung for his advice on my time and personal management. Also, I am grateful to Prof. Irwin King for teaching me how to research. Finally, I must thank my parents, my sister and all my friends who support me for long.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning has long been one of the main research areas in artificial intelligence. Some successful learning systems have been developed in past few decades. They differentiate between each other mainly on the knowledge representations adopted and different learning algorithms are designed for that particular representation language. For example, the well-known ID3 algorithm induces classification models in form of decision trees, while the back-propagation learning algorithm is especially designed for multi-layer neural network models. However, among the existing learning methods, none of them are capable to induce fuzzy and relational concepts at the same time. That motivates us to start our research.

1.1 Problem Definition

The goal of this thesis is to achieve the following statements:

Powerful representation language: We are seeking a representation language that enables us to express *inexact* and *relational* concepts at the same time.

Effective learning algorithm: The learning algorithm used by our system should be tractable in order to handle large volume of data. This concern is especially importance when we try to induce concepts from relation data, in which the rate of growth of tuples is fierce.

Compatible to existing induction tasks: FF01 should be able to induce concepts from existing well-known dataset. Also, it should be capable to handle the induction tasks tackled by some benchmark learning systems, e.g. C4.5 and FOIL.

1.2 Contributions

After setting our goals, we seek the solutions. They will be discussed in detail in the following chapters. Here we summarize the main contributions of this thesis:

Fuzzy first-order logic learning system: FF01 is a new learning system to use the fuzzy first-order logic language to represent and learn fuzzy concepts. The novel use of this language enables us to induce and describe fuzzy relational concepts. The thesis will show that the language acts as a superset of some other conventional logical languages, e.g. the zeroth-order fuzzy logic and the crisp first-order logic language. Thus, our system is backward compatible to existing induction tasks, e.g. the simple but large-volume attribute-value induction tasks, or the complex and abstract relational induction tasks. Moreover, the language allows us to handle a new area of induction problems.

Automatic generation of fuzzy concepts: Users may specify the fuzzy concepts artificially, or, more naturally, users could specify the continuous data in a simple attribute-value manner. The system will automatically generates fuzzy literals based on the distribution of the input data. And fine tuning of the fuzzy sets will be done in the postprocessing stage, in order to ensure that the fuzzy literals are best describing the target relation.

Extension to FOIL: FF01 also acts as an extension to the FOIL system. FF01 is capable to read the input data in FOIL's format (*.d). By enriching the original input data format of FOIL, FF01 allows the user to define fuzzy relations in input. Our system also supports some of the opinions available in FOIL, for example, we could limit the system to exclude negated literals. The whole FF01 system is a complete software

package.

New fuzzy set comparison method: We discover that under some assumptions, the subtraction of the membership functions of two fuzzy sets will result in some interesting observations. We summarized the observations as the nodal characteristics of fuzzy set comparisons. A sub-system called FF99 makes use of the nodal characteristics to search for literals in the construction of zeroth-order fuzzy concepts.

Enhanced covering method: The conventional covering method in machine learning is very effective in pruning the search space. However, the method only works for Boolean data. We will enhance the simple α-covering method, which is the simplest method to allow fuzzy covering. The enhanced method is called the adaptive α-covering method, which is shown to be very robust and could be applied in any fuzzy induction systems.

1.3 Thesis Outline

We start our story by reviewing some relevant literatures in **Chapter 2**. The review involves several areas in computer science, in particular, we will give a overview of different methods in representing inexact knowledge. As FF01 is a machine learning system, we will then review different paradigms of machine learning. After that, we highlight some of the learning systems that are directly related to our system. Finally, we give a section of the mathematical background of fuzzy logic, which will be often referenced by later chapters. We start to describe our system in **Chapter 3**. The usefulness and the uniqueness of the fuzzy first-order logic will be presented. We will also discuss the treatment of continuous variables and the literal forms supported by FF01. The next section outline the system architecture of FF01, which includes the

enriched input data format of FOIL, the Horn-clause formation loop, the preprocessing and the postprocessing stages. **Chapter 4** details the sub-system FF99, which learns fuzzy zeroth-order concepts. We will first review some existing closeness measures between fuzzy sets. Then we introduce the error function and the observations on the shape of it. The nodal characteristics will then be discussed in details. Next, we quantify the nodal characteristics by the continuous information theory in order to form a *COST* heuristics. The heuristics is shown to be very effective through the illustration of an example. In **Chapter 5**, we first discuss the importance of covering in machine learning and the difficulties in applying the covering concept in fuzzy systems. The α-covering method is adopted to be the primitive solution. Then, we continue to develop a more robust adaptive α-covering method based on the α-covering method. But these two methods are still not enough, we go on to develop a probabilistic-covering method. Finally, we will show in **Chapter 6** that our system is functionally more advanced than existing learning systems. It is illustrated by the comparing the experiments that are already tested by C4.5 and FOIL. Also, we would demonstrate some experiments that could be handled by our system but not our learning systems. Finally, we will summarize and discuss the whole system and some suggestions on possible improvements of our system will be given in **Chapter 7**.

Chapter 2

Literature Review

As our research involves several areas in computer science, we are going to review some of the related issues. First, we give an overview of how could we represent inexact knowledge. Then, we turn to discuss different paradigms of machine learning. Particularly, we will highlight some of the related learning algorithms. Finally, a brief reference on the notation and convention of fuzzy logic will be given.

2.1 Representing Inexact Knowledge

Inexact information is natural in human thinking and reasoning. This kind of inexact information should be formulated so that it can processed and inferred by machines. This section gives an overview of the nature of inexact knowledge and several approaches to formulate inexact knowledge into machine-understandable form.

2.1.1 Nature of Inexact Knowledge

Inexact knowledge normally refers to the information that contains *uncertainty* or *imprecision.*

Uncertainty occurs when one is not absolutely certain about a *fact* or a *proposition.* It arises mainly because of the incompleteness and incorrectness of information. For example, in a medical diagnosis system:

"John has a 39^oC fever" — uncertain fact

"`if` John has a 39^oC fever `then` he has a flu" — uncertain proposition

On the other hand, imprecision occurs when the information contains terms with *vague* meaning. And almost every adjectives and adverbs in *natural languages*, that which is used by ordinary people on a daily basis, represent information with vague meaning.

For example, in the sentence:

"John has a slight fever" — the adjective 'slight' is vague in natural and we call this sentence contains imprecise information.

2.1.2 Probability Based Reasoning

As discussed in **Section 2.1.1**. Uncertain knowledge is typical in the medical domain, as well as most other judgement domains: laws, business, design, automobile repair, and so on. The agent's knowledge can at best provide only a *degree of belief* in the relevant sentences. One main tool for dealing with degrees of belief is *probability theory*, which assigns a numerical degree of belief between 0 to 1 to sentences. Probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance. And the probability, that an agent assigns to a proposition depends on the percepts that is has received, is called *evidence*.

Basic Probability Theory

It is normal to use a small set of axioms that constrain the probability assignments that an agent can make to a set of propositions. There are three most importance axioms:

1. All probabilities are between 0 and 1.

$$0 \leq P(A) \leq 1 \tag{2.1}$$

2. Necessarily true (i.e. valid) propositions have probability 1, and necessarily false (i.e. unsatisfiable) propositions have probability 0.

$$P(True) = 1 \qquad P(False) = 0 \tag{2.2}$$

3. The probability of a disjunction is given by

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B) \tag{2.3}$$

A and B are called ***random variables*** which are propositions including equalities. The first two axioms serve to define the probability scale. Altogether with the third axiom, we can derive all other properties of probabilities.

One of the most important results of the probability theory is *Bayes' theorem.* Bayes' results provide a way of computing the probability of a hypothesis following from a particular piece of evidence, given only the probabilities with which the evidence follows from actual causes (hypothesis). Bayes' theorem states:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{2.4}$$

Where:

$P(A)$ and $P(B)$ are the ***unconditional*** or ***prior probabilities*** that the proposition A and B is true respectively.

$P(A|B)$ and $P(B|A)$ are the ***conditional*** or ***posterior probabilities.*** $P(A|B)$ is read as "the probability of A given that *all we know* is B." Vice versa.

On the surface, Bayes' rules in (2.4) does not seem very useful. It requires three terms — a posterior probability and two prior probabilities — just to compute one posterior probability. However, Bayes' rule is very useful in practice because there are many cases where we do have good probability estimates for these three numbers and need to compute the fourth. As in the medical diagnosis task, we often have posterior probabilities on causal relationships and want to derive a diagnosis.

There are two main assumptions for the use of (2.4): 1. All $P(A)$, $P(B)$ and $P(A|B)$ are known. 2. $P(A|B)$ is independent of the changes in $P(A)$ and $P(B)$. Actually, these two assumptions, especially the second one, are *difficult to satisfy.* In general, and especially in areas such as medicine, such assumption of independence cannot be justified. However, if these assumptions are met, Bayesian approaches offer the benefit of a *mathematically well founded*

and statistically correct handling of uncertainty.

Bayesian Network

A ***Bayesian network***, which is the same as ***belief network***, ***probabilistic network***, causal network and knowledge map, is a data structure used to represent the dependence between random variables and to give a concise specification of the joint probability distribution. A Bayesian network is a graph in which the following holds:

1. A set of random variables makes up the nodes of the network.

2. A set of directed links or arrows connects pairs of nodes. The intuitive meaning of an arrow from node X to node Y is that X has a *direct influence* on Y.

3. Each node has a conditional probability table that quantifies the effects that the parents have on the node. The parents of a node are all those nodes that have arrows pointing to it.

4. The graph has no directed cycles (it is a directed, acyclic graph, or DAG)

The use of Bayesian network has shown up in several areas of AI research, including pattern recognition and classification problems. The first expert system using Bayesian networks was CONVINCE [1]. More recent related system includes the PATHFINDER system for pathology analysis [2].

Bayesian network helps us to capture uncertain knowledge in a natural and efficient way. Unfortunately, it *cannot* handle imprecise information. Also, many system domains do not meet the assumptions of Bayes' rule and must reply on more heuristic approaches.

2.1.3 Certainty Factor Algebra

The problems in the application of Bayes' rule led researchers to search for other measures of "confidence". Probably the most important alternative approach was the ***Certainty Factor Algebra***. Certainty factors were invented for use in the medical expert system MYCIN [3], which was intended both as an engineering solution and as a model of human judgement under uncertainty.

Certainty theory is based on a number of observations. The most important one is that, in traditional probability theory the sum of *confidence for* a proposition and *confidence against* the same proposition must add to 1. However, it is often the case that an expert might have confidence 0.8 (say) that a proposition is true and have no feeling about it being not true. The case is best described by the ***measure of belief*** (MB) and ***measure of disbelief*** (MD):

$$\begin{aligned} \text{either}: &\quad 1 \geq MB(H) \geq 0 \text{ while } MD(H) = 0 \\ \text{or}: &\quad 1 \geq MD(H) \geq 0 \text{ while } MB(H) = 0 \end{aligned}$$

The two measures constrain each other in that a given piece of evidence is either for or against a particular hypothesis (H). This is an important difference between certainty theory and probability theory. Once the link between measures of belief and disbelief has been established, they may be tied together again with the certainty factor calculation:

$$CF(H) = MB(H) - MD(H) \tag{2.5}$$

As the certainty factor (CF) approaches 1, the evidence is stronger for a hypothesis; as CF approaches -1, the evidence against the hypothesis gets stronger; and a CF around 0 indicates that little evidence exists either for or against the hypothesis.

The certainty theory can be formally applied in *rule-based systems*. In which each premise can be associated with a CF. When we logically combine

two premises by $\wedge$ (logical `and`) or $\vee$ (logical `or`) operations, the CF of the premises are combined to produce a CF for the overall premise in the following manner:

For premises $P1$ and $P2$,

$$CF(P1 \wedge P2) = min(CF(P1), CF(P2))$$
$$CF(P1 \vee P2) = max(CF(P1), CF(P2))$$

The combined CF of the premises is then multiplied by the CF of the $ruletogettheCF$ of the conclusion of the rule. For instance:

$$\begin{aligned} &P1 && (CF = 0.6) \\ &P2 && (CF = 0.4) \\ &P3 && (CF = 0.2) \\ &\texttt{if } (P1 \wedge P2) \vee P3 \texttt{ then } R && (CF = 0.7) \end{aligned}$$

Then we get the conclusion:

$$R \qquad (CF = 0.28 = 0.7 \times max(min(0.6, 0.4), 0.2))$$

Now suppose another rule fire the same conclusion R with $CF = 0.3$, the two CF are combined to give an overall estimate of the certainty following the calculations:

$$CF = \begin{cases} CF_1 + CF_2 - (CF_1 \times CF_2) & : \quad CF_1 \text{ and } CF_2 \text{ are +ve} \\ CF_1 + CF_2 + (CF_1 \times CF_2) & : \quad CF_1 \text{ and } CF_2 \text{ are -ve} \\ \frac{CF_1 + CF_2}{1 - min(|CF_1|, |CF_2|)} & : \quad \text{otherwise} \end{cases} \tag{2.6}$$

Finally, the combined certainty of R is increased:

$$CF = 0.3 + 0.28 - (0.3 \times 0.28) = 0.496$$

The certainty factor approach is criticized, as it is not as rigorously founded as in formally probability theory. However, it does not require a large volume of statistical data and could be used in many domain areas. So, it is widely adapted in many expert systems.

2.1.4 Fuzzy Logic

The classical ***Boolean Logic*** is short in accurate modeling of real life problems, as it cannot handle *imprecision* and *uncertainty* information. Early in 1920, Lukasiewicz, who was an Ireland mathematician, developed the ***Three-valved Propositional Calculus*** and worked on ***Many-valued Logics***. Consequently, many mathematics theories were developed fro the tolerance of imprecision and uncertainty in the past few decades.

The basic idea of many-valued logic has been explored to some extent by a number of mathematicians in the past century, but the real breakthrough was made by Prof. Lotfi Zadeh. In 1965, he published a paper of the theory of ***Fuzzy Sets*** [4]. Unlike the certainty algebra or the Bayesian network, which require knowledge to be represented in specific forms (production rules or network, respectively), fuzzy set theory is an *extension* of the original set theory. So that, fuzzy set theory can be used in a large variety of applications. Our system, FF99, cites fuzzy set theory in knowledge representation. A more complete review of related elements in fuzzy logic is given in **Section 2.4**.

2.2 Machine Learning Paradigms

Machine learning (ML) is important for practical applications of artificial intelligence. Traditionally, it is used to tackle the "knowledge engineering bottleneck", which is the major obstacle to the widespread use of expert systems [5]. And now, machine learning is applied in more domains such as data mining and control systems. This subsection generally discusses the nature and difference between various machine learning paradigms, particularly the way they capture the knowledge and the way they search through the solution space.

2.2.1 Classifications

Any machine learning method belongs to either the class of ***supervised learning*** or the class of ***unsupervised learning***. The reliance of *training* instances of *known classification* defines the task of supervised learning. While unsupervised learning addresses how an intelligent agent can acquire useful knowledge in the *absence of artificially classified training data.* Regression, category formation, and conceptual clustering are fundamental problems in unsupervised learning.

Supervised learning systems can be categorized into ***knowledge-based system*** or ***non knowledge-based system***. The former approach models learning as the acquisition of explicitly *represented* domain knowledge. Based on its experience, the learner constructs or modifies expressions in a formal language, such as propositional logic, and retains this knowledge for future use. In contrast, learning in Bayesian network or neural network do not acquire knowledge (the learned information) in a symbolic language.

Among the supervised learning methods, some of them study learning with an emphasis on ***inductive inference*** and other are primarily interested in ***deductive inference***. Using inductive inference, general conclusions are drawn on the basis of some particular examples. On the other hand, in deductive inference, the conclusions follow necessarily from the axioms according to specified rules of inference. Generally, inductive inference relies on large number of examples to define the essential properties of a general concept. Algorithms that generalize on the basis of patterns in training data are referred to as ***similarity-based***. In contrast to similarity-based methods, ***explanation-based methods*** use prior knowledge of the domain to guide generalization. The explanation-based learning (EBL) makes use of deductive inference. It uses an explicitly represented domain theory to construct an explanation the training examples; in which the explanation usually *proves* that the training

examples follows the theory logically. By generalizing from the explanation of the instances, rather than from the instances themselves, EBL organizes training data into a systemic and coherent structure. Mitchell [6] developed Explanation-Based Generalization algorithm, while DeJong and Mooney [7] developed a variation of EBL in 1986.

2.2.2 Neural Networks and Gradient Descent

We know that a brain can perform a complex task — recognize a face, for example — in less than a second. We also know that a serial computer requires ten (or even thousand) times of operation to perform the same task. Altogether with other advantages of brain such as fault tolerance and the highly adaptive learning ability, computer scientists were motivated to imitate the brain using computers. Neural network is such a mathematical model of the operation of brain.

Basic Concepts

A neural network is composed of a number of ***nodes***, connected by ***links***, in which the nodes and links model the neurons and nerve fibers in brain respectively. Each link has a numeric ***weight*** associated with it. Weights are the primary means of *long term storage* in neural networks, and learning usually takes place by updating the weights. Some of the units are connected to the external environment, and can be designated as ***input*** or ***output nodes***. The weighted are modified so as to try to optimize the performance of the whole input/output blackbox as in **Fig. 2.1**. Each node has a set of input links from other nodes, a set of output links to other nodes, a current ***activation level***, and a means of computing the activation level at the next cycle. **Fig. 2.2** shows a typical node. Each node performs a simple computation: it receives signals from its input links and computes a new activation level that it sends

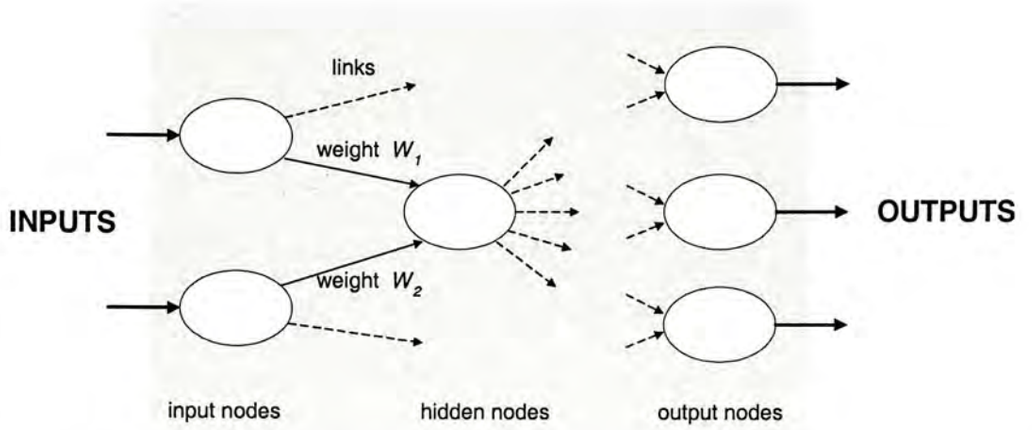


Figure 2.1: Neural network as a blackbox of inputs/outputs mapping

along each of its output links. The computation involves two components. First is the *input function* that computes the weighted sum of the node's input values. Second is the ***activation function***, g, that transforms the weighted sum into the output activation value, a_i, in next cycle:

$$g(\sum_j W_{j,i} \times a_j) \longrightarrow a_i \tag{2.7}$$

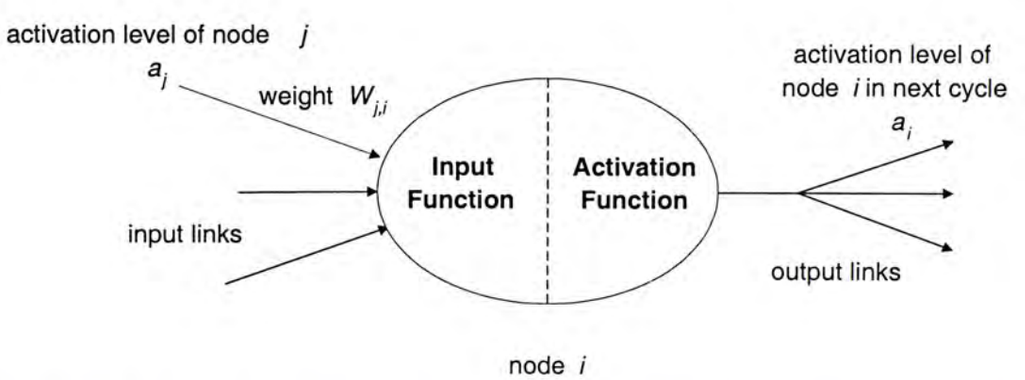


Figure 2.2: A node consists of an input function and an activation function

Note that the input function is linear while the activation function is normally *non-linear*. Some basic choices of the activation function are the step function and the sigmoid function.

Learning Mechanisms

There are a variety of kinds of network structure, each of which results in very different computational properties. The main distinction to be made is between ***feed-forward*** and ***recurrent*** networks.

A feed-forward network is a direct acyclic graph (DAG) and usually it is arranged in *layers*. In a layered feed-forward network, each node is linked only to nodes in the next layer, none in another layers. Because of the lack of cyclic network path, the activation from the previous time step plays no part in the computation. Hence, a feed-forward network simply computes a function of the input values that depends on the weight settings—it has *no internal states* (short term memory) other than the weights themselves.

Obviously, the brain cannot be a feed-forward network, else we would have no short-term memory. In other words, the brain is a "recurrent network". Because the activation is somehow fed back to the units that caused it, recurrent networks have *internal state stored* in the activation levels of the nodes. It results in a less orderly computation compared with that of feed-forward network. Unfortunately, recurrent network can become *unstable*, or *oscillate*; given some input value, it can take a long time to compute a stable output and learning is made more difficult.

Layered feed-forward networks were first studied in the late 1950s under the name ***perceptrons***. Perceptron is a very simple neural network with one input layer, one output layer and *no* hidden layers. This makes the learning problem much simpler, but it also limit what kind of function it can represent. The publication of ***Perceptrons*** [8] marked the end of the era of early neural network research, as it proved that a perceptron algorithm can learn any *linearly separable function*, given enough training examples. Unfortunately, the book also proved that a perceptron *cannot* learn non-linearly separable function.

Neural networks with one or more layers of hidden units are called *multilayer networks.* A multilayer network, which has one (sufficiently large) layer of hidden units, is able to represent any continuous function (linear or non-linear) of inputs; it is even able to represent discontinuous functions if it has two sufficiently large hidden layers.

By using the *perceptron learning rule*, the weights in a perceptron can be modified to fit any linearly separable data. While the multilayer feed-forward network uses the ***back-propagation*** learning algorithm to fit the data. The back-propagation learning algorithm was first invented in 1969, but was not popularized until 1986 [9]. In the terminology of machine learning, both learning algorithm are performing *gradient descent* in weight spaces.

Gradient Descent

Gradient descent and ***hill-climbing*** are in the class of *iterative improvement algorithms.* The general idea is to start the searching with a complete configuration and to make modifications to improve its quality in each iteration. In contrast with the common search algorithms for a chess playing problem, an iterative improvement algorithm does not maintain a search tree. It is particularly suitable for the problem having the property that the state description itself contains all the information needed for a solution; and the path by which the solution is reached is irrelevant, e.g. the 8-QUEENS problem.

Gradient descent (or, alternatively, hill-climbing if the evaluation function is viewed as a quality rather than a cost) is simply a loot that continually moves in the direction of decreasing cost. The algorithm does not maintain a search tree, so the data structure need only record the state and its evaluation. Two main components involved in a gradient descent are:

1. A cost function defined on the space of states.

2. A gradient estimation function.

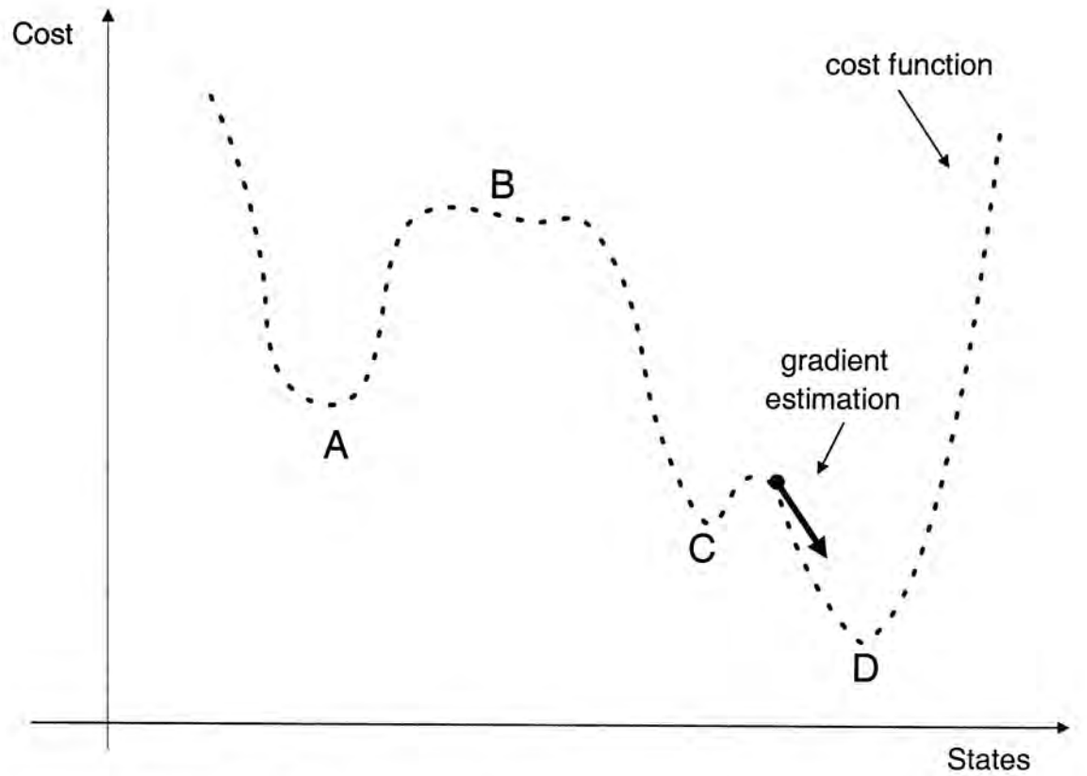


Figure 2.3: The cost function of a gradient descent

Backpropagation is a perfect example of the gradient descent. In the case, the gradient is on the error surface in the weight space: the surface that describes the error on each examples as a function of all the weights in the neural network. The key importance of backpropagation is that: it provides a way of dividing the calculation of the gradient among the nodes, so the change in each weight can be calculated by the corresponding node (by a partial derivative), using only *local* information.

Pros and Cons

To conclude, we first point out the pros of using neural networks:

- Neural networks (multilayer) are capable to represent *any* continuous function and Boolean function, given enough layers and nodes. In particular, to represent all Boolean functions of n inputs, $2^n/n$ hidden units are needed.

- They are an attributed-based representation and are well-suited for continuous inputs and outputs, unlike most decision trees.
- In general, they are effective in learning a "acceptable" solution. If there are m training examples and w weights, each epoch only takes $O(mw)$ time; despite that the worst-case number of epochs can be exponential in n, the number of inputs.
- They provide a good generalization from the training data.
- They are essentially doing nonlinear regression, so they are very tolerant of noise in the input data.

However, neural networks using backpropagation inherit the weakness of gradient descent. As in **Fig. 2.3**, we see the gradient descent policy has three well-known drawbacks [10]:

- ***Local minima***: it is a valley, e.g. A or C, that is higher than the ***global minima***, i.e. the lowest valley D, in the whole state space. Once the algorithm reach the local minima, it is trapped even though the solution may be far away from satisfactory.
- ***Plateau***: it is an area of the state space where the cost function is basically flat, e.g. B. The search may conduct a random walk here.
- ***Valley***: it may have steeply sloping sides, so that the search reaches the bottom easily, but the bottom may slope only very gently toward the foot. The search may result in oscillations.

Moreover, all neural networks suffer the common problems:

- They do not have the *expressive power* of general logical representations, e.g. rule-based systems and decision trees. With decision trees and other logical representations, the result can be interpreted by human beings; while it is possible with neural networks.

- Neural networks are essentially black boxes and lack *transparency.* Even if the network gives good experiment results and predicts new cases well, many users will still be dissatisfied because they will have no idea *why* a given output value is reasonable.
- Designing a "good" or "perfect" topology for a neural network is still under research.
- Neural networks have no means to provide the *degree of certainty* in their output values.
- Because of the lack of transparence, it is hard to improve the performance of neural networks by *prior knowledge.*

2.3 Related Learning Systems

This section reviews two kinds of learning systems which are related in this research, namely relational concept learning and learning of fuzzy concepts. This is no existing method available in literature to combine the learning of relational and fuzzy concepts, the possible difficulties will be discussed in **Section 5.2.**

2.3.1 Relational Concept Learning

Since the late 1980s, there had been increasing interest in systems which induce first order logic programs from examples. However, many difficulties need to be overcome. By that time, many well-known algorithm failed to discover correct logical descriptions for large classes of interesting predicates, due either to the intractability of search or overly strong limitations applied to the hypothesis space. Quinlan's FOIL [11] and Muggleton's GOLEM [12] were the first two successful algorithms to get around the problem.

The main difference between those systems is that FOIL employs inductive inference while GOLEM makes use of deductive inference (please read **Section 2.2.1**). Particularly, GOLEM avoids searching by using the relative least general generalization; the search of logic program is replaced by the process of constructing a unique clause which covers a set of examples relative to given background knowledge.

However, this setting can be hardly modified to learn fuzzy logic program. On the other hand, FOIL guides the hill-climbing search by using an information-based heuristic, which will be replaced by other heuristics for inducing fuzzy logic programs. So that, here we give an overview of FOIL and its development.

Overview of the original FOIL

In a nutshell, the original FOIL [11] is a system for learning function-free Horn clause definitions of a relation in terms of itself and other relations.

FOIL's input consists of information about the relations, one of which (the ***target relation***) is to be defined by a Horn clause program. For each relation it is given a set of tuples of constants that belongs to the relation. For the target relation it might also be given tuples that are known not to belong to the relation; alternatively, the *closed world assumption* may be invoked to state that no tuples, other that those specified, belong to the target relation. Tuples known to be in the target relation will be referred to as $\oplus$ tuples and those not in the relation as $\ominus$ tuples. The learning task is then to find a set of clauses for the target relation that accounts for all the $\oplus$ tuples while not covering any of the $\ominus$ tuples.

The basic approach used by FOIL is an AQ-like covering algorithm [13]. It starts with a *training set* containing all $\oplus$ and $\ominus$ tuples, constructs a function-free Horn clause to 'explain' some of the $\oplus$ tuples, removes the covered $\oplus$ tuples from the training set, and continues with the search for the next clause.

When clauses covering all the $\oplus$ tuples have been found, they are reviewed to eliminate any redundant clauses and reordered so that any recursive clauses come after the non-recursive base cases.

Perfect definitions that exactly match the data are not always possible, particularly in real-world situations where incorrect values and missing tuples are to be expected. To get around this problem, FOIL uses encoding-length heuristics to limit the complexity of clauses and programs. The final clauses may cover most (rather than all) of the $\oplus$ tuples while covering few (rather than none) of the $\ominus$ tuples.

FOCL and the Development of FOIL

Pazzani and Kibler developed FOCL [14] to extend the original FOIL in a variety of ways. Each of these extensions affects only how FOCL selects literals to test while extending a (possibly empty) clause under construction. The following extensions allow FOCL to use domain knowledge to guide the learning process:

1. Using *theoretical constants* to limit the search space.

2. Using intensionally defined predicates (i.e., predicated defined by a rule instead of a collection of examples) in a manner similar to the extensionally defined predicates in FOIL. A collection of intensionally defined predicates is identical to the domain theory of EBL.

3. Accepting input as a partial, possibly incorrect rule that is an initial approximation of the predicate to be learned.

The report in [15] summarized the development of FOIL from 1989 up to 1993. FOIL borrowed the idea of ***determinate terms*** and ***variable depth*** from GOLEM to enhance pruning. The improved FOIL also accepted further literal forms, in such a way that a ***theory constant*** can appear explicitly in

a definition (as in FOCL) and a ***numeric variable*** is allowed. The augmentation of FOIL to handle data with continuous variables and data with missing values enables it to be applied to a wider range of problems, including conventional *attribute-value classification* problem, which is described in [16].

The latest implementation of FOIL (version 6.4), which was published in 1996, basically includes all the features proposed by the several successor of the original FOIL. This version is always available at the URL: *http://www.cse.unsw.edu.au/ quinlan/foil6.sh* .

2.3.2 Learning of Fuzzy Concepts

The induction of fuzzy modules from a set of input-output examples (a zeroth order dataset) has been widely discussed. The methods differentiate in the way they represent the knowledge and most importantly, the way they learn. There is no authoritative paper in this field, probably because the fuzzy set theory is not defined that clearly compared with the classical set theory; there are always alternatives in the fuzzification process, in the composition of degree of memberships, in the interpretation of fuzzy inference, etc. And the systems are normally problem-specific, yet are not general learning algorithms. Here I highlight some of the representative fuzzy learning methods.

Use of Linguistic Quantifiers

An approach to inductive learning under imprecision and errors is proposed in [17]. It is the first inductive learning system to apply the concepts of "linguistic quantifiers" (please refer to **Section 2.4.5**) and "fuzzy covering". For the latter concept, the system assume that the classification into the positive and negative examples is to a degree (of positiveness and negativeness) between 0 and 1. The algorithm use a linear procedure to generate a "typoid", which represents the conjunction of selectors that would produce the "most positive"

output. It iteratively pick up the selectors in the typoid to generate a most appropriate clause. The paper claims that the algorithm is in general very effective and efficient in practice.

However, the grade of positiveness and negativeness of each example should be given explicitly beforehand. This shortcoming restricts the system to use practical dataset, where no degree of positiveness or negativeness will be given. Anyway, the generalization of the covering concept is highly related to our research and we will discuss it in deep in **Chapter 5**.

Modification of AQ

The FAQR algorithm [18] induce fuzzy linguistic rules to solve the parallel loop scheduling problem. It generalizes the AQ algorithm [13], which performs a heuristic search through the hypothesis space to determine the descriptions that account for all positive instances and no negative instances (almostly). AQ processes the training instances in stages; each stage generates a single rule, and then removes the instances it covers from the training set. The step is repeated until enough rules have been found to cover (almost) all the instance of the chosen class. Note that AQ is also the underlying concept for the FOIL algorithm.

FILSMR [19] use a similar algorithm with FAQR. But it uses the "α-covering" concept, i.e. a predefined significant level α (say, 0.5), is used to determine an example is α-covered or not. This approach is simple yet in some sense it is a still a crisp classification system.

FAQR generalizes the "covering" concept in AQ to the "fuzzy covering" concept as in [17]. FAQR claims that it performs more accurately than other crisp learning algorithms through the iris experiment.

Information-theoretic Approach

[20] generates a very specific rule for each example. Then it generates the rule set by produce fuzzy sets to conclude the variable content across the rules. It is in fact an adaptive fuzzification process. In the pruning stage, the system evaluates each rule by the J-measure [21]. The method is successfully applied to a vehicle control system experiment.

Other Approaches

Some of the other approaches include induction of fuzzy decision-trees instead of fuzzy ruleset. And some other non-inductive learning methods are also proposed, e.g. the use of belief-network, neural network or genetic algorithm, which are out of our scope of interest.

2.4 Fuzzy Logic

Fuzzy logic, or interchangeably referred to as fuzzy set theory, had it's formal start in 1965 when Lotfi Zadeh published his innovating paper "Fuzzy Sets" [4]. Since then it has been infiltrating in almost all branches of pure and applied mathematics that are set theory-based, and resulted in a vast number of real applications crossing over a broad range of domains and disciplines [22]. An extensive survey in fuzzy systems is given in [23].

In this section, we will introduce the basic concepts of fuzzy logic that are of interest or relevance to the discussions of successive sections. Specificially, **Section 2.4.1** describes the definition of fuzzy sets. **Section 2.4.2** and **Section 2.4.3** discuss some of the related notations and operations. While in **Section 2.4.4**, fuzzy relations are introduced. Finally, **Section 2.4.5** describes the fuzzy first-order logic and the development of fuzzy Prolog.

2.4.1 Fuzzy Set

A ***fuzzy set*** is a generalization of an ordinary (non-fuzzy) set. Formally, let U be the ***universe of discourse***, a fuzzy set F on U is characterized by a ***membership function*** $\mu_F : U \rightarrow [0, 1]$, which associates each element $u \in U$ with a number $\mu_F(u)$ representing the grade of membership of u in F. Symbolically,

$$F \triangleq \int_U \mu_F(u)/u \tag{2.8}$$

or

$$F \triangleq \mu_F(u)/u | u \in U \tag{2.9}$$

As in (2.8), $\mu_F(u) = 0$ means non-membership, $\mu_F(u) = 1$ means full membership, while $0 < \mu_F(u) < 1$ means *partial* membership. In general, a fuzzy set is assumed to be imbedded in a nonfuzzy universe of discourse, which may be any collection of objects, concepts (discrete), or mathematical constructs (discrete or continuous). One of the main contributions of fuzzy logic is a methodology for computing with words and no other methodology serves this purpose [?], i.e. fuzzy sets can represent the semantics of linguistic terms. In such cases, the fuzzy sets are referred as *linguistic variables*, which are applied in the *modeling of concepts.*

In the example shown in **Fig. 2.4**, `Age` (0 to 90) is the universe of discourse of the three fuzzy sets (or the three linguistic variables): `Young`, `Middle_aged` and `Old`. For instance, at the age 25, $\mu_{Young}(25) \approx 0.84 > \mu_{Middle\_aged}(25) \approx 0.17$, which is interpreted as "People of age 25 are *more likely* considered to be young rather than to be middle-aged".

To simply the representation of *finite* fuzzy subsets, the notation "+" can be extended to play the role of union rather than the arithmetic sum. That is, a *finite* fuzzy set F of $U = \{u_1, \ldots, u_n\}$ is expressed as the linear form

$$F = \mu_1/u_1 + \cdots + \mu_n/u_n \tag{2.10}$$

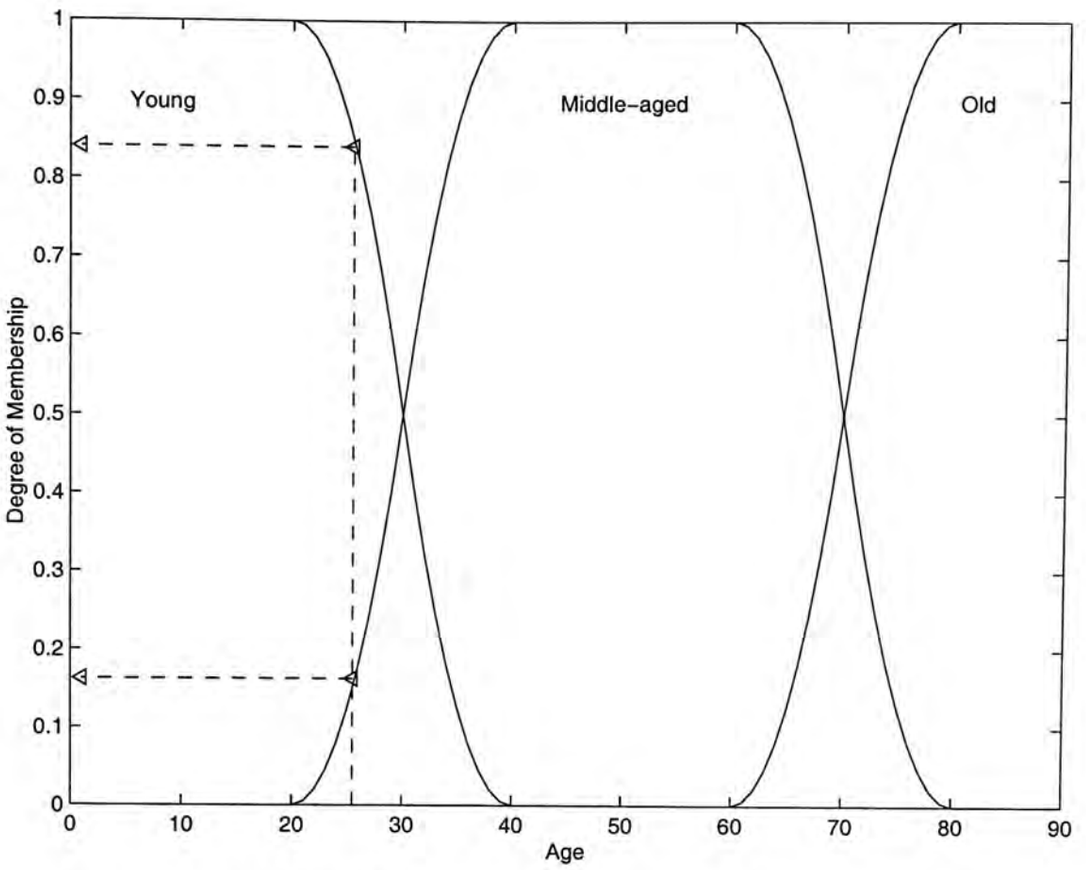


Figure 2.4: Universe of discourse and fuzzy sets

or

$$F = \sum_{i=1}^{n} \mu_i / u_i \tag{2.11}$$

where μ_i, $i = 1, \ldots n$, is the grade of membership of u_i in F. The notation "/" is the separator symbol for disambiguation in case that the u_i are numbers.

Note that any ordinary set, which does not support the concept of partial membership, is just a special case of fuzzy sets. In such a way, fuzzy set theory extends the ordinary set theory, and the operations on fuzzy sets should be consistent with the corresponding operations on ordinary sets. The following sub-sections list the notations and operations of fuzzy sets which are related to FF99.

2.4.2 Basic Notations in Fuzzy Logic

Let U be an ordinary set and A be a fuzzy set on U with the membership function $\mu_A : U \rightarrow [0, 1]$, then:

- The set of the elements that have non-zero degrees of membership in A is called the ***support*** of A, denoted by $supp(A)$:

$$supp(A) = \{u \mid u \in U \text{ and } \mu_A(u) = 1\} \tag{2.12}$$

- The set of the elements whose degrees of membership in A are greater (at least equal to) α, where $0 \leq \alpha < 1$ $(0 < \alpha \leq 1)$, is called the ***strong (weak)*** α***-level-set*** of A, denoted by $A_{\alpha+}(A_\alpha)$:

$$\begin{aligned} A_{\alpha+} &= \{u \mid u \in U \text{ and } \mu_A(u) > \alpha\}, \\ A_\alpha &= \{u \mid u \in U \text{ and } \mu_A(u) \leq \alpha\} \end{aligned} \tag{2.13}$$

- The ***height*** of a fuzzy set A on U is defined as:

$$height(A) = \sup_{u \in U} \mu_A(u) \tag{2.14}$$

- The ***plinth*** of a fuzzy set A on U is defined as:

$$plinth(A) = \inf_{u \in U} \mu_A(u) \tag{2.15}$$

2.4.3 Basic Operations on Fuzzy Sets

Zadeh's original extension of the classical set union, intersection and complementation are based Morgon on soft algebra. Let A and B be fuzzy sets on the same universe U with the membership functions μ_a and μ_B respectively, then

Intersection: $A \cup B$ is the fuzzy set respresenting the intersection of A and B with the membership function:

$$\forall u \in U \ , \ \mu_{A \cup B}(u) = \mu_A(u) \ \mathrm{t} \ \mu_B(u) \tag{2.16}$$

where $\mathrm{t} : [0,1] \times [0,1] \rightarrow [0,1]$ is the so-called ***t-norm*** defined as, for each $a, b, c, d \in [0,1]$:

1. $a \ \mathrm{t} \ 1 = a$
2. $a \ \mathrm{t} \ b = b \ \mathrm{t} \ a$
3. $a \ \mathrm{t} \ b \geq c \ \mathrm{t} \ d$ if $a \geq c, b \geq d$
4. $a \ \mathrm{t} \ b \ \mathrm{t} \ c = a \ \mathrm{t} \ (b \ \mathrm{t} \ c) = (a \ \mathrm{t} \ b) \ \mathrm{t} \ c$

Some examples of t-norms are: $a \wedge b = min(a, b)$, which is the most commonly used, ab, and $1 - (1 \wedge ((1-a)^p + (1-b)^p)^{1/p}, p \geq 1$.

The intersection operation represents the logical `AND` of the concepts modeled by the corresponding fuzzy sets.

Union: $A \cup B$ is the fuzzy set representing the union of A and B with the membership function:

$$\forall u \in U \ , \ \mu_{A \cup B}(u) = \mu_A(u) \ \mathrm{s} \ \mu_B(u) \tag{2.17}$$

where $\mathrm{s} : [0,1] \times [0,1] \rightarrow [0,1]$ is the so-called ***s-norm*** (***t-conorm***) defined as, for each $a, b, c, d \in [0,1]$:

1. $a \ \mathrm{s} \ 0 = a$
2. $a \ \mathrm{s} \ b = b \ \mathrm{s} \ a$
3. $a \ \mathrm{s} \ b \geq c \ \mathrm{s} \ d$ if $a \geq c, b \geq d$
4. $a \ \mathrm{s} \ b \ \mathrm{s} \ c = a \ \mathrm{s} \ (b \ \mathrm{s} \ c) = (a \ \mathrm{s} \ b) \ \mathrm{s} \ c$

Some examples of s-norms are: $a \vee b = max(a, b)$, which the the most commonly used, $a + b - ab$, and $1 \wedge (a^p + b^p)^{1/p}, p \geq 1$.

The union operation represents the logical `OR` of the concepts modeled by the corresponding fuzzy sets.

Complementation: $\overline{A}$ is the fuzzy set respresenting the complementation of A with the membership function:

$$\forall u \in U \ , \ \mu_{\overline{A}}(u) = 1 - \mu_A(u) \tag{2.18}$$

The complementation operation represents the logical `NOT` of the concept modeled by the fuzzy set.

2.4.4 Fuzzy Relations, Projection and Cylindrical Extension

Fuzzy Relations

According to the original definition in [4]: If U is the Cartesian product of n universes of discourse $U_1, \ldots, U_n$, then an ***n-ary fuzzy relation***, R, in U is a fuzzy subset of U. As in (2.8), R may be expressed as the union of its constituent singletons $\mu_R(u_1, \ldots, u_n)/(u_1, \ldots, u_n)$, i.e.,

$$R \triangleq \int_{U_1 \times \cdots \times U_n} \mu_R(u_1, \ldots, u_n)/(u_1, \ldots, u_n) \tag{2.19}$$

where μ_R is the membership function of R.

A ***binary fuzzy relation*** is the most frequently used fuzzy relation. Common examples of binary fuzzy relation includes: `much_greater_than`, and `close_to`. For example, if $U_1 = U_2 = (-\infty, \infty)$, the relation `close_to` may be defined by

$$R_{\texttt{close\_to}} = \int_{U_1 \times \cdots \times U_2} e^{-|u_1 - u_2|}/(u_1, u_2)$$

We can visualize `close_to` by plotting the two-dimensional membership function in **Fig. 2.5**.

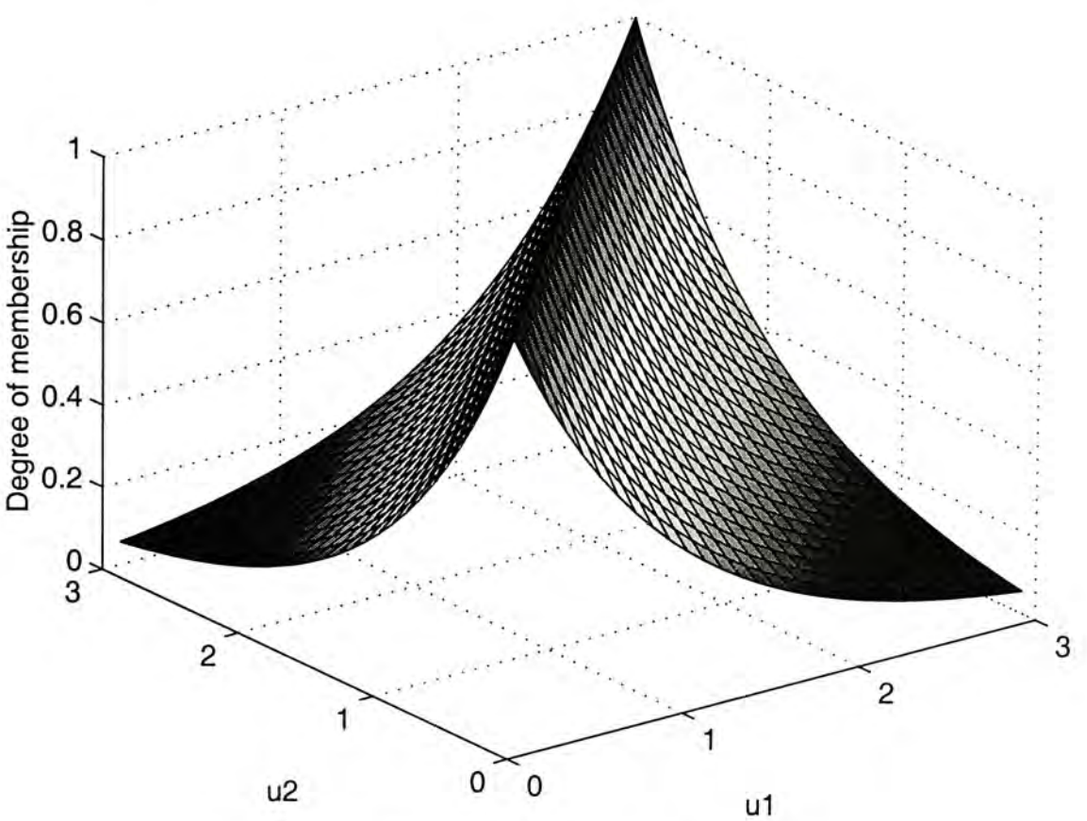


Figure 2.5: 3D Visualization of $\mu_{\text{close to}}(u_1, u_2)$

Some operations on fuzzy relations are just the trivial extensions of the corresponding operations on fuzzy sets. Such operations include union, intersection and complementation as mentioned in **Section 2.4.3**. However, there are some operations, e.g. projection and cylindrical extension, are defined solely on fuzzy relations but not on fuzzy sets. They will be discussed as they are closely related to the research of this thesis.

Projection

If R is an n-ary fuzzy relation in $U_1 \times \cdots \times U_n$, then its ***projection*** (also called ***shadow*** or ***marginal fuzzy restriction***) on $U_{i_1} \times \cdots \times U_{i_k}$ is a k-ary fuzzy

$\mu_R(u_1, u_2)$		u_1			
		1	2	3	4
u_2	1	0	0.3	0.8	1.0
	2	0	0.9	0	0.8
	3	0	0	0	0.3
	4	0	0	0	0

Table 2.1: Binary fuzzy relation R

relation R_q in U which is defined by

$$R_q \triangleq \int_{U_{i_1} \times \cdots \times U_{i_k}} (\bigvee_{u_{(q')}} \mu_R(u_1, \ldots, u_n))/(u_{i_1}, \ldots, u_{i_k}) \tag{2.20}$$

where q is the index sequence $(i_1, \ldots, i_k)$; $u_{(q)} \triangleq (u_{i_1}, \ldots, u_{i_k})$; q' is the complement of q; and $\bigvee_{u_{(q')}}$ is the supremum of $\mu_R(u_1, \ldots, u_n)$ over the u's which are in $u_{(q)}$, i.e.

$$\mu_{R_q}(u_{i_1}, \ldots, u_{i_k}) = \sup_{\substack{u_x = u_{i_j} \\ j=1,\ldots,k}} \mu_R(u_1, \ldots, u_n) \tag{2.21}$$

To illustrate the situation, if $U_1 = U_2 = 1 + 2 + 3 + 4$ and the binary fuzzy relation R is defined by the **Table 2.1**. Then the corresponding projection of R onto U_1 and U_2 are namely R_1 and R_2:

$$R_1 = 0.0/1 + 0.9/2 + 0.8/3 + 1.0/4$$

and

$$R_2 = 1.0/1 + 0.9/2 + 0.3/3 + 0.0/4$$

Cylindrical Extension

It is clear that distinct fuzzy relations in $U_1 \times \cdots \times U_n$ can have identical projections on $U_{i_1} \times \cdots \times U_{i_k}$. However, given a fuzzy relation R_q in $U_{i_1} \times \cdots \times U_{i_k}$, there exists a *unique largest* relation $\overline{R_q}$ in $U_1 \times \cdots \times U_n$ whose projection

$\mu_{\overline{R_1}}(u_1, u_2)$		u_1			
		1	2	3	4
u_2	1	0	0.9	0.8	1.0
	2	0	0.9	0.8	1.0
	3	0	0.9	0.8	1.0
	4	0	0.9	0.8	1.0

Table 2.2: Cylindrical extension of R_1

$\mu_{\overline{R_2}}(u_1, u_2)$		u_1			
		1	2	3	4
u_2	1	1.0	1.0	1.0	1.0
	2	0.9	0.9	0.9	0.9
	3	0.3	0.3	0.3	0.3
	4	0	0	0	0

Table 2.3: Cylindrical extension of R_2

on $U_{i_1} \times \cdots \times U_{i_k}$ is R_q. In consequence of (2.21), the membership function of $\overline{R_q}$ is given by

$$\mu_{\overline{R_q}}(U_1 \times \cdots \times U_n) = \mu_{R_q}(U_{i_1} \times \cdots \times U_{i_k}) \tag{2.22}$$

In (2.22), $\overline{R_q}$ is referred as the ***cylindrical extension*** of R_q, with R_q constituting the ***base*** of $\overline{R_q}$.

Following the previous example of R as described in **Table 2.1**, the cylindrical extensions of the R_1 and R_2 are as **Table 2.2** and **Table 2.3** respectively.

2.4.5 Fuzzy First Order Logic and Fuzzy Prolog

In this section, we examine fuzzy first-order logic and fuzzy Prolog. Respectively, we first discuss the properties of the classical first-order logic and Prolog, then we step forward to combine fuzzy logic with them.

Fuzzy First-Order Logic

In ***propositional logic***, ***symbols*** represent the whole propositions (***facts***); for example, A might have the interpretation "the wumpus is dead." which may or may not be a true proposition. Proposition symbols can be combined be combined using ***Boolean connectives*** to generate ***sentences*** with more complex meanings. Such a logic makes very little commitment to how things are represented,so it is not surprising that it does not give us much mileage as a representation language.

First-order logic commits to the representation of worlds in terms of ***objects*** and ***predicates*** on objects, which are the ***properties*** of objects or ***relations*** between objects. It uses ***connectvies*** and ***quantifiers***, which allow sentences to be written about everything in the universe at once. First-order logic is able to capture a good deal of what we know about the world, and has studied for about a hundred years.

To make the difference less abstract, we illustrate it the following sentences:

1. Mary is the mother of Peter.
2. John is the father of Peter.
3. `IF` Mary is the mother of Peter `AND` John is the father of Peter
   `THEN` John is the husband of Mary.

In propositional logic, sentence 1 and 2 are called the ***atomic sentence***, which could be evaluated to be **True** or **False**. While sentence 3 is called the ***rule***, which is one type of ***complex sentence***. And that's all: what we could only do is to evaluate the truth value of the atomic sentences and use propositional inference (in specific, by Modus Ponens) to get the truth value of the rule.

On the other hand, in first-order logic, we identify "mother", "father" and "husband" as ***relations***, while "Mary", "John" and "Peter" are called the ***constants***. Sentences 1 and 2 form two ***predicates***, while sentence 3 uses

Sentence	→	*AtomicSentence* | *ComplexSentence*
AtomicSentence	→	***True*** | ***False***
		| *Mary is the mother of Peter*
		| *John is the father of Peter*
		| *John is the husband of Mary* | ...
ComplexSentence	→	*(Sentence)*
		| *Sentence Connective Sentence*
		| ¬ *Sentence* ⟨*NOT*⟩
Connective	→	∧ ⟨*AND*⟩ | ∨ ⟨*OR*⟩ | ⇔ ⟨*EQUIVALENT*⟩ | ⇒ ⟨*IMPLY*⟩

Table 2.4: A BNF grammer of sentences in propositional logic

connectives to link up the predicates. They are re-written in first-order logic language as :

1. `mother(Mary,Peter)`

2. `father(John,Peter)`

3. `mother(Mary,Peter)` ∧ `father(John,Peter)` ⇒ `husband(John,Mary)`

As you can see, first-order logic interprets a proposition in the depth of relations and the connection between relations. More importantly, for what first-order logic superior to propositional logic, is the use of ***variables*** and ***quantifiers*** enabling us to *generalize* the concept as:

$$\forall \text{ A,B,C } \text{mother(A,B)} \wedge \text{father(C,B)} \Rightarrow \text{husband(C,A)}$$

To summerize, **Table 2.4** and **Table 2.5** gives the syntax of the two languages in BNF (Backus-Naur Form).

Fuzzy first-order logic enriches first-order logic based on fuzzy logic, which can have degrees of truth in a sentence, so that a fact need not to be absolutely true or false. However, there is no formal syntax of fuzzy first-order logic as it is highly application oriented. For example, different implementation

Sentence	$\rightarrow$	*AtomicSentence* | *Sentence Connective Sentence* | *Quantifier Variable, ... Sentence* | $\neg$ *Sentence* $\langle$*NOT*$\rangle$ |*(Sentence)*
AtomicSentence	$\rightarrow$	*Predicate(Term, ...)* | *Term=Term*
Term	$\rightarrow$	*Constant* | *Variable*
Connective	$\rightarrow$	$\wedge$ $\langle$*AND*$\rangle$ | $\vee$ $\langle$*OR*$\rangle$ | $\Leftrightarrow$ $\langle$*EQUIVALENT*$\rangle$ | $\Rightarrow$ $\langle$*IMPLY*$\rangle$
Quantifier	$\rightarrow$	$\forall$ $\langle$*UNIVERSAL*$\rangle$ | $\exists$ $\langle$*EXISTENTIAL*$\rangle$
Constant	$\rightarrow$	*John* | *Mary* | *Peter* | ...
Variable	$\rightarrow$	*A* | *B* | *C* | ...
Predicate	$\rightarrow$	*mother* | *father* | *husband* | ...

Table 2.5: A BNF grammer of sentences in first-order logic

of fuzzy Prolog (will be discussed in **Section 2.4.5**) may accept sentences in different syntax of fuzzy first-order logic. As well in inductive learning systems, which usually define their own acceptable syntax of fuzzy first-order logic. In this thesis, we would define our syntax in **Section 3.1**. The following enhancements of conventional first-order logic are available in the literature of fuzzy first-order logic:

Predicate → Fuzzy Predicate : In first-order logic, a predicate symbol refers to a particular relation in the model. For example, the `mother` symbol might refer to the relation of motherhood. `mother` is a binary predicate symbol, and accordingly motherhood is a relation that holds (or fails to hold) between pairs of objects. In any given model, the relation is defined by the set of ***tuples*** of objects that satisfy it. A tuple is a collection of objects arranged in a fixed order, e.g.

$$\texttt{mother} = \{\langle Mary, Peter\rangle,\ \langle Susan, Sam\rangle\}$$

Thus, formally speaking, `mother` refers to this set of tuples under the interpretation we have chosen. On the other way, in fuzzy first-order logic, the degree of membership of the particular collection of objects is attached in the tuple. One might define the underlying fuzzy relation as the form in (2.19), or in form of a set of extended tuples. For example, the binary fuzzy predicate `friend` might be defined as:

$$\texttt{friend} = 0.9/(Thomas, Bobo) + 0.2/(Mary, Susan)$$

or

$$\texttt{friend} = \{\langle Thomas, Bobo, 0.9\rangle,\ \langle Mary, Susan, 0.2\rangle\}$$

In this example, we would get $\mu_{\texttt{friend}}(Thomas, Bobo) = 0.9$ and $\mu_{\texttt{friend}}(Mary, Susan) = 0.2$. For the tuples that are not specified in

the fuzzy relation set, they might be treated as unknown or with a zero degree of truth, depends if the close world assumption is applied or not. That is, if the close world assumption is not used, the tuples with zero degree of truth should be specified explicitly in the fuzzy relation set, or else they will be treated as unknown.

Connective → Fuzzy Connective : The Boolean connectives $\wedge$ and $\vee$ are no longer valid in linking fuzzy predicates. Instead, the t-norm and s-norm operators, as described in **Section 2.4.3**, are used respectively. Most of fuzzy logic are such that implication is Kleen-Diene's, i.e. if the antecedent is true to some degree of membership, the n the consequent is also true to that same degree. The consequent specifies a fuzzy set be assigned to the output. The implication function then modifies that fuzzy set to the degree specified by the antecedent. The most common ways to modify the output fuzzy set are truncation using the *min* function or scaling using the $\times$ function. In some fuzzy Prolog system, such as [24], use the Lukasiewicz implication operator defined as :

$$a \Rightarrow b \;=\; min(1 - a + b, 1) \tag{2.23}$$

The Boolean negation operator $\neg$ is universally changed to be the fuzzy complementation as in **Section 2.4.3**.

Connecting Fuzzy Predicates in Different Arity : As described before, fuzzy connectives are used in fuzzy first-order logic sentences. For some cases, that the arity of predicates to be connected are not the same, ***projection*** or ***extension*** have to be processed before the connectives are applied. Description of of the projection and the extension processes is in **Section 2.4.4**. For instance, the following sentence which contain three fuzzy predicates, all in different arity :

$$\mathtt{p1}(A, B) \wedge \mathtt{p2}(B, C) \Rightarrow \mathtt{p3}(A, C)$$

Assume that the problem is to find the membership function of the fuzzy relation p3, $\mu_{\texttt{p3}}(A, C)$, given those from p1 and p2, i.e. $\mu_{\texttt{p2}}(A, B)$ and $\mu_{\texttt{p2}}(B, C)$. First, we have to extend all fuzzy relation to the space $(A \times B) \cup (B \times C) \cup (A \times C) = A \times B \times C$. In such a way, we produce $\mu_{\overline{\texttt{p1}}}(A, B, C)$ and $\mu_{\overline{\texttt{p2}}}(A, B, C)$. Then, use the *min* union connective to get the extension of $\mu_{\texttt{p3}}(A, C)$:

$$\mu_{\overline{\texttt{p3}}}(A, B, C) = min(\mu_{\overline{\texttt{p1}}}(A, B, C), \mu_{\overline{\texttt{p2}}}(A, B, C))$$

Finally, the projection is processed to get :

$$\mu_{\texttt{p3}}(A, C) = \sup_{A,C} \mu_{\overline{\texttt{p3}}}(A, B, C)$$

Quantifier → Linguistic Quantifier : Zadeh produced the calculus of ***linguistically quantified propositions*** in [25]. A linguistically quantified proposition can be generally written as

$$Q\ y's \text{ are } F \tag{2.24}$$

where Q is a linguistic quantifier (e.g., "most"), $Y = \{y\}$ is a set of objects (e.g., "experts"), and F is a property (e.g., "convinced"). Then the proposition represents "most experts are convinced". In Zadeh's approach the fuzzy linguistic quantifier Q is assumed to be fuzzy set in $[0, 1]$. For instance, $Q =$ "most" may be given as

$$\mu_{\text{most}}(u) = \begin{cases} 1 & : \ u \geq 0.8 \\ 2u - 0.6 & : \ 0.3 < u < 0.8 \\ 0 & : \ u \leq 0.3 \end{cases} \tag{2.25}$$

In some fuzzy first-order logic systems, the universal quantifier $\forall$ is extended to the linguistic quantifier $\tilde{\forall}$ to represent "most" or "almost all". For instance, the following sentence

$$\tilde{\forall} X \ \texttt{rich}(X) \Rightarrow \texttt{business}(X)$$

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes)
Proposition logic	facts	true/false/unknown
First-order logic	facts,objects,relations	true/false/unknown
Fuzzy logic	facts	degree of belief 0 ... 1 / unknown
Fuzzy first-order logic	facts,objects,relations	degree of belief 0 ... 1 / unknown

Table 2.6: Formal logics and their ontological and epsitemological commitments

describes "most rich people are doing business".

This linguistic quantifier is used in several fuzzy inductive learning systems in [17], [19] and [18].

Other : Some other fuzzy data mining systems may attached a certainty factor or a degree of truth in each of the rule to allow fuzziness. However, they may be not classified into formal fuzzy first-order logic systems, and we just mention them for completeness.

Table 2.6 summarizes the comparison of different logic languages discussed.

Fuzzy Prolog

Prolog, a programming language based on the first order predicate calculus, has been widely used in artificial intelligence research. One of its shortcomings is the lack of a natural mechanism to deal with uncertainty. A possible solution to this problem is to base Prolog on fuzzy logic rather than on conventional two-valued logic. This leads to to a more general system, of which standard Prolog is a special case. And the development of fuzzy Prolog systems makes fuzzy first-order logic language be a practical language rather than a purely research topic.

Hinde [26] was the first author to incorporate fuzzy expressions into the

Prolog language. The translation of English into horn clause format is described and is used to illustrate the simplicity of representation using "variable functors". Apart from enabling imprecise facts and rules to be expressed, it proposes a natural method of controlling the search which makes the search tree admissible.

The FPROLOG system [27] is a fuzzy Prolog interpreter written in Lisp. It is one of the early well-known fuzzy Prolog system as it could be linked with the famous FRIL system [28] system developed at Bristol. The main novel feature of FPROLOG is its ability to handle *fuzzy facts* and process the truth values (the degrees of membership) to produce the overall truth value for each solution. When a *query* is evaluated, FPROLOG sets an internal truth value to 'true'. Each time a fuzzy fact is used to solve a *sub-goal*, the associated truth value is compared to the current internal truth value, which is reset when appropriate. By default, the intersection operator is taken as the minimum of the two truth values, but the programmer is free to specify any combination operator. On backtracking, of course, the internal truth value must be restored to its previous value.

Since that, various approaches of fuzzy Prolog were proposed, though there was no common interpretation for fuzzifying Prolog. In [29], from the user's side it considered what fuzzy Prolog have to be able to do and it proposed some approaches for fuzzifying Prolog which satisfy user's postulates. LBFP [24] is a fuzzy Prolog based on these proposals. However, a common interpretation of fuzzifying Prolog is still more or less a open research topic.

Chapter 3

Knowledge Representation and Learning Algorithm

3.1 Knowledge Representation

This section explains why we choose fuzzy first-order logic as our representation language. It also discusses numerous literal forms supported by our system in order to represent a large variety of concepts effectively.

3.1.1 Fuzzy First-order Logic — A Powerful Language

One major dimension along which to differentiate concept learning systems is the complexity of the input and output languages that they employ. At one extreme are learning systems that use a propositional attribute-value language for describing entities and classification rules. The simplicity of this formalism allows such systems to deal with large volumn of data and thus to exploit statistical properties of collections of examples and counter examples of a concept. However, this formalism is simple but limited. As discussed in **Section 2.4.5**, propositional logic gives no commitment in describing concepts in terms of objects and relations. An object must be specified by its values for a *fixed set of attributes*, and rules must be expressed as functions of these same attributes.

To see why this presents a problem, consider a domain involving directional networks as the one in **Fig. 3.1**. A similar domain was demonstrated in Quinlan's FOIL paper [11] also.

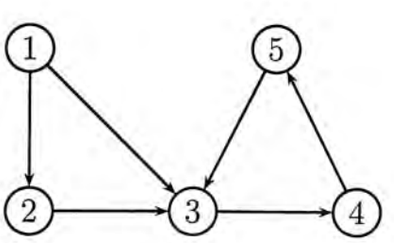


Figure 3.1: A small network illustrating the limitations of propositional descriptions

Suppose now we attempt to set down a collection of attributes which is *sufficient* to describe *any* network. Immediately, we would find that the task

is infeasible: as the number of nodes is unlimited and each node could be linked to an unlimited number of other nodes, thus the number of attributes should be unlimited and *cannot be fixed.* Even though the description task is simplified by restricting networks to a maximum of five nodes, with each node connected to at most two others, say. Any such network could then be represented in terms of the ten attributes

attributes A_1, B_1 : the nodes to which node 1 is linked

attributes A_2, B_2 : the nodes to which node 2 is linked

$\vdots$

attributes A_5, B_5 : the nodes to which node 5 is linked

and perhaps using a zero to denote "not linked". Now, **Fig. 3.1** is described by the possible attribute-value set in **Table 3.1**

	Attribute									
	A_1	B_1	A_2	B_2	A_3	B_3	A_4	B_4	A_5	B_5
Value	2	3	3	0	4	0	5	0	3	0
	3	2	3	0	4	0	5	0	3	0
	$\vdots$									
	3	2	0	3	0	4	0	5	0	3

Table 3.1: The possible attribute-values to describe the network in **Fig. 3.1**

The expression of this concept in the above representation is truly horrific. It is clear that such a propositional description language flounders when faced with complex objects and objects. On the other extreme, structural information such as the network can be represented naturally as a collection of *relations.* A relation is associated with a k-ary predicate and consists of set of k-tuples of constants that satisfy that predicate. For instance, in the above network, the predicate `linked_to`(X, Y), denoting that node X is directly linked to node Y, can be represented by the relation

$$\texttt{linked\_to} = \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 2,3\rangle, \langle 3,4\rangle, \langle 4,5\rangle, \langle 5,3\rangle\}$$

This first-order formalism especially benefits in expressing more complex and *general* concept. Consider the predicate `can_reach`(X, Y), which denotes that node X is directly or indirectly linked to node Y, is represented by the relation

$$\begin{aligned}\texttt{can\_reach} = \{\ & \langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 2,5\rangle, \\ & \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle, \langle 4,3\rangle, \langle 5,3\rangle, \langle 5,4\rangle\ \}\end{aligned}$$

This extensional definition of `can_reach` is applicable to only the given network, yet from this relation and the relation `linked_to` given previously, one could write the *general* definition in first-order logic :

$$\begin{aligned}\forall X, Y, Z \quad & \texttt{linked\_to}(X, Y) \vee (\ \texttt{linked\_to}(X, Z) \wedge \texttt{can\_reach}(Z, Y)\ ) \\ \Rightarrow\ & \texttt{can\_reach}(X, Y)\end{aligned}$$

which is valid for *any* network.

We have demonstrated the power of the first-order language over the propositional language. Now, we would go one step further to allow fuzzyness in the relations, thus the fuzzy first-order logic is chosen to be the language to represent concepts in this thesis. Fuzzy logic is a superset of the traditional Boolean logic. It has a clear advantage over the Boolean logic of expressing human-like concepts, which are vague in nature. For instance, in a domain involving the relationship between the age of people, the concepts `much_older_than`, `middle_aged`, etc, are all vague. Obviously, they are best described by fuzzy logic. Using the fuzzy first-order logic language, we could express them by the fuzzy relations (see **Section 2.4.4**) :

$$\begin{aligned}\texttt{much\_older\_than} = \{\ & \langle 1, 10, 0.0\rangle, \langle 1, 30, 0.0\rangle, \langle 1, 50, 0.0\rangle, \langle 1, 80, 0.0\rangle, \\ & \langle 10, 1, 0.5\rangle, \langle 10, 30, 0.0\rangle, \langle 10, 50, 0.0\rangle, \langle 10, 80, 0.0\rangle, \\ & \langle 30, 1, 1.0\rangle, \langle 30, 10, 0.9\rangle, \langle 30, 50, 0.0\rangle, \langle 30, 80, 0.0\rangle, \\ & \langle 50, 1, 1.0\rangle, \langle 50, 10, 1.0\rangle, \langle 50, 30, 0.5\rangle, \langle 50, 80, 0.0\rangle, \\ & \langle 80, 1, 1.0\rangle, \langle 80, 10, 1.0\rangle, \langle 80, 30, 1.0\rangle, \langle 80, 50, 0.9\rangle\ \}\end{aligned}$$

$$\texttt{middle\_aged} = \{\ \langle 1, 0.0\rangle, \langle 10, 0.0\rangle, \langle 30, 1.0\rangle, \langle 50, 0.9\rangle, \langle 80, 0.0\rangle\ \}$$

and the fuzzy concept "a middle-aged person" could be generally defined as

$$\begin{aligned} \forall X\ \exists Y, Z \qquad & \texttt{much\_older\_than}(Y, X) \wedge \texttt{much\_older\_than}(X, Z) \\ \Rightarrow\ & \texttt{middle\_aged}(X) \end{aligned}$$

For more details about the notation of fuzzy relation and the literature review on fuzzy logic, please refer to **Section 2.4**

This thesis is to learn a ***concept description*** C in a special form of the fuzzy first-order logic language to describe a ***target concept*** T. The concept description C is defined as follows:

$$\begin{cases} C\ :-\ L_{11}, L_{12}, \ldots, L_{1n_1} \\ C\ :-\ L_{21}, L_{22}, \ldots, L_{2n_2} \\ \quad \vdots \\ C\ :-\ L_{m1}, L_{m2}, \ldots, L_{mn_m} \end{cases} \tag{3.1}$$

in which, each $C :- L_{i1}, L_{i2}, \ldots, L_{in_i}$ is an ***extended function-free Horn clause***. It is interpreted as "if L_{i1} and L_{i2} and ... and L_{in_i} then C". Formally, a Horn clause has the form in first order logic language (see **Section 2.4.5**) :

$$P_1 \wedge P_2 \wedge \cdots \wedge P_n \Rightarrow Q \tag{3.2}$$

where the P_i and Q are nonnegated atoms. This form of sentence is special because it matches the premises of the Modus Ponens rule, which guarantees a polynomial-time inference procedure. However, not every knowledge base can be written as a collection of Horn clauses.

As the learned concept description C is not expected to be connected to any inference engine, i.e. the fuzzy Prolog systems (see **Section 2.4.5**), we are free to extend the form of Horn clauses, e.g. to allow negated atoms.

The L_{ij} are called the ***literals***, or alternatively the ***selectors*** in some machine learning literatures. They are expressed in form of fuzzy predicates. We will examine all the literal forms in **Section 3.1.2**.

The collection of Horn clauses represents the disjunction of the conjunctions of selectors. The concept description C is defined by the membership function μ_C :

$$\begin{aligned} \mu_C = \quad & \mu_{\hat{L}_{11}} \text{ t } \mu_{\hat{L}_{12}} \text{ t } \ldots \text{ t } \mu_{\hat{L}_{1n_1}} \\ \text{s } & \mu_{\hat{L}_{21}} \text{ t } \mu_{\hat{L}_{22}} \text{ t } \ldots \text{ t } \mu_{\hat{L}_{2n_2}} \\ \text{s } & \ldots \\ \text{s } & \mu_{\hat{L}_{m1}} \text{ t } \mu_{\hat{L}_{m2}} \text{ t } \ldots \text{ t } \mu_{\hat{L}_{mn_m}} \end{aligned} \tag{3.3}$$

where $\mu_{\hat{L}_{ij}}$ is the membership function of the *projection* or the *extension* of the literal L_{ij} in order to match the arity of C. The basic definitions of projection and extension are located in **Section 2.4.4**. The symbol "t" and "s" are the t-norm and s-norm operators respectively, they are defined in **Section 2.4.3**. Throughout this thesis, the t-norm and s-norm operators are chosen to be the Zadeh's *max* and *min* operations respectively. The universal quantifier $\forall$ is taken for granted and is omitted in C. While the projection operation would eliminate the existential quantifier, given that the existential variables do not appear in the left hand side of C (the consequent part). Note that not all sentences can be converted into Horn form. Fortunately, the Horn sentences are good enough to deal with most classification and data mining applications.

To continue the illustration, the concept "middle-aged people" might now be expressed as

$$\texttt{middle\_aged}(X) :- \texttt{much\_older\_than}(Y, X), \texttt{much\_older\_than}(X, Z)$$

3.1.2 Literal Forms

We have developed and defined the following literal forms specially for our learning system. They are also the literal forms allowed in early version of FOIL. We follow the usual Prolog convention of starting variables with a capital letter, all other atoms being constants. Predicates are typeset in `typewriter` style.

- **Fuzzy Predicate**

$$\mathtt{p}(V_1, V_2, \ldots, V_{\mathtt{p}}) \ , \ \neg\mathtt{p}(V_1, V_2, \ldots, V_{\mathtt{p}})$$

where p is a fuzzy relation (where crisp relation is a special case) and the V_i's are variables. $\neg\mathtt{p} = \bar{\mathtt{p}}$ represents the negation of the fuzzy relation p, we have

$$\mu_{\bar{\mathtt{p}}} \triangleq 1 - \mu_{\mathtt{p}}$$

as defined already in (2.18).

At least one of V_i must have occurred already in the clause. For instance, suppose we now have a clause

$$\mathtt{c}(X, Y) : - \ \mathtt{a}(Y, Z)$$

then the predicate $\mathtt{b}(Z, W)$ is eligible to be appended to the right hand side of the clause as variable Z have occurred already, while the predicate $\mathtt{d}(W)$ is ineligible as the variable W have not occurred in the whole clause (including the left hand side and right hand side).

- **Comparison of Existing Variables**

$$V_i = V_j \ , \ V_i \neq V_j$$

where V_i and V_j are ***existing variables*** (as just discussed before). This literature form acts as a constraint for the binding of constants into variables. Without these constraints, all constants are eligible to be bound to any variables in the same domain.

- **Theory Constant**

Certain constants can be identified as ***theory constants*** that can appear explicitly in a definition. Examples might include a constant []

representing the null list in list-processing task, or the integers 0 and 1 in tasks that involve the natural numbers. For such a theory constant c, the system will also consider literals of the forms

$$V_i = c \ , \ V_i \neq c$$

where V_i is an existing variable of the appropriate type (domain).

As an illustration, consider the task of learning a program for the relations of natural numbers. Consider the semantics of the predicates in **Table 3.2**

Predicate	Meaning
`greater_than`(X, Y)	$X > Y$
`mult`(X, Y, Z)	$Z = X \times Y$

Table 3.2: Predicates in a natural number reasoning task

Now, with the introduction of the theory constant "0", we could express the concept "$W \times Z > W \times 0$ for any $Z \neq 0$" by the clause

$$\texttt{greater\_than}(X, Y) : - \texttt{mult}(W, Z, X), \texttt{mult}(W, 0, Y), Z \neq 0$$

3.1.3 Continuous Variables

This group of literal forms are more substantial. Relations encountered in real world are not limited to discrete information but commonly include numeric fields as well. We could imagine simple relations such as

$$\texttt{height}(X, H)$$

which provides the (numeric) height H of each person X. The system includes the following literal types to deal with numeric values.

- **Comparison of Numeric Variables**

$$V_i > V_j \ , \ V_i \leq V_j$$

where V_i and V_j are existing numeric variables. Along with the comparison of existing variables as discussed in **Section 3.1.2**, we could now have a full range of restrictions on the binding of values into numeric variables, which includes the constraints $=, \neq, >, \leq$.

- **Numeric Constant**

$$V_i > k \ , \ V_i \leq k$$

that allow an existing numeric variable V_i to be compared against a numeric constant, or a ***threshold*** k of the same type. Unlike the theory constants discussed in **Section 3.1.2**, which are explicitly defined by the user in the database, the thresholds are automatically "discovered" by the system.

It might seem that tests on numeric variables would be difficult to formulate, since they contain arbitrary thresholds. This is not so: the method for finding such thresholds can described very concisely. The previous algorithm for finding appropriate thresholds against which to compare the values of continuous attributes are found in Quinlan's C4.5 [30], and the same method is used in FOIL. The method is a linear search on the given numeric values in the training set. First, the training cases are sorted on the numeric domain A being considered. As there are only a *finite* number of these values, so let us denote them in order as $\{v_1, v_2, \ldots, v_m\}$. Any threshold value k lying between v_i and v_{i+1} will have the same effect of dividing the cases into those value of the attribute A lies in $\{v_1, v_2, \ldots, v_i\}$ and whose those value is in $\{v_{i+1}, v_{i+2}, \ldots, v_m\}$.

Thus, there are only $m-1$ possible splits on A, all of which are examined one by one.

Although this method can be carried out in one pass when the cases have been sorted as above, it is obviously expensive to examine all $m-1$ such thresholds. So, here we propose a bisection search, rather than the linear search used in C4.5 and FOIL, to choose the optimal threshold. To illustrate, let's consider the relations `greater_than` and `mult` as defined in **Table 3.2**. Suppose now we want to complete the program

$$\texttt{greater\_than}(X, 20) : - \texttt{mult}(4, Y, X), ?$$

by appending a suitable literal. The program represents the knowledge "4 multiply by any natural number greater 5 will give a result greater than 20". Given the sorted training domain as

$$A = \{0, 4, 5, 7, 9, 11, 13, 17, 19, 20, 21, 23, 25, 27, 31, 33, 37\} \quad (3.4)$$

We denote the positive example set as $\oplus$ and the negative example set as $\ominus$. The bisection process is shown on **Fig. 3.2**. First, the mid-point in A will be chosen to be the threshold to examine, i.e. $k = 19$. It results in two partitions by appending $Y > 19$ or $Y \leq 19$ to the clause. We see that both $\oplus$ and $\ominus$ examples exist in the $Y \leq 19$ partition while the $Y > 19$ partition comprises $\oplus$ examples only. That implies that a *smaller* threshold should be chosen, thus the mid-point of the smaller half set of examples are going to be examined, i.e. $k = 9$. The process iterates until an optimal threshold is found to produce two partitions, of which one contains only $\oplus$ examples and the other contains only $\ominus$ examples. Finally, the literal give raise to the pure $\oplus$ set will be appended to the clause.

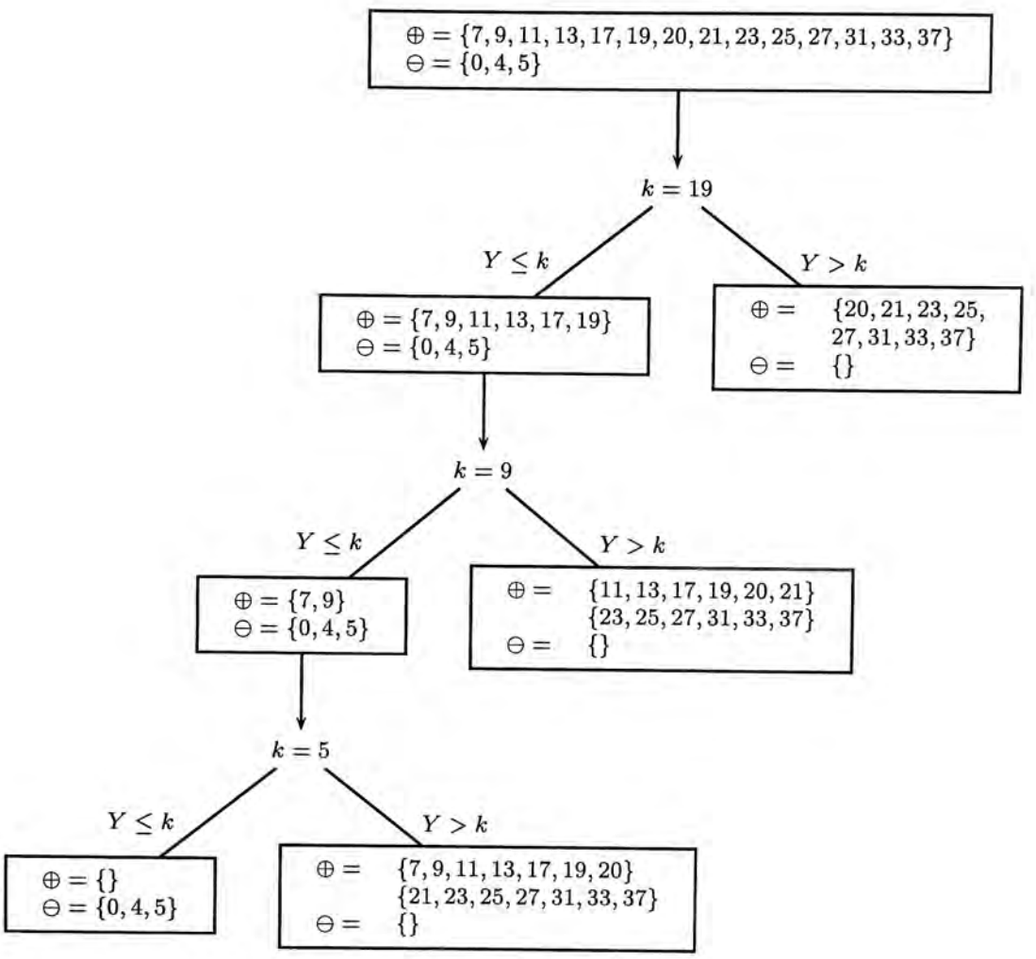


Figure 3.2: A bisection procedure to choose the threshold k

Following the bisection algorithm discussed above, we get the optimal threshold be $k = 5$ and the literal $Y > 5$ is appended to the clause to complete the program

$$\texttt{greater\_than}(X, 20) : - \texttt{mult}(4, Y, X), Y > 5$$

Note that this bisection search works *only* if there exists a threshold to partition the examples *perfectly*. We could easily see that the algorithm fails if the $\oplus$ and $\ominus$ examples have no clear cut into two sides by a single threshold. Say, when the $\oplus$ examples are located in "center" as

0	4	5	7	9	11	13	17	19	20	21	23	25	27	31	33	37
$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$

would be a typical situation to founder the bisection algorithm. It requires two thresholds for two literals, say, $Y > 7$ and $Y \leq 25$, to bound the $\oplus$ region. However, such situations are best described by a fuzzyifing the numeric attribute into several fuzzy predicates, which will be described in the coming section. Whenever the bisection method fails, all the remaining possible threshold values will be examined and the optimal one would be chosen. So, it's always no bad to perform the bisection method first. If it successes, it will reduce the complexity from $O(m)$ to $O(log(m))$; if not, the complexity still remains at $O(m)$ as the linear search does.

The previous two literal forms are the extensions in later version of FOIL. These extensions permit bound numeric values to be used in conditions on the right-hand side of a clause. However, they fall a long way short of fuzzy facilities that allow inexact descriptions of numeric attributes. Our system use a more powerful representation language — the fuzzy first-order logic, obviously we will provide some literal forms that do not exist in FOIL or other non-fuzzy first-order learning systems. The new literal forms will be discussed as follows.

- **Automatic Generation of Fuzzy Predicates** A fuzzy predicate could be defined explicitly by the user by a fuzzy relation. On the other hand, the system will automatically generate fuzzy predicates for any numeric attribute. Particularly, five fuzzy predicates will be generated through the fuzzification of the numeric attribute by five trapezoidal membership functions respectively. A trapezoidal membership function depends on

four parameters as given by

$$f(x;a,b,c,d) = \begin{cases} 0 & : \quad x \leq a \\ \dfrac{x-a}{b-a} & : \quad a \leq x \leq b \\ \dfrac{d-x}{d-c} & : \quad c \leq x \leq d \\ 0 & : \quad d \leq x \end{cases} \tag{3.5}$$

where the parameters a and d locate the "feet" of the trapezoidal and the parameters b and c locate the "shoulders". A typical trapezoidal function is displayed on **Fig. 3.3** with $a = 1, b = 4, c = 6, d = 8$.

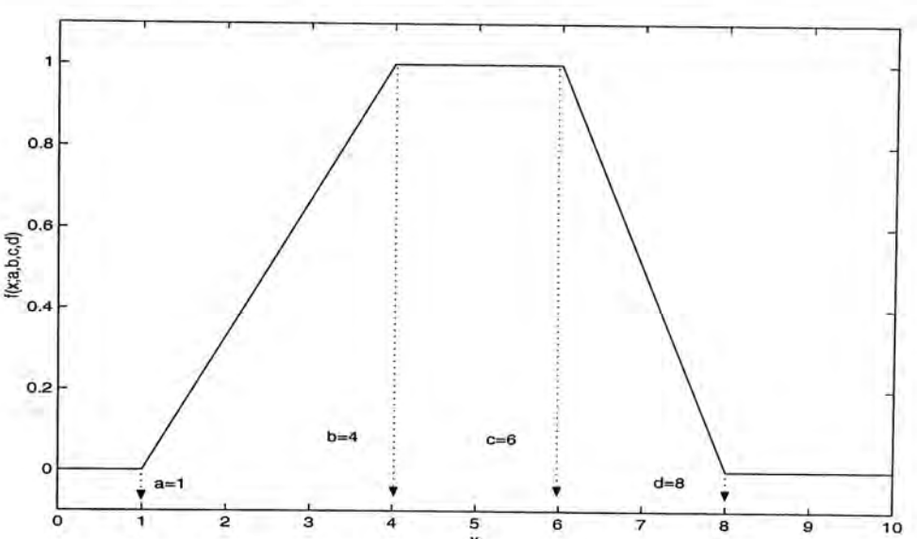


Figure 3.3: A typical trapezoidal function

For a numeric attribute A, five predicates, namely `very_small_A`, `small_A`, `medium_A`, `large_A` and `very_large_A`, will be generated. The usual rules of fuzzyification state that the when one fuzzy relation has a degree of membership being one, then those for the other fuzzy relations should be zero, say, $\mu_{\texttt{small\_A}}(x) = 1.0$ and all other $\mu_{(\bullet)}(x) = 0.0$. Moreover, the membership functions of *adjacent* fuzzy relation should intersect at 0.5, say, if $\mu_{\texttt{very\_small\_A}}(x) = 0.5$, then $\mu_{\texttt{small\_A}}(x) = 0.5$. By this way, the distribution of A would be partitioned into nine parts. To make an *initial*

guess for the parameters of the trapezoidal functions, we *evenly divide* the distribution of A into nine partitions. That is, the 0% percentile point, the 100% percentile point and the $n \times 11.1\%$, $n = 1, 2, \ldots, 8$ percentile points are chosen to be the parameters. To illustrate, consider the numeric attribute A in (3.4), the values of the ten percentiles are calculated as in **Table 3.3** then the numeric attribute A is fuzzyified into five

Percentile	0	11.1	22.2	33.3	44.4	55.5	66.6	77.7	88.8	100
Value	0	4.4	7.5	11.3	17.1	19.9	22.6	26.4	32.2	37

Table 3.3: The percentiles of A are chosen to be the parameters in the fuzzyification process

fuzzy predicates, their corresponding membership functions are shown in **Table 3.4** and are plotted in **Fig. 3.4**. We can see that the trapezoidal functions (they are not plotted in "typical trapezoidal shape" as the number of samples is small) fullfill the constraints of fuzzyification.

A	$\mu_{\texttt{very\_small\_A}}(A)$	$\mu_{\texttt{small\_A}}(A)$	$\mu_{\texttt{medium\_A}}(A)$	$\mu_{\texttt{large\_A}}(A)$	$\mu_{\texttt{very\_large\_A}}(A)$
0	1.00	0.00	0.00	0.00	0.00
4	1.00	0.00	0.00	0.00	0.00
5	0.81	0.19	0.00	0.00	0.00
7	0.17	0.83	0.00	0.00	0.00
9	0.00	1.00	0.00	0.00	0.00
11	0.00	1.00	0.00	0.00	0.00
13	0.00	0.71	0.29	0.00	0.00
17	0.00	0.02	0.98	0.00	0.00
19	0.00	0.00	1.00	0.00	0.00
20	0.00	0.00	0.98	0.02	0.00
21	0.00	0.00	0.61	0.39	0.00
23	0.00	0.00	0.00	1.00	0.00
25	0.00	0.00	0.00	1.00	0.00
27	0.00	0.00	0.00	0.90	0.10
31	0.00	0.00	0.00	0.21	0.79
33	0.00	0.00	0.00	0.00	1.00
37	0.00	0.00	0.00	0.00	1.00

Table 3.4: The five fuzzy predicates generated for A

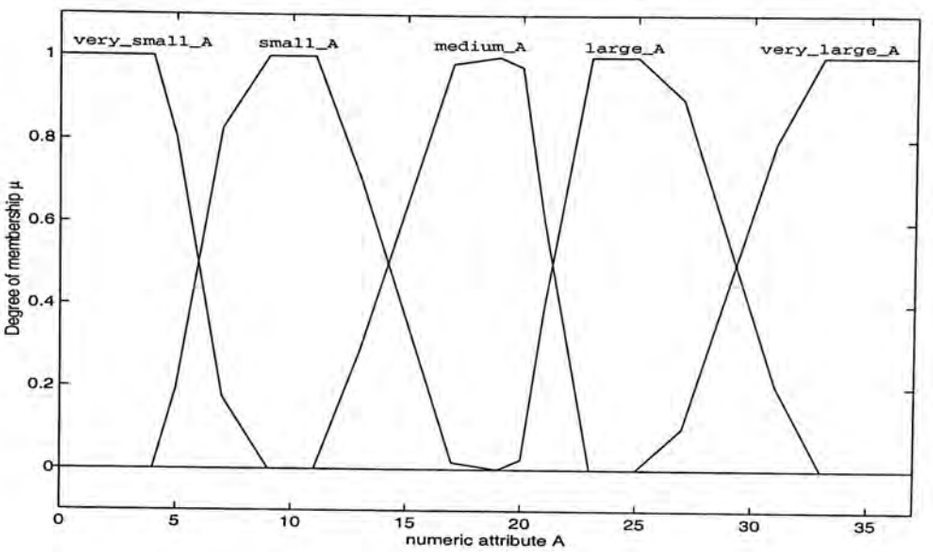


Figure 3.4: The plots for **Table 3.4**

The percentiles provide a set of adaptive parameters for the fuzzyification process. However, the set of parameters may not be the best description of some particular concepts. Thus, each fuzzy predicate will undergo a fine tuning procedure after the concept description C is learned. The fine tuning acts as a postprocessing procedure in the whole algorithm flow, it will be discussed in detail in **Section 3.2.2**.

- **Fuzzy Comparison**

$$V_i \cong k \ , \ V_i \cong V_j$$

where V_i and V_j are existing variables, and the threshold k are all in compatible numeric domains. These two literal forms have the similar semantics as the forms $V_i = k$ and $V_i = V_j$ respectively. However, those two literal forms are representing *crisp* relations. Using a fuzzy comparison operator $\cong$, which means "approximately equals" or "almost the same", we could now represent fuzzy relations. The actual implementation of the $\cong$ operator is quite arbitrary. In our system, we keep on using the trapezoidal function as the fuzzyification process. The choice

of the parameters in the trapezoidal function is nevertheless more intuitive. Given a numeric domain A sorted ascending with no repeating value as $\{v_1, v_2, \ldots, v_m\}$, and the threshold k equals the i-th value, i.e. $k = v_i$. We simply define the trapezoidal function with v_i be the center. Following the notation in (2.10), the trapezoidal fuzzy set for $\cong$ is defined as:

$$\begin{aligned} F_{\cong} = \{&0.0/v_1 + 0.0/v_2 + \cdots + 0.0/v_{i-4} \\ &+0.33/v_{i-3} + 0.67/v_{i-2} + 1.0/v_{i-1} + 1.0/v_i \\ &+1.0/v_{i-1} + 1.0/v_{i+1} + 0.67/v_{i+2} + 0.33/v_{i+3} \\ &+0.0/v_{i+4} + \cdots + 0.0/v_{m-1} + 0.0/v_m\} \end{aligned} \tag{3.6}$$

Note that the artificially constructed membership function may not be in perfect trapezoidal shape because the degrees of membership are fixed by the *order* of the unique numbers, regardless of the *actual value* of the numbers. Consider the numeric domain A in (3.4), the fuzzy relation $X \cong 20$ is represented as the membership function artificially constructed in **Fig. 3.5**.

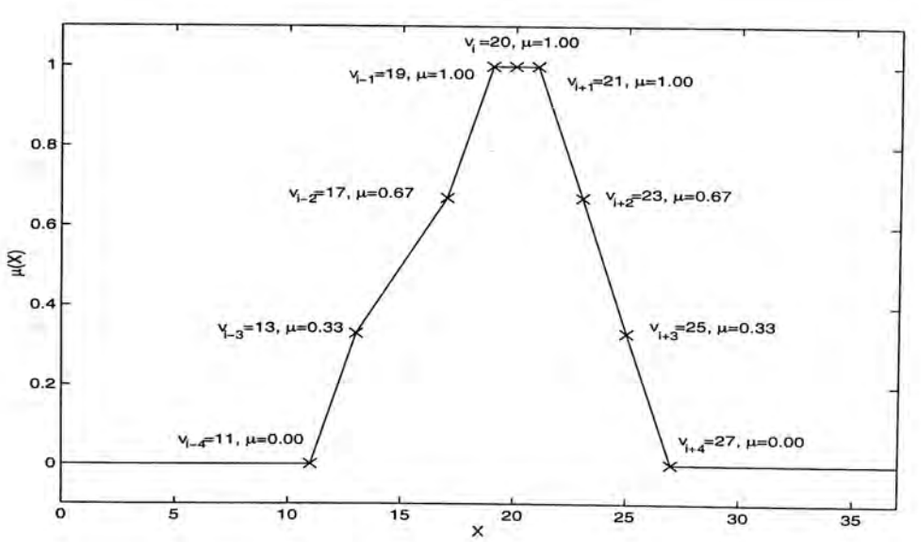


Figure 3.5: The membership function for the fuzzy relation $X \cong 20$

For the search of the best threshold k, now we cannot the bisection method as in **Fig. 3.2**. Because the literal $X \cong k$ is a fuzzy relation, we could not directly classify each sample into the $\oplus$ or the $\ominus$ set. Although the generalized covering concept is one of the main researches in this thesis, the concepts of fuzzy positive set $\tilde{\oplus}$ and fuzzy negative set $\tilde{\ominus}$ are not used in the search of the optimal k, mainly for the simplicity in the implementation and the time saving issues. Thus, a linear scan is done through all the possible value of k. That is, for the numeric domain $A = \{v_1, v_2, \ldots, v_m\}$, m fuzzy predicates (in form of unary fuzzy relation), namely, $X = v_1$, $X = v_2$, $\ldots$, $X = v_m$, will be generated for the variable X being investigated. Each of the fuzzy predicates will be evaluated through the heuristics in the learning algorithm, and they will be compared with other forms of predicates in order to make a best choice of literal to add to the concept definition C.

The final literal form, $V_i = V_j$, is somewhat the same with the form $V_i = k$. However, this form involves two variables, thus the fuzzy predicate is now equivalent to a binary fuzzy relation $\texttt{fuzzy\_equal}(V_i, V_j)$. The construction of this binary fuzzy relation is similar to that of the unary fuzzy relation in (3.6). It's best to use a table to present the binary fuzzy relation as in **Table 3.5**

$\mu_{\texttt{fuzzy\_equal}}(V_i, V_j)$		V_i												
		v_1	...	v_{k-4}	v_{k-3}	v_{k-2}	v_{k-1}	v_k	v_{k+1}	v_{k+2}	v_{k+3}	v_{k+4}	...	v_m
V_j	v_1	1	...	0	0	0	0	0	0	0	0	0	...	0
	$\vdots$							$\vdots$						
	v_{k-4}	0	...	1	1	0.67	0.33	0	0	0	0	0	...	0
	v_{k-3}	0	...	1	1	1	0.67	0.33	0	0	0	0	...	0
	v_{k-2}	0	...	0.67	1	1	1	0.67	0.33	0	0	0	...	0
	v_{k-1}	0	...	0.33	0.67	1	1	1	0.67	0.33	0	0	...	0
	v_k	0	...	0	0.33	0.67	1	1	1	0.67	0.33	0	...	0
	v_{k+1}	0	...	0	0	0.33	0.67	1	1	1	0.67	0.33	...	0
	v_{k+2}	0	...	0	0	0	0.33	0.67	1	1	1	0.67	...	0
	v_{k+3}	0	...	0	0	0	0	0.33	0.67	1	1	1	...	0
	v_{k+4}	0	...	0	0	0	0	0	0.33	0.67	1	1	...	0
	$\vdots$							$\vdots$						
	v_m	0	...	0	0	0	0	0	0	0	0	0	...	1

Table 3.5: The membership function constructed for the fuzzy relation $V_i \tilde{=} V_j$

Let's consider the numeric domain A in (3.4) again. The plot of the binary fuzzy relation becomes a 3-D visualization in **Fig. 3.6**. Again, the hill displayed is not in exact trapezoidal shape because we consider the order rather that the actual value of the entities in a numeric domain.

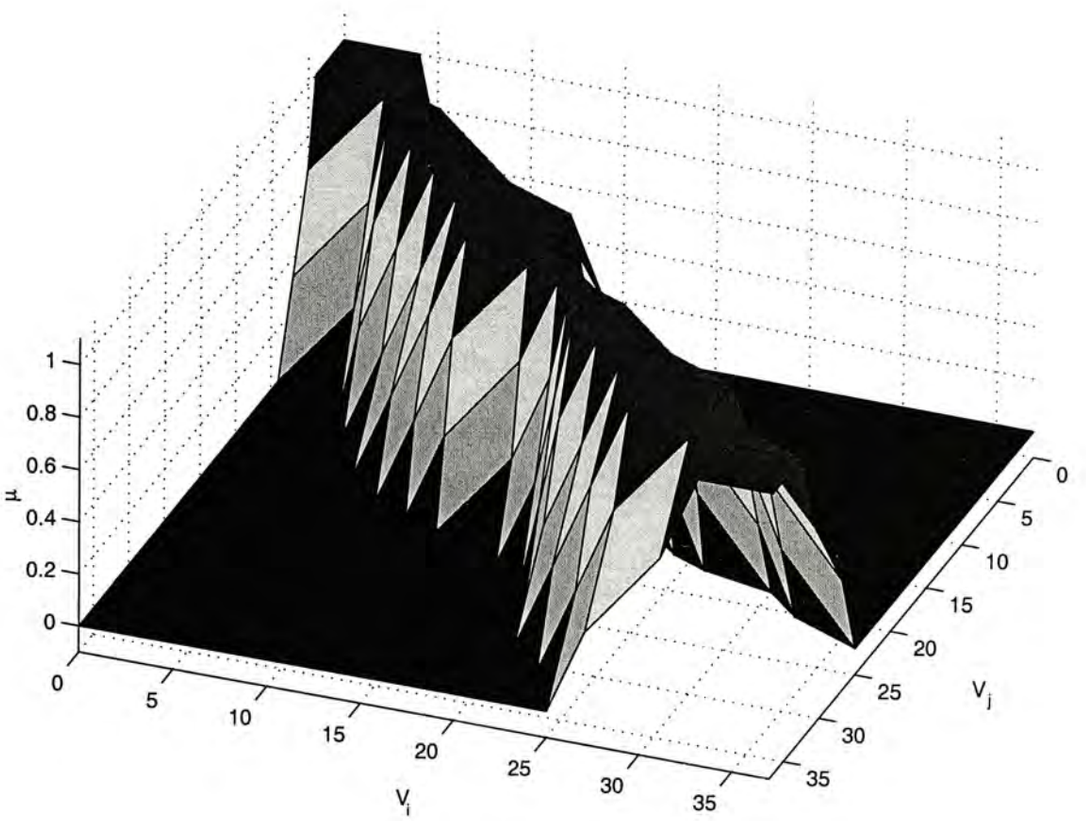


Figure 3.6: The fuzzy relation $V_i \tilde{=} V_j$, $V_i, V_j \in A$

The propositional attribute-value formalism is simple but limited. The FOIL opened up a new machine learning area by the use of first-order logic formalism. Our system moves one important step forward to accept the expression of fuzzy as well as relational knowledge. Having specified a more powerful language in this section, the next step is to develop learning methods capable of exploiting it.

3.2 System Architecture

3.2.1 Data Reading

The system is code-named FF01, meaning a fuzzyified FOIL that written in year 2001. The script file "FF01.m" is written in the Matlab 5.3 language. It is invoked in the Matlab environment by the command:

`FF01('`*option argument*`', '`*option argument*`', ... )`

As FF01 is considered as an extension of FOIL, it maintains some of the options available in FOIL, and also adds some options especially for our system. The options and their meanings are listed in **Table 3.6**. Note that the options 'FOIL', 'FF99', 'AlphaCover' and 'SlopeCover' are handled with the preference : SlopeCover>AlphaCover>FF99>FOIL . So, the command `FF01('i apple.d', 'n 1', 'vverb 2', 'FOIL 1', 'AlphaCover 1')` invokes the system with no negated literals allowed, verbosity level equals 2 and using the α-covering learning algorithm to manipulate the input file 'apple.d'.

As the system is learning a novel fuzzy first-order logic program, we define a new file format for data input. The file format is *downward compatible* to that of FOIL v6.4, i.e., it is a *extended* FOIL input file format. So that, we could use the test data of FOIL directly and perform some comparisons between these two systems. The extensions mainly concern in defining fuzzy relations, in the aspect that FOIL and other first-order learning systems unable to do. The input file format consists of three main blocks, it is defined in **Fig. 3.7**.

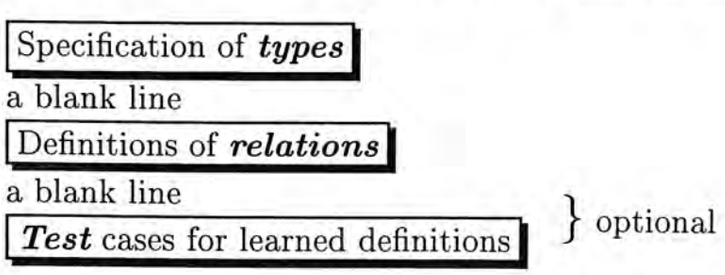


Figure 3.7: Block diagram of the input file format

The details of each of the blocks are given as follows:

Types

Each constant or variable in the system is conformed with a ***t***ype. A type is a domain that defines the universe of discourse for a particular collection of objects. For examples, the type "people" may contain the constants "John", "Peter", "Mary", while the type "natural number" may contain the constants "0", "1", "5", etc. The specification of types is also important in the binding of variables, say, the variable X is of type "people", then it is eligible to bind "John" to X, while the binding of "1" to X is inappropriate. The system accepts two kinds of type specifications, namely the ***discrete type*** and the ***continuous type***. The former kind is suitable to describe *nominal* and *ordinal* measurements. And the latter kind is suitable for *interval* and *ratio* measurements.

In our system, each ***discrete type*** specification consists of the type name followed by a colon, then a series of constants separated by commas and terminated with a period. This may occupy several lines and the same constant can appear in may types. There are three kinds of discrete types:

- **Nominal types** — type name preceded by '#'
  The constants in a nominal type are unordered. No assumptions are made about the quantitative difference between the constants. So, the user is free to put any order on the constants in this type. For example, the nominal type "sex" that contains the constants "M" and "F" may be specified by the line:

  ```
  #sex:  M, F.
  ```

- **Ordinal types** — type name preceded by '*'
  The constants in a ordinal type indicate the order of categories, but not the quantitative distances between them. The constants are specified in ascending order. For instance, the clothing sizes may be specified by the ordinal type:

```
*size:  XS, S, M, L, XL.
```

- **General discrete types** — type name with nothing preceding
  They are possibly ordered types. Following the way in FOIL, we will attempt to discover an ordering of the constants that may be useful for handling of recursive definitions. Although the discovery of natural ordering of constants is done before the learning of the concept definition, we handle the recursive definitions in the postprocessing time, so, this idea will be discussed further in **Section 3.2.2**.

The system does not distinguish between interval types and ratio types. They are all called the ***continuous types***. The categories of an interval measurement reflect exact measurement with standardized units, and the units are equally spaced. Thus, we could measure the *precise difference* between two categories. Ratio measurement has all of the characteristics of interval measurement plus an *absolute zero point*. At the point of absolute zero, it may be said that there is none, or an absence, of whatever characteristics is being measured. Raios make sense only with ratio variables. Since our system does not provide the functionality of number calculations, we won't calculate the ratio between numbers. So that, although our system can identify a numeric constant, the constant is still considered to be an atom, just as the same way as discrete constants.

Each continuous type specification consists of the type name followed by ": continuous." on one line. The constants corresponding to a continuous type are the usual integers and real numbers. In fact, any string that can be converted to a float by C's atof() should work when specifying a value in a tuple. For instance, the type "age" may be specified by the line:

```
age:  continuous.
```

Constants

A non-numeric constant consists of any string of characters with the exception that any occurence of certain delimiter characters must be prefixed by the escape character '\'. The delimiter characters are '(' , ')' , ',' , '.' and ';'. A ***theory constant*** that can appear in a definition should be preceded by a '*'. For instance, the line

```
*opair:  *\(0,0\), *\(1,1\), \(2,3\), \(3,5\), \(4,4\), \(5,3\).
```

defines a continuous type "ordered pair" that contains six constants. The first two theory constants are preceded by '*', meaning that they could appear in the definition directly. Note that the parenthesis are required to be prefixed by '\' as they are the delimiter characters.

Two one-character constants have a special meaning and should not be used otherwise. They are listed in **Table 3.7**.

The handling of those two special characters in **Table 3.7** will be in **Section 3.2.2**.

Relations

In FOIL, all relations are defined in terms of the set of positive tuples of constants for which the relation is true, and optionally the set of negative tuples of constants for which it is false. Our system extends the definition of relations by fuzzy tuples of constants for which the relation is true to a degree from zero to one. If only positive tuples and fuzzy tuples are given, all other constant tuples of the correct types are considered to be negative. It is called the ***closed-world assumption.***

Each relation is defined by a ***header*** and one or two or three sets of ***constant tuples***. The header can be specified as follows :

```
name(type, type, ... , type)
```

In the original FOIL input file format, one could optionally specify a set of

keys to limit the ways the relation may be used. This feature is dropped in our system for simplicity, however, for the compatibility with the original FOIL input file format, the user could still specify a set of keys at the end of the relation head section, yet they are meaningless in our system. The header of all relations other than ***target relations*** begins with '*'. The argument types limit the ways the variables may be bound to constants of different domains.

Following the header line are a series of lines containing constant tuples:

positive tuple
positive tuple
⋮
; ⎤
negative tuple
negative tuple
⋮ optional
; ⎦
fuzzy tuple
fuzzy tuple
⋮ ⎦
.

Each positive tuple or negative tuple consists of constants separated by commas and must appear on a single line. Each fuzzy tuple consists of constants separated by commas, and is trailed by the degree of membership of that tuple, which is a real number ranged between zero and one. The character ';' separates positive tuples from negative tuples, and negative tuples from fuzzy tuples, which are optional. Note that if the user wants to specify some fuzzy tuples, he should specify at least one positive tuple and one negative tuple beforehand. The following input file example defines a continuous type "age" and a relation "much_older_than", which is a fuzzy relation.

```
age:  continuous.

much_older_than(age, age)
80,1
80,20
60,1
60,20
40,1
20,1
;
1,1
1,20
1,40
1,60
1,80
20,20
20,40
20,60
20,80
40,40
40,60
40,80
60,60
60,80
80,80
;
80,40,0.8
80,60,0.3
60,40,0.2
40,20,0.5
.
```

Tests

The optional test relations may be given to test the learned Horn clause definitions. The addition input consists of

a blank line (indicating start of test relation specification)

relation name

test tuples

.

relation name

test tuples

.

⋮

Each test tuple consists of a constant tuple followed by ": +" if it is totally belongs to the relation, ": -" if it is totally not belongs to the relation and ": μ" if it is partially belongs to the relation, where μ is a real number ranged between zero and one.

Reading of Attribute-Value Dataset

Traditional first-order learning systems like FOIL and GOLEM are commonly demonstrated on abstract tasks such as learning recursive definitions of relations on lists of discrete constants. Such domains are markedly different from the real-world datasets successfully tackled by zeroth-order systems such as C4.5 [30], particularly when some attributes have continuous numeric values and when some attribute values are unknown. Quinlan addressed the problems in [16] by extending the original FOIL system to handle continuous numeric values and also missing values. The extensions are embedded in later versions of FOIL and in our system also. However, since most of the existing datasets in literature are designed for zeroth-order system, our system could not read the dataset directly. Among those attribute-value dataset, the C4.5 input file format becomes a standard. Fortunately, Mike Cameron-Jones wrote a program "c4tofoil.c" for converting files from the common C4.5 "*.data" format to the FOIL "*.d" format. For the use and listing of the program, please refer to **Appendix A**. By the use of this utility, which could be compiled in many platforms, we can test our system through some well-known attribute-value datasets. And the results can then be compared to some zeroth-order learning systems.

3.2.2 Preprocessing and Postprocessing

Preprocessing

We follow [31] to order the constants. The ordering is needed when learning recursive definitions. A recursive definition is a concept description that involves the target relation. The most obvious recursive definition to prevent is the binary symmetric relation, for instance:

```
close_to(X,Y) :-close_(Y,X)
```

is clearly meaningless.

Also, we have to calculate the percentiles for each numeric attributes. The percentiles are useful in generating literal in the fuzzyification process.

Postprocessing

After the concept description is learned, we are going to refine it. First, if there is any literal that is generated automatically through fuzzyification, the parameters of the trapezoidal membership function will be tuned. The tuning is done on the parameters a and d in (3.5) in a linear search manner.

Finally, the concept description will be polished by deleting each L in C. If the deletion of that literal does not worsen the $COST$, it will be removed. The process iterates until no literal could be deleted. This process is important as our system is performing a greedy search, which shortsighting is common.

Option	Meaning
i	The input file name. If it is not specified, the system will prompt the user to type in the input file name in run time. The path and file name follows the convention on that platforms the Matlab running.
n	Negated literals are not considered. This may be useful in domains where negated literals wouldn't make sense, or if the learned definitions must be formal Horn clauses.
N	This is similar to the option 'n', but permits negated equality literals in form of $V_i \neq V_j$ and $V_i \neq c$.
nauto	This option suppresses the automatic generation of literal forms for continuous attributes.
vverb	Set verbosity level [0,1,2,3 or 4; default: 1]. The program produces rather voluminous trace output controlled by this option. The default value of 1 gives a fair amount of detail; 0 produces very little output; 3 gives a blow-by-blow account of what the system is doing; 4 gives details of tuples in training sets etc.
Vvars	Set the maximum number of variables that can be used during the search for a definition. [default: 52]
mmaxt	Set the maximum number of tuples [default: 100000]. If the the default setting results in warnings that literals are being excluded due to the tuple limite, expanding the limit may be useful, but time-consuming.
FOIL	Use the FOIL learning algorithm if possible. This option suggests the system to apply the FOIL algorithm *if* there is *no fuzzy tuple* specified in the dataset. If there is any fuzzy tuple, this option is forbidden.
FF99	Use the FF99 learning algorithm. This option *forces* the system to apply the global literal evaluation discussed in **Chapter 4**.
AlphaCover	Suggest the system to use α-covering algorithm. This option *forces* the system to apply one of the partial literal evaluation methods discussed in **Chapter 5**.
AdaptiveAlpha	This is the default learning algorithm in FF01. This option ensures the system to adopt this learning method. The method will be discussed in **Chapter 5**.

Table 3.6: FF01 options and their meanings

?	indicates a missing or unknown value
^	indicates an out-of-closed-world value

Table 3.7: Special one-character constants

Chapter 4

Global Evaluation of Literals

This chapter discusses several means of global evaluation of literals. A global evaluation of literal refers to the method that measures the effectiveness of adding a particular literal L_{ij} to the concept description C in (3.1). Nevertheless, a global evaluation does not investigate which elements in a literal is contributing to the effectiveness of the whole literal. That is, a global evaluation of literal gives no hint of which examples are "covered" or not. This method is highly related to the closeness measures between fuzzy sets. So, we overview some of the existing closeness measures between fuzzy sets in **Chapter 4.1**. Then, we start to develop our own global evaluation measures on the ground of the error function and the error distribution as discussed in **Chapter 4.2**. Furthermore, we would quantify some interesting observations of the error distribution by the application of the continuous information theory in **Chapter 4.3**. This is literal selection heuristics adopted in FF99 [32]. Finally, we show that our global evaluation method works fine in attribute-value datasets in **Chapter 4.5**.

4.1 Existing Closeness Measures between Fuzzy Sets

The concept of equality ($=$) for ordinary (non-fuzzy) sets can be extended to account for the closeness ($\approx$) for fuzzy sets. This extension plays an important role in many "matching" problems where the degree of closeness between fuzzy sets A and B needs to be evaluated [22]. The analysis of closeness measures is especially important for FF99, which is a heuristic learning system depending *solely* on the closeness estimation between first-order fuzzy sets. The closeness measures tested and used in FF99 will be discussed later, while this sub-section gives a survey of the currently well-known methods.

Let $F(U)$ be the set ot all fuzzy sets on U, then $\forall A, B \in F(U)$, a *closeness measure* between A and B, denoted as $\approx$, is a mapping: $F(U) \times F(U) \rightarrow [0, 1]$.

As observed in many fuzzy extensions, the closeness measures may be defined in various ways:

- The first approach is a straightforward extension on the notion of set *inclusion* ($\subseteq$). That is, two ordinary sets A and B are equal ($A = B$), if and only if $A \subseteq B$ and $B \subseteq A$. Now consider A and B are fuzzy sets, we can extend the concept of inclusion with Lukasiewicz's fuzzy implication operators ($L(a, b) = min(1, 1 - a + b)$) [33]. Thus,

$$degree(A \subseteq B) = \inf_x L(\mu_A(x), \mu_B(x))$$

  Finally, the inclusion-based degree of $A = B$ is defined as:

$$\begin{aligned}
\approx_I (A, B) &= min(degree(A \subseteq B), degree(B \subseteq A)) \\
&= min(\inf_x L(\mu_A(x), \mu_B(x)), \inf_x L(\mu_B(x), \mu_A(x))) \\
&= \inf_x min(L(\mu_A(x), \mu_B(x)), L(\mu_A(x), \mu_B(x))) \\
&= \inf_x min(min(1, 1 - \mu_A(x) + \mu_B(x)), min(1, 1 - \mu_B(x) + \mu_A(x))) \\
&= \inf_x min(1 - \mu_A(x) + \mu_B(x), 1 - \mu_B(x) + \mu_A(x)) \\
&= \inf_x (1 - |\mu_A(x) - \mu_B(x)|)
\end{aligned}$$

- Another closeness measure is based on the *commonality* of fuzzy sets. It is also referred to as the height of $A \cap B$, denoted as $\approx_H$:

$$\approx_H (A, B) = \sup_x min(\mu_A(x), \mu_B)x))$$

  For **Fig. 4.1**, $\approx_I (A, B) = a + c$ and $\approx_H (A, B) = b$

- The other class of closeness measures based on the concept of *distance* [34]. For example, a family of distance measures can be summarized as the Minkowski-distance measure:

$$d(A, B) = \left(\frac{1}{n} \sum_x |\mu_A(x) - \mu_B(x)|^s \right)^{1/2}$$

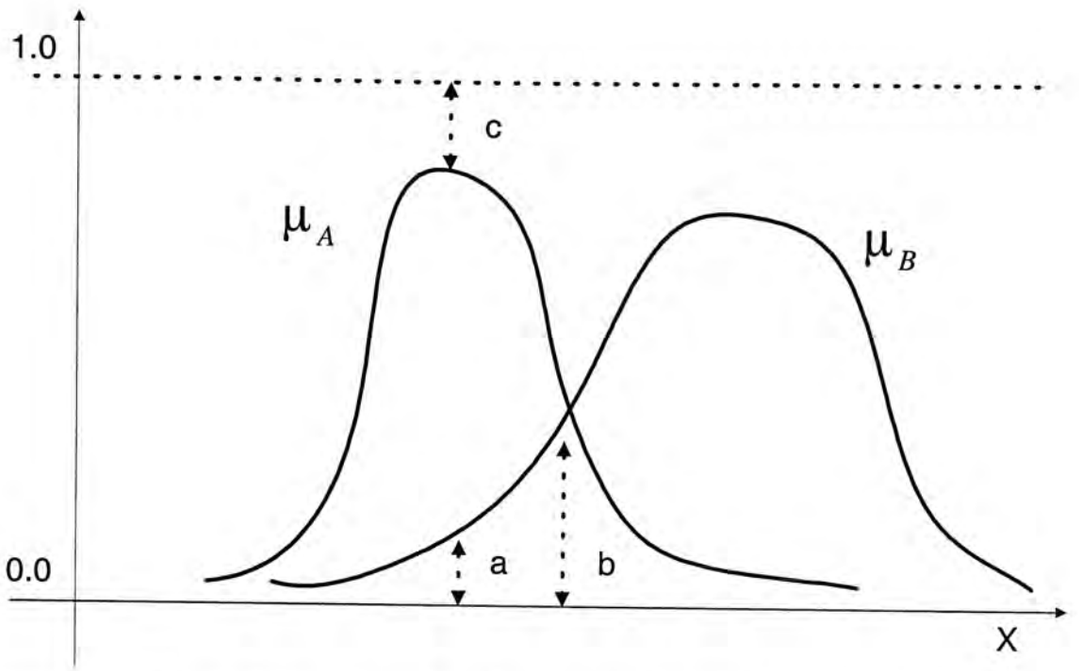


Figure 4.1: The inclusion-based and commonality-based closeness measure

where s is a parameter.

Generally, the distance-based closeness measure, denoted as $\approx_D$, is a mapping $F(U) \times F(U) \rightarrow [0,1]$ such that:

$$\approx_D (A, B) = 1 - d(A, B)$$

He [34] summarized some of other closeness measures found in the literature.

Notice that the three mentioned closeness measures satisfies $\approx (A, B) \in [0,1]$, $\approx (A, B) = \approx (B, A)$ and $\approx (A, A) = 1$

- There are some closeness measure developed for other specific applications. For examples, the similarity and the affinity measures discussed in [35]. The measures are proven to be a useful tool to determine the similarity between the fuzzy data in a fact base and the fuzzy patterns in the premise part of a rule. Thus, it is especially useful in the consistency checking for fuzzy expert systems.

4.2 The Error Function and the Normalized Error Functions

4.2.1 The Error Function

The several closeness measures reviewed in **Section 4.1** gives different aspects of evaluating the similarity between two fuzzy sets. However, they are not good enough to be the global evaluation of literals in our learning system. Hence, we're going to develop our own closeness measure, the reasoning would be apparent later on.

Remember that our goal is to learn a concept description C to describe the target concept T. We intuitively define the ***error function*** E to measure the difference of degree of membership of T and C as:

$$E(X) \triangleq \mu_T(X) - \mu_C(X) \tag{4.1}$$

where μ_T and μ_C are the membership functions of the target concept and the concept description respectively. The notation $E(X)$ means that the error function E is defined only on unary universe space. Thus, the learning method discussed in this whole chapter works only for attribute-value datasets but not first-order datasets. The calculation of E is straightforward, let us consider a unary space example to illustrate the calculation of E as in **Table 4.1**.

X	1	10	30	50	80
$\mu_T(X)$	0.0	0.1	1.0	0.9	0.0
$\mu_C(X)$	1.0	0.2	0.8	1.0	0.2
$E(X)$	-1.0	-0.1	0.2	-0.1	-0.2

Table 4.1: A simple example to illustrate the calculation of E

It is obvious that the error function always returns a real value in $[-1, 1]$. We could easily measure the closeness between T and C by some quantitative

calculations on E. One of the most commonly used candidate is the mean square error, which is defined as follows:

$$MSE = \frac{\sum_X (E(X))^2}{|X|} \tag{4.2}$$

The interpretation of the MSE measure is: $MSE = 0$ means that the concept description perfectly matches the target concept, where $MSE = 1$ means that C and T are totally different. We choose the MSE measure mainly because it is simple and easy to implement, also it is widely accepted and sign insensitive. If we adopt the MSE to be the global evaluation method in our learning algorithm, we simply pick the literal to add to C that would give rise to the minimum MSE. And hopefully, we could develop a C that gives $MSE = 0$ eventually.

4.2.2 The Normalized Error Functions

However, it's not our only purpose to work out the error function in order to facilitate the calculation of MSE. As you can see, the MSE measure takes no special advantage over other closeness measures as discussed in **Section 4.1**. In fact, the calculation of the error function is the essential step before we can work out the ***normalized error functions***. The error function gives us information about the *quantitative* difference between two fuzzy sets. While the normalized error functions enable us to analyze the *shape* of the error function.

Before we work out the normalized error functions, we first "split" the error function $E(X)$ into two functions, namely the positive error function $E_\oplus(X)$ and the negative error function $E_\ominus(X)$. They are defined as follows:

$$E_\oplus(X) = \begin{cases} E(X) & : \quad E(X) > 0 \\ 0 & : \quad E(X) \leq 0 \end{cases} \tag{4.3}$$

$$E_\ominus(X) = \begin{cases} -E(X) & : \quad E(X) < 0 \\ 0 & : \quad E(X) \geq 0 \end{cases} \tag{4.4}$$

we can see that $E_{\oplus}(X) - E_{\ominus}(X) = E(X)$.

Now we are ready to define the normalized error functions. A normalized positive (negative) error function $E'_{\oplus}(X')$ ($E'_{\ominus}(X')$) is the scaled positive (negative) error function, which has an area equals 1. The normalization take the following steps:

1. Map the universe space X to X' such that $X' = [0\ 1]$. For numeric type X, a linear scaling and shifting is done. For ordinal type X, we assume a uniform space between each sample $x \in X$, i.e. $x_i - x_j = x_j - x_k$. And the whole ordinal type X is mapped into $X' = [0\ 1]$ in a way that the largest $x \to 1$ and the smallest $x \to 0$. The normalization does not work for nominal universe, in fact it is one main restriction to apply the FF99 algorithm, the assumptions for FF99 will be discussed in **Section 4.3**.

2. Calculate the area for the positive (negative) error function over the normalized space

$$a_{\oplus} = \int_0^1 E_{\oplus}(X')dX' \tag{4.5}$$

and

$$a_{\ominus} = \int_0^1 E_{\ominus}(X')dX' \tag{4.6}$$

3. Scale the positive (negative) error function to get a unit-area normalized positive (negative) error function

$$E'_{\oplus}(X') = \frac{E_{\oplus}(X')}{a_{\oplus}} \tag{4.7}$$

and

$$E'_{\ominus}(X') = \frac{E_{\ominus}(X')}{a_{\ominus}} \tag{4.8}$$

The above normalization procedure fails when $E_{\oplus}(X) = 0$ or $E_{\ominus}(X) = 0$. In those cases, the area of the error functions would be zero and we could never normalize them to get unit area. They will be treated as special cases. In fact, as we will see later, $E_{\oplus}(X) = 0$ or $E_{\ominus}(X) = 0$ means a partial perfect match.

Remember that we are using the normalized error functions to analyze the shape of the error functions. That's why we need to normalize the error function such that the absolute value of errors does not affect our shape analysis. The reason of the unit-area normalization will become apparent in next section when we try to quantify the shape of the errors. In **Fig. 4.2**, we could visualize the example error function in **Table 4.1**, also with its corresponding normalized error functions.

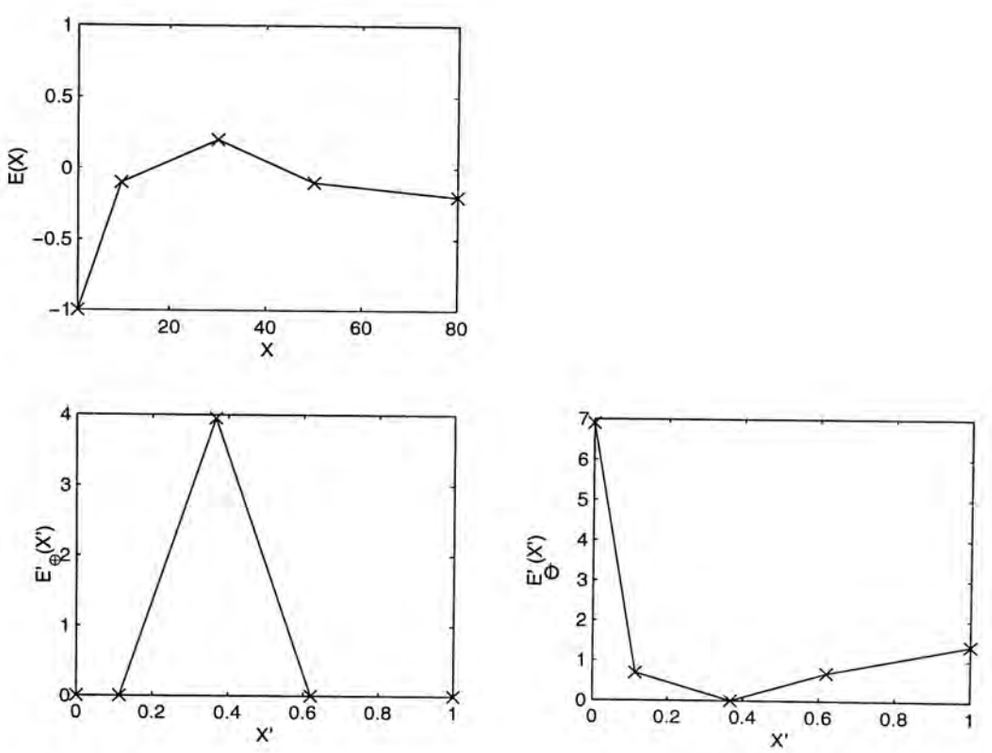


Figure 4.2: The plots of $E(X)$ in **Table 4.1**, also the generated $E'_{\oplus}(X')$ and $E'_{\ominus}(X')$

4.3 The Nodal Characteristics and the Error Peaks

4.3.1 The Nodal Characteristics

In last section, we detailed the calculations of the error function E and the normalized error functions $E'_{\oplus}$ and $E'_{\ominus}$. From the error function, we could directly measure the closeness between the target relation T and the concept description C through MSE. The reader might wonder why we need to work out the normalized error functions (although it's not very costly). In fact, we observed that the shape of the error function is highly related to the usefulness of a literal. We denote the observations by the ***nodal characteristics*** of literals and we claim that:

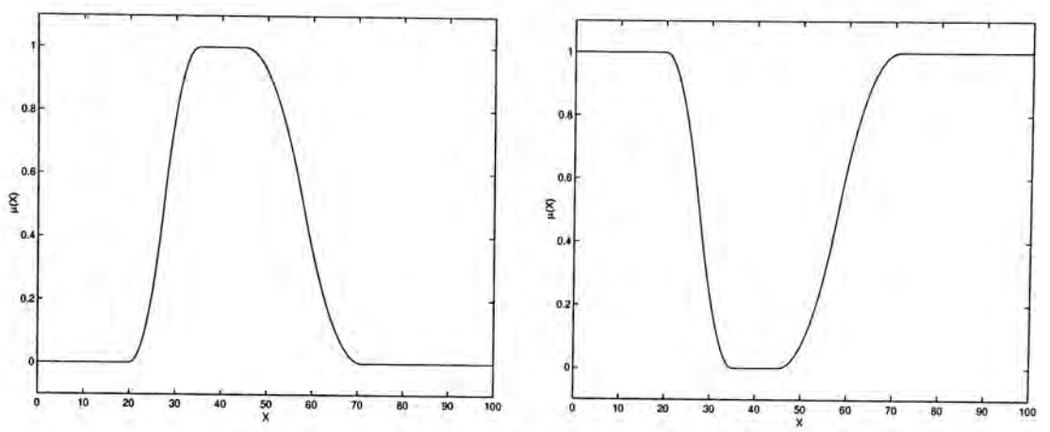
"A useful literal is likely to produce *nodal* normalized error functions in a way that $E'_{\oplus}$ (or $E'_{\ominus}$) has a large portion lies on the zero line and may has a few apparent peaks. While an improper literal is likely to produce random-shaped normalized error functions in a way that $E'_{\oplus}$ (or $E'_{\ominus}$) is peaked in the whole range of the universe."

The nodal characteristics gives us insight to select a suitable literal from the shape of the normalized error functions they produce. Note that there are two important assumptions to fulfill in order to obtain the nodal characteristics:

1. The target relation T can be described by a relatively simple C, say, no more than 3 clauses, and no more than 3 literals in each clauses. The more complex the concept description required, the more the nodal characteristics vanishes.

2. The membership function of each literal involved is *convex* in shape. This implies that the literal should be defined over an universe of an ordered type (e.g. ordinal types and numeric types), so that we can talk about the "shape" of the membership function. For the literal defined over an

nominal universe, basically we cannot visualize the membership function as each element and its degree of membership is an unique case, with no relation or ordering with other elements.



(a) (b)

Figure 4.3: (a) A convex membership function (b) A non-convex membership function

Remember that for any continuous type, the system would automatically generate five literals through the fuzzification process as discussed in **Section 3.1.2**. These literals are all in trapezoidal shape and thus satisfy this convex condition. Other candidates that satisfy this condition includes the literal form $V_i \cong k$ and some user-specified relations.

4.3.2 The Zero Error Line and the Error Peaks

In last section, we claim the nodal characteristics that a useful literal would produce some apparent peaks in the normalized error functions. We called these peaks the ***error peaks***. There are two types of error peaks, namely the ***positive error peak*** and the ***negative error peak***. We also claim that a useful literal will produce a long ***zero error line***. We are going to explain how these characteristics form and what they imply.

The Zero Error Line

As defined, the error function E gives the difference of the degree of membership between each sample of T and C. Imagine that there is a certain portion of samples have similar degree of membership in T and C, then we would get a certain portion of error $e_i = E(X_i),\ X_i \in X$ with $e_i \approx 0$. Furthermore, if C perfectly describes T, we will get $E(X) = E_{\oplus}(X) = E_{\ominus}(X) = 0$. We denote the portion of $E'(X) \cong 0$ be the *zero error line*. The longer the zero error line, the more samples in C matches T. In other words, whenever we observe that a literal results in a long zero error line, we can conclude that there is a certain large portion of samples in C is closely describing those in T.

The Positive Error Peaks

To illustrate the formation of the positive error peaks, we consider four literals $L_i,\ i = 1 \ldots 4$, of which their membership functions are plotted in **Fig. 4.4**. They are all defined on a continuous universe $X = [0\ 100]$. Suppose we have the target concept T that is the disjunction of two literals. The membership function of T is visualized on **Fig. 4.5**.

$$T :- L_1$$
$$T :- L_2$$

and we start with a null concept description, i.e. $C :-$. Now we have to select the suitable literal to append to C. Let us denote the error function for literal L_i be $E_i(X)$, we have:

$$\begin{aligned} E_i(X) &= \mu_T(X) - \mu_{L_i}(X) \\ &= max(\mu_{L_1}(X), \mu_{L_2}(X)) - \mu_{L_i}(X) \qquad (4.9) \\ &\quad (\because \mu_T = max(\mu_{L_1}, \mu_{L_2})\ ) \end{aligned}$$

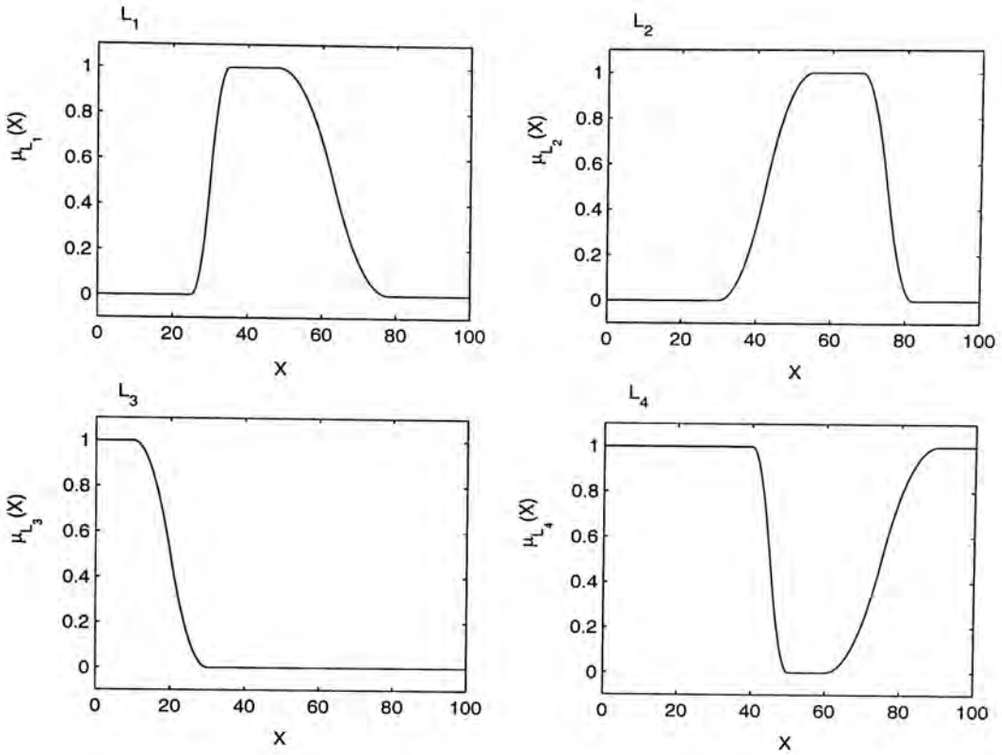


Figure 4.4: The membership functions of four literals

we can see that $E_1(X) \geq 0$ and $E_2(X) \geq 0$, but we cannot make wild guess on the range of $E_3(X)$ and $E_4(X)$ as L_3 and L_4 have no special relation with T. In **Fig. 4.6**, we can see that the zero error line constitutes a large portion in both E_1 and E_2, yet we cannot see that in E_3 and E_4. We can also observe that E_1 and E_2 are much more neat and tidy compared to E_3 and E_4.

Now we pick two literals, say E_1 and E_4 to analysis their normalized error functions as plotted in **Fig. 4.7**. Let us consider their corresponding normalized error functions be $E'_{\oplus 1}$, $E'_{\ominus 1}$, $E'_{\oplus 4}$ and $E'_{\ominus 4}$ respectively. We can see an apparent tall (relatively) peak in $E'_{\oplus 1}$, yet is not found in $E'_{\oplus 4}$ or $E'_{\ominus 4}$. We call it be the *positive error peak*, meaning that an apparent peak located in $E'_{\oplus}$. More importantly, L_1 does not produce the normalized negative error function as $E_1(X) \geq 0$. It implies that the learning of current clause is completed and we could start learning a new

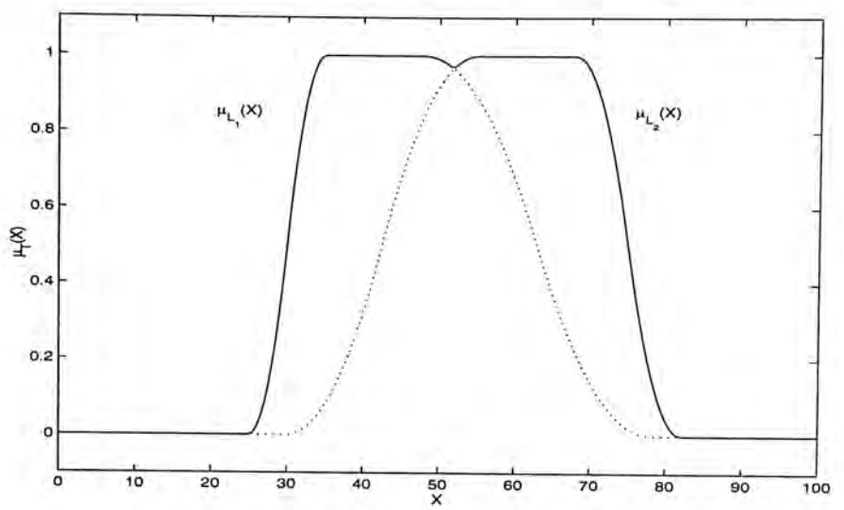


Figure 4.5: A target concept defined by the disjunction of two literals

clause.

In conclusion, the shape of the $E'_{\oplus}$ gives us hints about the usefulness of a literal. If a literal produces some neat and tiny positive error peaks and some long zero error peaks, we may consider adding it to the current clause. Particularly, if the literal produces only the positive normalized error function but not the negative one, we can conclude that adding this literal would terminate the learning of current clause. And the number of positive error peaks is very probably equal to the number of clauses needed to discover.

The Negative Error Peaks

The formation of negative error peaks is analogous to that of positive error peaks. However, this time they come from the incomplete description within the current clause to learn. To illustrate, consider the simple example that T is a single clause, which is a conjunction of two literals. The membership function of T is visualized on **Fig. 4.8**

$$T :- L_1, L_2$$

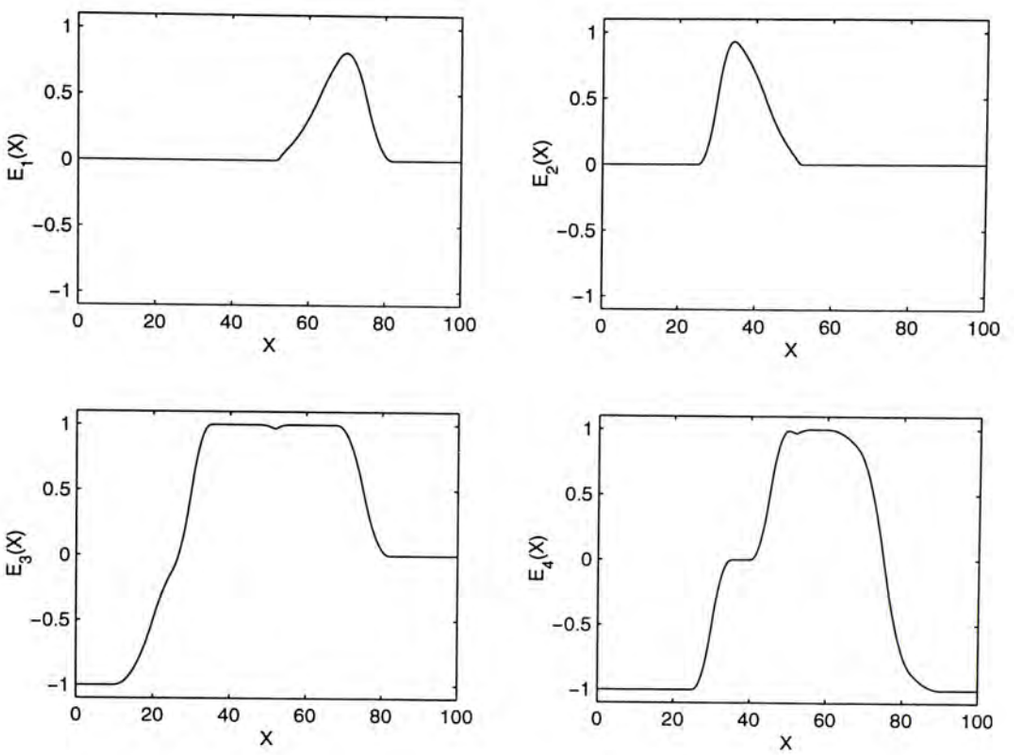


Figure 4.6: The error functions for appending the literals to C

Again, we start with a null concept description $C :-$ and we are going to select a literal to append to C. The corresponding error function in selecting literal L_i is denoted as $E_i(X)$, we have:

$$\begin{aligned} E_i(X) &= \mu_T(X) - \mu_{L_i}(X) \\ &= min(\mu_{L_1}(X), \mu_{L_2}(X)) - \mu_{L_i}(X) \\ &\quad (\because \mu_T = min(\mu_{L_1}, \mu_{L_2})\,) \end{aligned} \tag{4.10}$$

this time we can see that $E_1(X) \leq 0$ and $E_2(X) \leq 0$, but we cannot make such bound on $E_3(X)$ and $E_4(X)$ as L_3 and L_4 have no special relation with T. We skip some detailed steps and directly look at the normalized error function for L_1 and L_4 in **Fig. 4.9**. Since $E_1(X) \leq 0$, there is no normalized positive error function. The shape of $E'_{\ominus 1}$ is much more neat than $E'_{\ominus 4}$. Also, $E'_{\ominus 1}$ has a greater portion belongs to the zero error line compared with $E'_{\ominus 4}$.

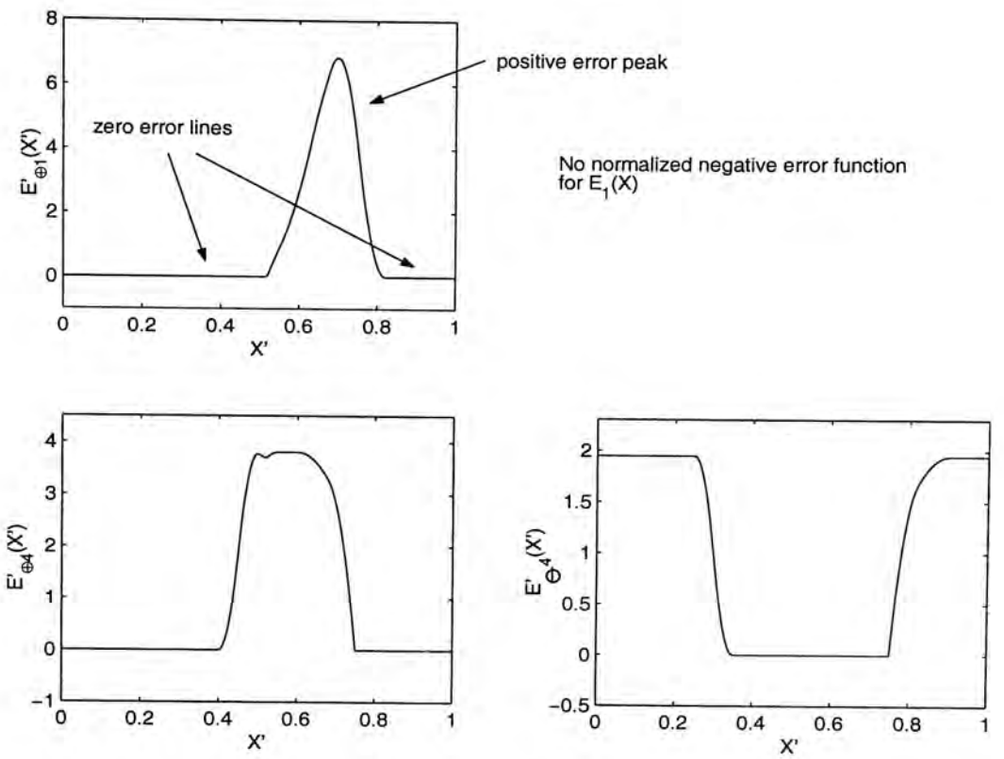


Figure 4.7: The normalized error functions for L_1 and L_4

This example demonstrates that when there is no more clause needed, i.e. the current clause is the final clause to learn, it is likely to get a error function that results in no normalization positive error function. And its normalization negative error function should contain some distinct peaks. The peaks are so called the *negative error peaks*, meaning that they locate in $E'_{\ominus}$. The number of negative error peaks possibly represents the number of literals need to learn in order to complete the whole concept description.

4.4 Quantifying the Nodal Characteristics

In last section, the nodal characteristics of literals tell us that the *shape* of the normalized error functions. The problem is: how do we *quantify* the shape of the functions? It's easy for human beings to distinguish a neat peak from a

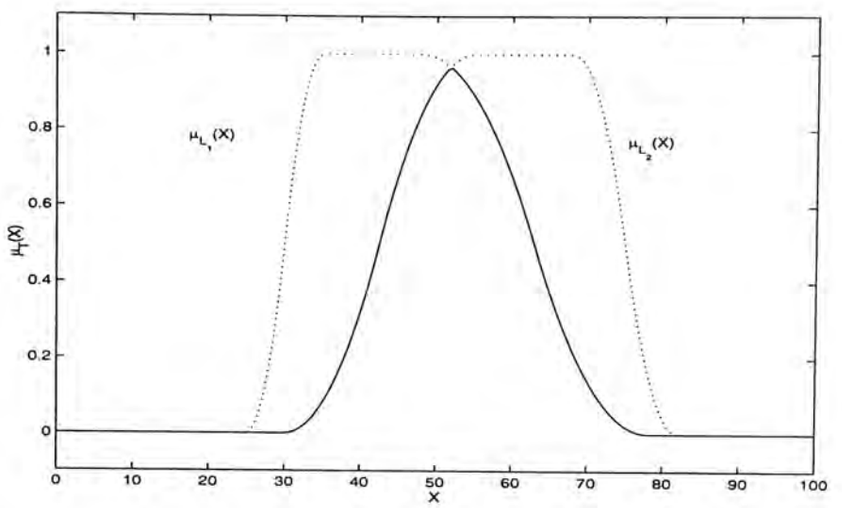


Figure 4.8: A target concept defined by the conjunction of two literals

mess. But the shape analysis of functions is a complex research topic in computer science. Moreover, we give no constraints on the shape of the normalized error functions, say, we don't know the number of turning points, we don't know if it could be expressed by the combination of some known kernel functions or not. Bascially, what we know are only $E'(X') \geq 0$, $\int_0^1 E'(X')dX' = 1$ and $E'(x_i) = 0, \forall x_i \notin [0\ 1]$. Thus, we need a general shape quantification method for any $E'(X)$ that satisfies the above two conditions. We choose the well-known information theory as our tools.

4.4.1 Information Theory

Shannon presented a mathematical foundation of the information theory in his pioneer paper "A mathematical theory of communication" in 1948 [36]. Shannon introduced the concepts of ***entropy*** and mutual information from the viewpoint of communication reliability. Entropy is defined as a measure of "uncertainty" or "randomness" of a random phenomenon. Suppose that some information about a random variable is received, then a quantity of uncertainty is reduced, and this reduction in uncertainty can be regarded as the quantity of transmitted information, which is called the mutual information.

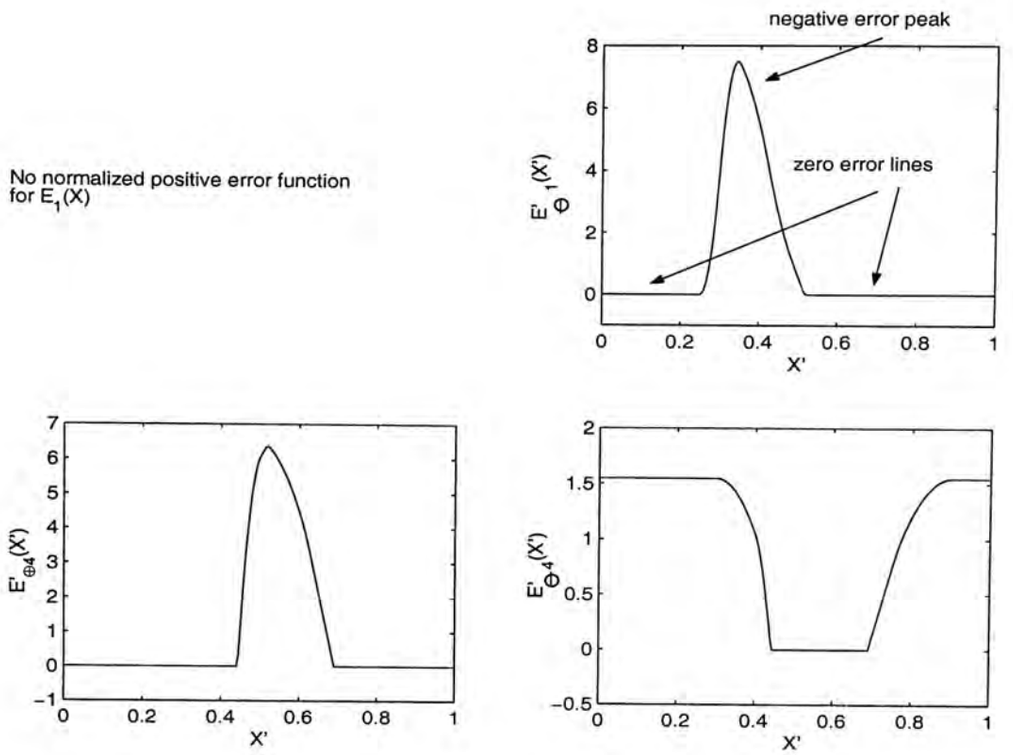


Figure 4.9: The normalized error functions for L_1 and L_4

The entropy H of a random variable of discrete distribution is defined as:

$$H(R) \triangleq -\sum_{r_i \in R} P(r_i) \times \log_2 P(r_i) \tag{4.11}$$

where R is a random variable with the distribution:

$$\text{Probability}(R = r_i) = P(r_i) \quad ; \quad \sum_{r_i \in R} P(r_i) = 1 \tag{4.12}$$

and the quantity $-\log_2 P(r_i)$ is the ***information*** received after $R = r_i$ occurred.

The notation of discrete entropy can be extended to handle continuous random variables [37]. Now consider R to be a continuous random variable and $f(r)$ is the ***density function***. A density function is defined by:

$$\text{Probability}(r_a \leq r \leq r_b) = \int_{r_a}^{r_b} f(r)dr \tag{4.13}$$

and a density function has the properties: $f(r) \geq 0$ and $\int_R f(r) = 1$.

The ***continuous entropy*** (CE) of a continuous random variable is then defined as:

$$h(R) \triangleq -\int_R f(r) \times \log_2 f(r) dr \tag{4.14}$$

it is also referred as the ***differential entropy***.

The interpretation of the discrete entropy and the continuous entropy is very similar. However, there are important differences:

1. The discrete entropy measures the uncertainty in an absolute way. However, a continuous entropy does not work as a measure of uncertainty by its value. By the way, the difference $h(R_1) - h(R_2)$ of entropies indicates the difference of the uncertainties of R_1 and R_2. That is, the continuous entropy measure the uncertainty in a relative way.

2. We cannot estimate the continuous information gain after receiving a single signal. The calculation involves the integration over a spectrum of signals.

3. In discrete case, the entropy represents the lower bound of average coding length in bits. In contrast, the continuous entropy is not related to the coding length.

Although the value of a continuous entropy has less meaning than that of a discrete entropy, the continuous entropy measurement is still very useful in *comparing* the uncertainty of different continuous distributions.

4.4.2 Applying the Information Theory

The continuous entropy has been proven to be very effective in characterizing the shape of density functions. In fact, it works well for any function that satisfies the condition $F(x) \geq 0$ and $\int_{-\infty}^{\infty} F(x)dx = 1$. We can see that the normalized error functions satisfy the conditions, thus they are the candidates to apply the continuous entropy measure. In fact, it is the reason

why we have to split $E(X)$ into $E_{\oplus}(X)$ and $E_{\ominus}(X)$, and then execute the normalization procedures. According to the nodal characteristics, a "suitable" literal produces "neat" normalized error functions; in contrast, an "unrelated" literal produces "mess" (relatively) normalized error functions. These observations could be quantified in terms of the randomness of the normalized error functions. Remember that now we treat the normalized error functions to be density functions, so the "randomness" of the normalized error functions should be considered in context of probability density.

We denote the continuous entropy measures on $E'_{\oplus}$ and $E'_{\ominus}$ be $CE_{\oplus}$ and $CE_{\ominus}$ respectively. They are defined as:

$$CE_{\oplus} \triangleq \int_0^1 E'_{\oplus}(x)dx \tag{4.15}$$

and

$$CE_{\ominus} \triangleq \int_0^1 E'_{\ominus}(x)dx \tag{4.16}$$

We will soon see that the a neat error function will gives a *smaller* CE than that of a mess error function.

4.4.3 Upper and Lower Bounds of CE

As we are using the CE to quantify the shape of error functions, we need to know the bounds of this measurement. Without the bounds, we don't know if it is good or not when getting a large CE, also we have no idea of what does the value of CE represent.

Upper Bound

The largest value of CE occurs if $E'(x)$ is a *uniform function*, i.e. $E'(x) = 1$ (as $\int_0^1 E'(x)dx = 1$). So, the upper bound of CE is:

$$\begin{aligned} CE &= -\int_0^1 f(x)\log_2 f(x)dx \\ &= -\int_0^1 1\log_2 1dx \\ &= 0 \end{aligned} \tag{4.17}$$

We can prove the upper bound mathematically; consider two density functions $f(x)$ and $g(x)$, both defined on $x \in [0,1]$, we have:

$$\begin{aligned} \int_0^1 f(x)\log_2\frac{g(x)}{f(x)}dx &= \frac{1}{\ln 2}\int_0^1 f(x)\ln\frac{g(x)}{f(x)}dx \\ &\leq \frac{1}{\ln 2}\int_0^1 f(x)(\frac{g(x)}{f(x)}-1)dx \qquad (\because \ln X \leq X-1 \quad \forall X \geq 0) \\ &= \frac{1}{\ln 2}\int_0^1 (g(x)-f(x))dx \\ &= \frac{1}{\ln 2}(\int_0^1 g(x)dx - \int_0^1 f(x)dx) = 0 \\ \therefore \int_0^1 f(x)\log_2\frac{g(x)}{f(x)}dx &\leq 0 \end{aligned} \tag{4.18}$$

where the equality holds only if $f(x) = g(x)\ \forall x \in [0,1]$. Next, we expand:

$$\begin{aligned} \int_0^1 f(x)\log_2\frac{g(x)}{f(x)}dx &\leq 0 \\ \Rightarrow \int_0^1 f(x)\log_2\frac{1}{f(x)}dx + \int_0^1 f(x)\log_2 g(x)dx &\leq 0 \\ \Rightarrow -\int_0^1 f(x)\log_2 f(x)dx &\leq -\int_0^1 f(x)\log_2 g(x)dx \\ (\text{by putting } g(x)=1) \Rightarrow -\int_0^1 f(x)\log_2 f(x)dx &\leq -\int_0^1 f(x)\log_2 1dx \\ \Rightarrow -\int_0^1 f(x)\log_2 f(x)dx &\leq 0 \\ \therefore \text{maximum } CE &= 0 \end{aligned} \tag{4.19}$$

Thus, CE is always less than or equal to 0. The situation occurs if the normalized error function is a *white spectrum*. Note that the putting of $g(x) =$

1 in (4.19) is the only way to get a constant upper bound (not as a function of $f(x)$). The white spectrum is shown in **Fig. 4.10**, which is an extreme case of a untidy, peaked error function such that there is a "peak" on every point.

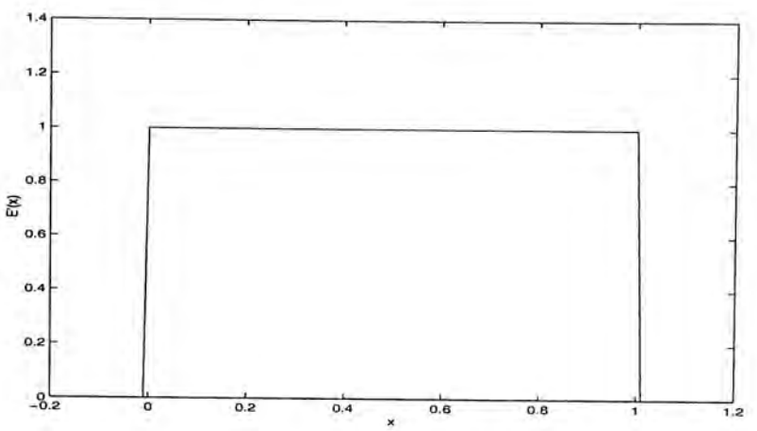


Figure 4.10: A white spectrum returns the maximum CE

Lower Bound

Theoretically, the lower bound of CE occurs if and only if $f(x) = \delta(x - x_0)$, where δ is the Dirac delta function (or the unit impulse function) such that $\int_{-\infty}^{\infty} \delta(x - x_0)dx = 1$ and $\delta(x) = 0\ \forall x \neq 0$, and x_0 is some constant in $[0, 1]$:

$$\begin{aligned} CE &= -\int_0^1 f(x) \log_2 f(x) dx \\ &= -\int_0^1 \delta(x - x_0) \log_2 \delta(x - x_0) dx \\ &= -\log_2 \delta(x_0 - x_0) \\ &= -\log_2 \delta(0) \\ &= -\infty \end{aligned} \tag{4.20}$$

However, the situation is different in real numerical computation. In practice, we cannot define an impulse function, which has infinite height and unlimited small width. Also, we the trapezoidal summation method to implement the integration. So, the impulse function is actually a zero function with a

particular point of finite height. Suppose there are $N+1$ samples located uniformly over $[0,1]$, such that there are N segments, each with width $1/N$. The numerical impulse function is plotted as a triangle as in **Fig. 4.11**. We can that the numerical lower bound of CE is:

$$\begin{aligned} -\int_0^1 f(x)\log_2 f(x)dx &= -\frac{1}{2N}\sum_{i=0}^{N-1}(f(\frac{i}{N})\log_2 f(\frac{i}{N}) + f(\frac{i+1}{N})\log_2 f(\frac{i+1}{N})) \\ &= -\frac{1}{2N}(0 + f(x_0)\log_2 f(x_0)) - \frac{1}{2N}(f(x_0)\log_2 f(x_0) + 0) \\ &= -\frac{1}{N}N\log_2 N \quad (\because f(x_0) = N) \\ \therefore \text{minimum } CE &= -\log_2 N \end{aligned} \tag{4.21}$$

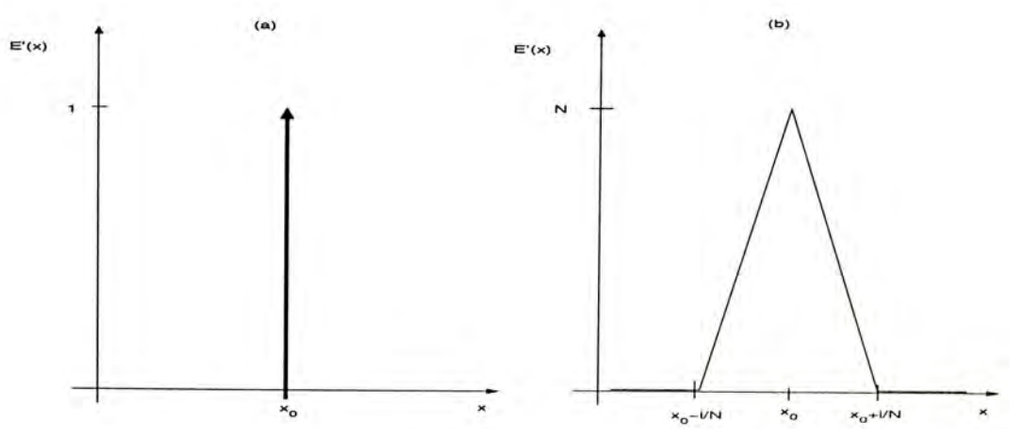


Figure 4.11: (a) The theoretical impulse function (b) The numerical impulse function

Our system limits the lower bound of CE to be -20, i.e. we support a maximum number of samples up to $N = 2^{20} = 10^5$. The limit of N is sufficient for most practical applications.

Remember that now we treat the normalized error function as if it is a density function. In probability theory, an impulse density function gives us the *most* information (the smallest CE), as we know there is only one point containing the signal. On the other hand, a white spectrum density gives us the *least* information (the largest CE) because every signal has the same probability. When we apply the continuous information theory in FF99, a

tidy and neat normalized error function is likely to produce a small CE while a messy one is likely to produce a large CE.

4.4.4 The Whole Heuristics of FF99

For literals in FF99, there are three scores: MSE, $CE_{\oplus}$ and $CE_{\ominus}$. We have to combine them to get a single heuristics for the final decision of literal selection. We know that a perfect literal gives $MSE = 0$ and an improper literal will give MSE tends to 1. This implies a *cost* heuristics in the range $[0, 1]$. In order to implement the CE scores easily, we would map the range of CE from $[-20, 0]$ to $[0, 1]$ also. The formula of the mapping is as:

$$CE' = 1 - \frac{\log_2(1 - CE)}{\log_2(20 + 1)} \tag{4.22}$$

where the mapping is done in a non-linear fashion in order to suppress the difference in CE of an impulse-like error function and that of a messy error function.

Then, the three scores are combined linearly to get a single cost in FF99:

$$COST = \frac{MSE + CE'_{\oplus} + CE'_{\ominus}}{3} \tag{4.23}$$

where the final $COST$ has the range in $[0, 1]$.

Remember that if the positive (negative) error function equals the constant 0, we cannot get the normalized positive (negative) error function, and in turn we cannot evaluate $CE_{\oplus}$ ($CE_{\ominus}$). In that cases, we would have special treatments, i.e. we would directly set $CE'_{\oplus} = 0$ ($CE'_{\ominus} = 0$) to skip the evaluation of CE. The setting of $CE' = 0$ implies that it is the most ideal situation, which there is no error at all. The whole literal selection process is summarized in the block diagram in **Fig. 4.12**.

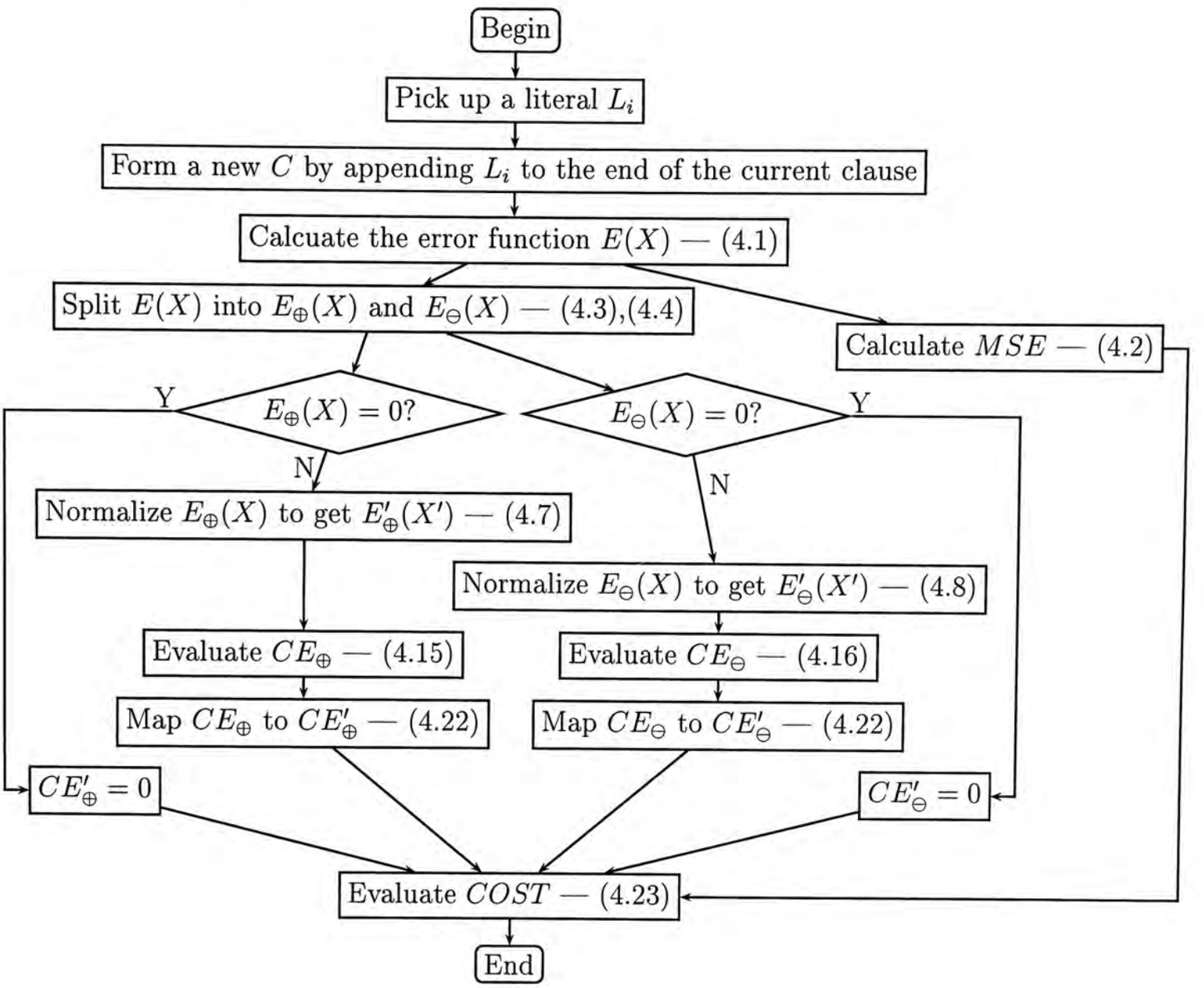


Figure 4.12: Literal selection in FF99

4.5 An Example

In this section, we are going to use an example to illustrate the effectiveness of the heuristics in FF99. In particular, we test the system by 10 literals $L_1, L_2, \ldots, L_{10}$ and we artificially construct the target relation T as:

$$T \ :- \ L_1, L_2, L_3$$

$$T \ :- \ L_4, L_5$$

We start the learning from a null concept description C. Following the

procedures in **Fig. 4.12**, we pick up each literal L_i, construct a new C and feed it to evaluate the corresponding $COST$. The results are shown in **Table 4.2**. The minimum MSE, $CE'_{\oplus}$ and $CE'_{\ominus}$ are highlighted with **bold** fonts.

C :–

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
MSE	.120	**.102**	.145	.127	.208	.142	.323	.450	.264	.509
$CE'_{\oplus}$	.602	.602	.602	.646	.646	.681	.751	**0**	.738	.420
$CE'_{\ominus}$	.634	.651	.628	**.544**	.580	.587	.594	.896	.549	.898
$COST$	.452	.452	.459	**.439**	.478	.470	.556	.449	.517	.609

Table 4.2: 1st iteration

L_4 gets the least $COST$, so it is appended to C. In the 2nd iteration, we exclude L_4 since it exists in the current clause already.

C :– L_4

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
MSE	.177	.177	.177	/	**.112**	.139	.177	.127	.177	.127
$CE'_{\oplus}$	.751	.751	.751	/	**.646**	.681	.751	**.646**	.751	**.646**
$CE'_{\ominus}$	**0**	**0**	**0**	/	**0**	.536	**0**	.544	**0**	.544
$COST$	.309	.309	.309	/	**.253**	.452	.309	.439	.309	.439

Table 4.3: 2nd iteration

This time, L_5 is chosen. Following the nodal characteristics in **Section 4.3.2**, we see that L_5 gets the least $COST$, at the same time, it gets $CE'_{\ominus} = 0$, which implies that the learning of the current clause should be terminated and we should start a new clause.

C :– L_4, L_5

C :–

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
MSE	.055	**.038**	.080	.127	.208	.127	.258	.450	.199	.509
$CE'_{\oplus}$	**0**	**0**	**0**	.646	.646	.646	.646	**0**	.630	.420
$CE'_{\ominus}$	.634	.651	.628	**.544**	.580	.587	.594	.896	.549	.898
$COST$	.230	**.229**	.236	.439	.478	.454	.499	.449	.459	.609

Table 4.4: 3rd iteration

In the 3rd iteration in **Table 4.4**, we see that L_2 results in the least $COST$, also with a zero $CE'_{\oplus}$ and a non-zero $CE'_{\ominus}$. According the nodal characteristics, it implies that the current clause should be the final clause to learn and it still need some literals to be completed. A similar situation occurs in the 4th iteration in **Table 4.5**. This time L_3 is selected.

C :– L_4, L_5

C :– L_2

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
MSE	.016	/	**.011**	.112	.112	.112	.112	.038	.099	.039
$CE'_{\oplus}$	**0**	/	**0**	.646	.646	.646	.646	**0**	.630	.420
$CE'_{\ominus}$	.606	/	.479	**0**	**0**	**0**	**0**	.651	.248	.640
$COST$	.207	/	**.164**	.253	.253	.253	.253	.229	.326	.336

Table 4.5: 4th iteration

C :– L_4, L_5

C :– L_2, L_3

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
MSE	**0**	/	/	.112	.112	.112	.112	.011	.099	.015
$CE'_{\oplus}$	**0**	/	/	.646	.646	.646	.646	**0**	.630	.420
$CE'_{\ominus}$	**0**	/	/	**0**	**0**	**0**	**0**	.479	**0**	.479
$COST$	**0**	/	/	.253	.253	.253	.253	.164	.243	.305

Table 4.6: 5th iteration

In the 5th iteration, we finally achieve the situation $MSE = CE'_{\oplus} = CE'_{\ominus} = 0$, which implies a perfect match and the learning of the whole concept description is terminated. The result is the following C, which is equivalent to T.

$$C \ :- \ L_4, L_5$$
$$C \ :- \ L_2, L_3, L_1$$

One may justify the usefulness of this example as there are only 10 literals to select and 5 of them are involved in T. However, this example actually shows that the heuristics is very effective in constructing the correct C through a *greedy search*. Remember there is no backtracking in a greedy search, and the $COST$ heuristics is to ultimate guide to search for the correct literals. In the sense of probability, if we were constructing C by a *random search*, the probability of constructing a correct C (which is equivalent to T) is:

$$\begin{aligned} \text{Probability(correct } C) &= \frac{2}{10} \times \frac{1}{9} \times \frac{3}{10} \times \frac{2}{9} \times \frac{1}{8} \times 2 \\ &= 0.00037 \end{aligned}$$

which is small enough to show that the correct construction of C is not trivial.

Note that throughout the construction of C, none of the scores: MSE, $CE'_{\oplus}$ or $CE'_{\ominus}$ could singly guide the search. In most of the cases, the "relevant" literals would give smaller MSE in comparing to those "irrelevant" literals.

However, we see that the difference of MSE may be too small and it is too risky to select a literal merely by its MSE. By the complementation of CE's, it is more safety to have a complete $COST$ heuristics and the reasoning was detailed in **Section 4.3** already. This example also demonstrated that the nodal characteristics take effects even when T is a disjunction of conjunction of literals (in the illustration of the nodal characteristics in **Section 4.3.2**, T is either a single conjunction or a single disjunction).

For simplicity, the example is written in a separate MATLAB script in **Appendix B**. It runs on MATLAB Version 5.3.1 on a 128MB Sun Ultra 5/270 machine. The sample size is 10000, average running time is 11.2 seconds (average of 10 executions), including the plotting and saving of data.

In conclusion, this chapter discussed a novel method in comparing two fuzzy sets. This method is shown to be effective in guiding a greedy search of the concept description.

Chapter 5

Partial Evaluation of Literals

This chapter describes the extension of the traditional covering method which is aimed at inducing concepts both in Boolean and fuzzy domains. First, we would review and analyze the importance of the original covering method in inductive learning systems. Then, the difficulties in applying this method for inducing fuzzy concepts are presented. In order to overcome the difficulties, we redefine the covering concept in a sense of set membership agreements. The following sections describes in detail the steps and calculations.

5.1 Importance of Covering in Inductive Learning

There has been considerable research on the task of ***inductive learning*** : given a training set of objects whose classes are known, find a presentation for predicting the class of an unseen object as a function of its attribute values. Many of those learning algorithms use one the two approaches, namely the ***divide-and-conquer*** method and the ***covering*** method.

5.1.1 The Divide-and-conquer Method

This method is mainly applied in system which represents the concept learned in form of *decision tree.* Classical examples include Hunt's CLS [38] and Quinlan's ID3 [39]. This method is summarized as follows:

- If all objects in the training set in the current node belong to a single class, this node is a leaf labeled with that class.
- Else,
  - select a literal to test the training set (based on some heuristics);

- *divide* the training set into subsets, each corresponding to one of the mutually exclusive outcomes of the test, and hence create a node (the son of the current node) for each of the subsets;
- recursively apply the procedures to each node until it is *conquered*, i.e. all objects in the training set are labeled by the descending leaves of the current node.

To demonstrate the divide-and-conquer method and the decision tree representation, consider a simple Boolean domain in Table 5.1 which consists of twelve objects only. The decision tree is shown in **Fig. 5.1**.

Universe	$X = \{1, 2, \ldots, 12\}$
Target Concept	$T \subset X = \{3, 4, 5, 6, 7, 9, 10, 11, 12\}$
Literal 1	$L_1 \subset X = \{1, 2, 3, 4, 5, 6, 7\}$
Literal 2	$L_2 \subset X = \{3, 4, 5, 6, 7, 8, 9\}$
Literal 3	$L_3 \subset X = \{9, 10, 11, 12\}$

Table 5.1: Data for demonstrating inductive learning

5.1.2 The Covering Method

For other induction algorithms, that represent classification knowledge as a disjunction of conjunctions of conditions, employ the covering method. Notable examples of those algorithms are the AQ family from Michalski [13] and FOIL from Quinlan [11]. This method is summerized as follows:

- If all objects in the training set in the current conjunction belong to a single class, this conjunction is completed. Then *remove* all objects that satisfy this conjunction and start a new conjunction.
- Else

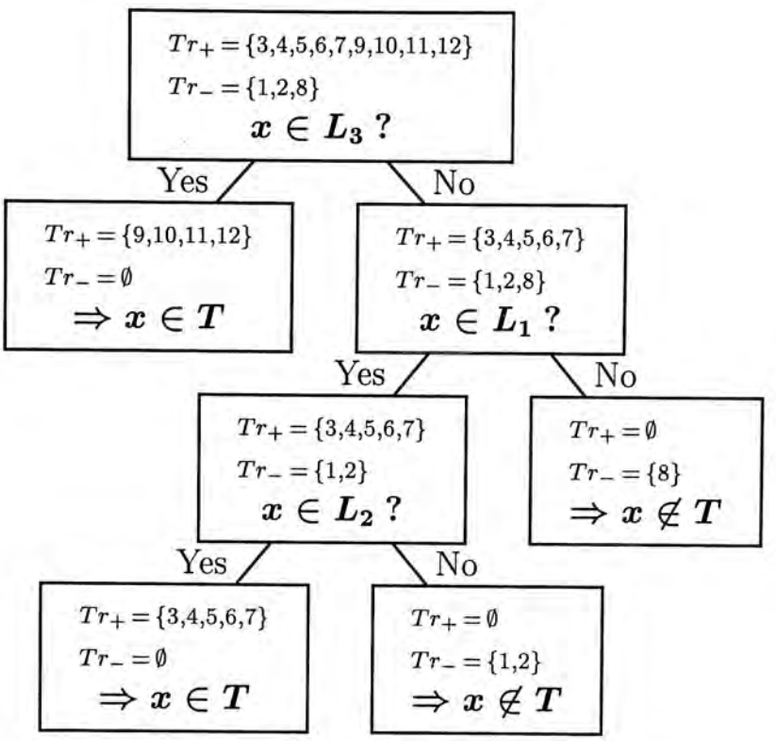


Figure 5.1: The divide-and-conquer method

  - select a literal and append it to the current conjunction (based on some heuristics);
  - *prune* those objects in the current training set which do not satisfy the newly-appended literal;

- Repeat the procedures until all objects are *covered*, i.e. they are satisfied by at least one of the conjunctions.

Again, we use the data in Table 5.1 to demonstrate how the covering method works, the result is shown in **Table 5.2**.

5.1.3 Effective Pruning in Both Methods

The main difference between the two methods is the way they represent the inducted concept. In particular, the divide-and-conquer method naturally induces a decision tree, while the covering method induces a set of rules or

$x \in T$	IF	$x \in L_1$	AND	$x \in L_2$
init. step		step 1.		step 2.
$Tr = X_+ \cup X_-$		$Tr = X_+ \cup X_-$		$Tr = X_+ \cup \emptyset$
$X_+ = \{3,4,5,6,7,9,10,11,12\}$		$X_+ = \{3,4,5,6,7\}$		$X_+ = \{3,4,5,6,7\}$
$X_- = \{1,2,8\}$		$X_- = \{1,2\}$		
	OR			
$x \in T$	IF	$x \in L_3$		
step 3:		step 4.		
{3,4,5,6,7} are covered already				
$Tr = X_+ \cup X_-$		$Tr = X_+ \cup \emptyset$		
$X_+ = \{9,10,11,12\}$		$X_+ = \{9,10,11,12\}$		
$X_- = \{1,2,8\}$				

Table 5.2: The covering method

Horn-clauses. Although the steps for converting between a decision tree and a rule-set are well-published (e.g. Quinlan's C4.5 [30]), it is just obvious to employ the covering method to induce rule-based concepts, or vice versa. So, our system, which induces a set of fuzzy Horn-clauses, is going to apply the concept of covering method.

However, both methods share the same core idea : it keeps on *pruning* the training set in successive induction steps. The divide-and-conquer keeps on *dividing* the current training set, while the covering method keeps on *removing* the objects that is not covered. This idea is obviously important if we want to induce concepts from a large dataset. Moreover, an efficient pruning method is crucial in *first-order* concept learning, because the main problem of most first-order systems is the curse of dimensionality. Notably, FOIL, which is the first efficient first-order learning algorithm, successfully apply the covering method to prune the multi-dimensional training set.

5.2 Fuzzification of FOIL

After reviewing the importance of pruning in inductive learning, we now discuss the possibilities in incorporating fuzzy ideas into the pruning methods. This section gives the reason why we want to fuzzify FOIL [11]. It also includes what we have to consider, what we want to achieve and what we have to sacrifice in the system fuzzification process.

5.2.1 Analysis of FOIL

We have discussed the development of FOIL in **Section 2.3.1**, now we go further to analyze the success and uniqueness of it. The analysis begins with the sketch of the algorithm as follows:

1. Initialize a null clause as "$C : -$"
   (a) $\oplus_C \leftarrow$ positive tuples in the target relation
   $\ominus_C \leftarrow$ negative tuples in the target relation
   (b) Calculate information as $I_C = -\log_2(\frac{|\oplus_C|}{|\oplus_C|+|\ominus_C|})$ where $|\odot|$ is the number of tuples in $\odot$
2. For each literal L
   (a) Append L to the right-hand side of C to form C'
   (b) $\oplus_L \leftarrow$ positive tuples of L
   (c) $\forall \quad \oplus_L \in \oplus_C \rightarrow \oplus_{C'}$
   (d) $\forall \quad \oplus_L \notin \oplus_C \rightarrow \ominus_{C'}$
   (e) Count N as the number of $\oplus'_C$ account for unique tuple in $\oplus_C$
   (f) Calculate information as $I_{C'} = -\log_2(\frac{|\oplus_{C'}|}{|\oplus_{C'}|+|\ominus_{C'}|})$
   (g) Calculate gain as $G = N \times (I_C - I_{C'})$

3. Select the literal L with the greatest gain G

   (a) If the greatest $G > 0$

   - Update C by C'
   - If the updated C contains no negative tuple, goto **Step 4**
   - Goto **Step 2**

   (b) Else (greatest $G \leq 0$)

   - If C is a null clause, terminate the whole learning
   - Goto **Step 4**

4. Complete the learning of the current clause

   (a) Start learning from a null clause as in **Step 1**

   (b) However, the $\oplus_C$ set is pruned by the $\oplus$ tuples in all previously completed clauses

   (c) If $\oplus_C = \emptyset$, terminate the whole learning

   (d) Goto **Step 2**

FOIL is proven to be very *efficient* and *accurate* in learning Boolean first-order concepts. By analyzing the above algorithm, we found that the success of FOIL comes from the following reasons:

Efficiency The covering technique is highly embedded in the algorithm. In particular, the specification process in **Step 2c** keeps on reducing the size of (unique) positive tuples throughout the induction of the current clause. We can see that for each literal L, we need only to consider its positive tuples, namely $\oplus_L$, in the generation of $\oplus_{C'}$ from $\oplus_C$. It results in $|\oplus_{C'}| \leq |\oplus_C|$, if they are in the same variable orderings. **Step 4b**, on the other hand, keeps on removing the covering positive tuples in successive clauses. These covering techniques provide the foundation of a successful first-order induction algorithm; as it will be shown by examples, the

number of tuples grows in an exponential manner when new variables are introduced. Without such pruning, the first-order induction task is by nature an intractable problem; it could not be solved (in reasonable time) even we consider relations among several tens objects only.

Accuracy The information estimation in **Step 2f** is a good heuristics for literal selection, in the sense that it could be applied in a wide variety of literals. As long as we can classify the training samples into $\oplus$ and $\ominus$ sets, the information heuristics is applicable, no matter what arity is the literal, no matter what types are the variables in. Since Quinlan's ID3 [39], these information gain settings have long been applied in many induction systems which produces high classification rates. Of course, computer sciencist like to use it because it gets a solid theoretical foundation from Shannon's information theory [36]. Also, many arithmetic operations are allowed on it. Remember that an information is the negative logarithmic of a probability, however, it has no semantics of "dividing a probability from another" or "a negative probability". The information setting allows more arithmetic operations, such as multiplication, division, etc., and it gives more flexibility to design the heuristics.

Integrity Efficiency and accuracy do not guarantee a successful system, unless both of them could be integrated seamlessly. The beauty of FOIL is that the system uses a *partial evaluation* method (the information gain heuristics) during a pruning process. In contrast to FF99, which always evaluates a literal globally (all tuples needed to be considered), FOIL evaluates some *selected* tuples from a literal. More importantly, once we select the tuples, we need not to go back and consider the unselected tuples again. The integrity of the efficient pruning techniques and the accurate evaluation method emerges the best first-order induction system in literature.

5.2.2 Requirements on System Fuzzification

Before we explore the ways to fuzzify FOIL, we step back to review two requirements to fulfill in fuzzifying any Boolean system, namely:

Compatibility The fuzzified system should behave in exactly the same way as the Boolean system if all the degree of memberships are 1 or 0. Usually, it is done by replacing the Boolean set operations (e.g. union and intersection) by t-norm operations (as discussed in **Section 2.4.3**). Functions, on the other hand, are quite different for Boolean sets and fuzzy sets. The functions involving Boolean sets usually works on the *size* of positive or negative samples; however, the functions for fuzzy sets operates on the degree of memberships, not the SIZE. If a fuzzified system fulfills this compatibility criterion, at least it would guarantee correct results if binary data are fed in.

Continuity Without the loss of generality, the continuity criterion is formulated as: For any function $f : x \rightarrow a$, where a is a real scalar and x is a fuzzy number with μ_x being its degree of membership, if $f(x_1) > f(x_2)$ $(f(x_1) < f(x_2))$ for some $\mu_{x_1} > \mu_{x_2}$ $(\mu_{x_1} < \mu_{x_2})$, then $f(x_i) > f(x_j)$ $(f(x_i) <> f(x_j))$ for *all* $\mu_{x_i} < \mu_{x_j}$ $(\mu_{x_i} < \mu_{x_j})$. The simplest valid example is $f(x) = \mu_x$. This criterion is simple, nevertheless, it preserves one important property of fuzzy set : the degree of belonging is continuously spread with *no discrete boundaries.* It also guarantees that samples of different degree of memberships yield different result, i.e. $f(x_a) \neq f(x_b) \forall \mu_{x_a} \neq \mu_{x_b}$. Many system fuzzification methods fail in this criterion when they try to trim several Boolean sets off the fuzzy set. By that way, $f(x_a) = f(x_b)$ even if $\mu_{x_a} \neq \mu_{x_b}$ but x_a and x_b are put into the same Boolean set.

Note that the former requirement is necessary while the latter is not, i.e., many fuzzified systems (a fuzzified system refers to a Boolean system that is

extended to allow fuzziness) do not fulfill the continuity condition. FAQR [18], which partitions the training samples into "soft positive set" and "soft negative set"; disregards the differences of class memberships among the samples in the same partition. Put it simple, any algorithm that partitions the samples is violating the continuity requirement to some extend, although the construction of partitions usually depends on the class memberships. On the other hand, the fuzzy C-means algorithm (FCM) [40] does satisfy both requirements; the class memberships are always involved in each iteration and a little change in one class membership distorts the original results.

As discussed in **Section 5.2.1**, the success of FOIL is nothing fortuitous. On top of that, we go further to specify the goals we *want* to achieve in fuzzifying FOIL.

Knowledge Representation Our system should provide more possible ways in representing knowledge, in comparison with FOIL, through the introduction of new literal forms. Of course, those "advanced" literal forms should make advantages of fuzzy logic — to avoid crisp classification boundaries and to allow the use of linguistic terms. The details of those literal forms are already discussed in **Section 3.1**.

Speed The problem of first-order induction is nothing special but suffering serious speed consideration. In fact, only two systems in literature, namely FOIL and GOLEM [12], are capable to induce first-order concepts in reasonable time. With the introduction of fuzzy relations, the original settings of FOIL or GOLEM are no longer valid, and a little change in the original algorithm may lead to a big difference in running time. Therefore, one should seriously monitor the speed of the modified system.

5.2.3 Possible Ways in Fuzzifing FOIL

Initially, we try to develop FF99 (please refer to **Chapter 4**), which is a completely new system, to induce fuzzy first-order concepts. Instead of applying the discrete information theory as in FOIL, FF99 makes use of the continuous one. Unfortunately, we failed somehow: the heuristics described in **Fig. 4.12** implicitly requires all the variables being defined on ordered universe, i.e., FF99 could not handle nominal data, which is very common in relational database. Also, it is a GLOBAL evaluation method — all samples have to be considered all the way through the algorithm — that is not acceptable in first-order learning. It leads us to step back and try to fuzzify FOIL instead of developing a wholly new system, though FF99 proves to be a novel method in zeroth-order fuzzy induction.

In the first sight, we tried to enhance the information measure for Boolean set to allow fuzziness. The Boolean information measure for a clause, I_C, is the negative logarithmic of the proportion of the number of positive tuples to the number of all tuples:

$$I_C = -\log_2(\frac{|\oplus|}{N}) \tag{5.1}$$

Note that this measure implicitly partitions the samples into two crisp sets : the positive set ($\oplus$) and the negative set ($\ominus$). If we want to follow the continuity criterion stated in **Section 5.2.2**, we have to explore a measure which does not require any partitioning. The extension, I'_C, replaces the partition size by the summation of class memberships:

$$I'_C = -\log_2(\frac{\sum_i \mu(i)}{N}) \tag{5.2}$$

where $\mu(i)$ is the class membership for the i-th sample and N is the total number of samples in the clause.

The case is more complex for the information measure of literal selection. Refer to **Step 2** in the FOIL algorithm discussed in **Section 5.2.1**, one need

to select the *positive tuples* from a literal and perform a set intersection with the *positive tuples* of the clause. Those sets of positive tuples, $\oplus_L$ and $\oplus_C$, no longer exist in this extension because we are avoiding any crisp partitioning. Instead of performing crisp set intersection, we express the information of a particular literal L by the multiplication of class memberships:

$$I'_{C'} = -\log_2\left(\frac{\sum_i \mu_L(i) \times \mu_C(i)}{\sum_i \mu_L(i)}\right) \tag{5.3}$$

where $\mu_L(i)$ and $\mu_C(i)$ are the class membership of the i-th sample of the literal and the current clause respectively.

Both extensions ((5.2) and (5.3)) perfectly fulfill the compatibility and continuity issues as discussed in **Section 5.2.2**. First, the limits of I_C and I'_C agree : a purely positive set (all μ's = 1) gives $I_C = I'_C = 0$; a purely negative set (all μ's = 0) gives $I_C = I'_C = \infty$. Also, by designing a function $f(I') = -I'$, we see that $f_1 > f_2$ if $\mu_1(a) > \mu_2(a)$, $\mu_1(i) = \mu_2(i) \quad \forall \quad i \neq a$. It implies every change in one class membership corresponds to a change in the extended information measure, i.e., all class memberships are responsible for the final results.

The replacement of crisp set operations by class membership arithmetics is not a fresh idea. Several systems, e.g. FUZZY VERSION SPACE [41], applied this idea and successfully induce fuzzy concepts on top of the classic VERSION SPACE algorithm. Yet, all these systems are zeroth-order learner. We can see the reason behind from (5.3) which "select" the "positive" samples by class membership multiplication — it implies a big problem : no pruning ! Remember that in the original information measure, we need not to consider the samples in the negative set, and the number of (unique) positive samples is getting smaller in successive learning. Finally, we found that the extension of Boolean information measure is, although correct theoretically, *not practical* for first-order induction.

We soon conclude that the continuity criterion contradicts the speed consideration. A trade-off always stays between them : the more you consider every bit of class membership, the more samples you have to process; the more you partition and prune the samples, the more class memberships you omit. We made the choice: speed. So, our system would maintain the same partition and pruning mechanism in FOIL. At the same time, the construction of partition should be sensitive to little changes in class memberships. The two proposed methods, namely α-covering and probabilistic-covering, differ in the way they construct the $\oplus$ and $\ominus$ partitions. The flow of FOIL are kept intact in these two algorithms, which will be discussed in the coming sections, hence guarantees the speed in first-order learning for both fuzzy and Boolean concepts.

5.3 The α Covering Method

For Boolean systems, we could simply define a "positive" sample as the target tuple that has degree of membership of one, where a "negative" sample as the sample having zero degree of membership. However, the situation is more complex in fuzzy system as a class membership lying between zero and one is allowed. Imagine a sample with $\mu = 0.5$, can we immediately classified it as "positive" or "negative"? The answer is no. Recall that a sample is "covered" means that both the target and the training sample are positive, i.e. $\mu_T = 1, \mu_C = 1$; where a sample is "uncovered" means that it has a positive target sample yet a negative training sample, i.e. $\mu_T = 1, \mu_C = 0$. So that, we first have to classify a fuzzy sample as "positive" or "negative" before we can determine the learning be "covered" or not.

5.3.1 Construction of Partitions by α-cut

The simplest solution to the above problem is to set a pre-defined threshold α to classify each sample. That is, the (target or training) samples are partitioned into two crisp sets : the "positive" ($\oplus$) and the "negative" ($\ominus$) sets:

$$\forall x \in X, \qquad \mu(x) > \alpha \Rightarrow x \in \oplus, \qquad \mu(x) \leq \alpha \Rightarrow x \in \ominus \tag{5.4}$$

or it can be rewritten into the form of α-level set as in (2.13):

$$\oplus = X_{\alpha+}, \qquad \ominus = X_{\alpha} \tag{5.5}$$

This method has been applied in many fuzzy inductive systems, for example, FILSMR [19] sets $\alpha = 0.5$ to learn modular rules. A sample is said to be ***α-covered*** if both the target and the training samples are positive (with respect to α). Consider the illustration in **Table 5.3**, we could pick out the α-covered samples very quickly.

μ_T	0.0	0.3	0.4	0.5	0.7	0.7	0.9	0.9	1.0
μ_C	0.0	0.1	0.1	0.5	0.4	0.6	0.1	0.7	1.0
α-covered ?				$\surd$		$\surd$		$\surd$	$\surd$

Table 5.3: α-covering with $\alpha = 0.5$

Following the application of α-partitioning method, we could then apply the FOIL algorithm as usual, which guarantees a fast induction. Of course, once the fuzzy set is partitioned into the crisp $\oplus$ and $\ominus$ sets, the class memberships are omitted. The high induction speed is achieved at the expense of the sensitivity of class memberships.

5.3.2 Adaptive-α Covering

The original α-covering method takes several advantages: it is simple, fast and compatible to Boolean logic. However, a fixed setting of α may not be the

optimal choice for all situations. As our system does not restrict the way how is the data collected, the data of a fuzzy relation may be collected through some questionnaires. In that situation, the fuzzy relation may be *heavily biased*, say, most of the samples have zero memberships. So, our system offers a better way to determine α in run-time: we use a K-means clustering algorithm to classify the samples into the "positive" or the "negative" set. We set the initial class centers to be $c_1 = 0.25$ and $c_2 = 0.75$, and after running the K-means iterations, we would get a optimal set of $\dot{c_1}$ and $\dot{c_2}$ to "separate" the samples. Finally the adaptive α is determined by:

$$\alpha = \frac{\dot{c_1} + \dot{c_2}}{2} \tag{5.6}$$

Note that α is determined separately for the target samples and the training samples, so we would get α_T and α_C. Following the previous example in **Table 5.3**, we work out $\alpha_T = 0.56$ and $\alpha_C = 0.39$ by the K-means algorithm. A new covering result is shown in **Table 5.4**. We see that μ_T is biased towards 1.0, μ_C is biased towards 0.0; in that way 0.5 is relatively not large enough for μ_T to be positive and 0.4 is relatively large enough for μ_C to be positive. The difference of the original α-covering and our adaptive α-covering is shown in the fourth and fifth pairs. We would see that the result of adaptive α-covering is more robust and as its name, more adaptive.

μ_T	0.0	0.3	0.4	0.5	0.7	0.7	0.9	0.9	1.0
μ_C	0.0	0.1	0.1	0.5	0.4	0.6	0.1	0.7	1.0
α-covered ?					√	√		√	√

Table 5.4: Adaptive α-covering with $\alpha_T = 0.56$, $\alpha_C = 0.39$

Table 5.3 and **Table 5.4** show one vital conceptual difference between the two methods, though only the fourth and fifth pairs changed. In fact, the fixed-α-covering method makes no difference in giving a Boolean dataset between specifying a fuzzy dataset. We can see that a sample with $\mu = 0.1$ or

$\mu = 0.4$ performs just the same as a Boolean negative sample (provided that the fixed-$\alpha = 0.5$). On the contrary, the class membership of each sample is effective in adjusting the adaptive-α. Adding adaptiveness in the simple fixed-α method makes it becomes a "nearly real fuzzy system" (we would elaborate on this point in next section). Obviously, it's still problematic to partition the samples merely by their class memberships, we would try to cope with this problem in next section.

After the data is read from input, we perform the α-cut partitioning which results in two crisp sets ($\oplus$ and $\ominus$) for each relation. In case of adaptive α-cutting, the "optimal" α is estimated for each *literal form* after all literals forms are generated (please refer to **Section 3.2.2**).

The user has a choice between the α covering and the adaptive-α covering methods. The former one could be chosen by specifying a particular α, or the latter method will be used by default.

5.4 The Probabistic Covering Method

The fixed-α covering provides the simplest means to partition the fuzzy samples into the crisp $\oplus$ and $\ominus$ sets, in order to adapt the efficient FOIL main flow; the adaptive-α method makes one step forward in relating the partition boundaries with the class memberships. Notwithstanding, they both suffer one serious problem *conceptually*: they tend to ignore the cases of low class memberships. One may argue that we *should ignore* those cases without questions, however, consider a strange case as follows: given 3 literals (L_0, L_1, L_2), learn the concept description for L_0. The correct answer should be:

$$\begin{aligned} L_0(X) &:- L_1(X) \\ L_0(X) &:- L_2(X) \end{aligned} \tag{5.7}$$

This problem seems to be too simple to allow any fault, nevertheless, consider the special arrangement in **Table 5.5**, surprisingly, the α-covering methods could *never* learn the perfect description as in (5.7).

i	1	2	3	4	5	6	7	8	9	10	11
$\mu_{L_0}(i)$	0.9	0.8	0.7	0.6	0.6	0.3	0.3	0.2	0.2	0.1	0.0
$\mu_{L_1}(i)$	0.9	0.8	0.7	0.6	0.6	0.0	0.0	0.0	0.0	0.0	0.0
$\mu_{L_2}(i)$	0.0	0.0	0.0	0.0	0.0	0.3	0.3	0.2	0.2	0.1	0.0

Table 5.5: A case to demonstrate the inadequacy of α-covering

We can see that L_1 perfectly describes the higher class membership samples ($i = 1 \rightarrow 5$), while L_2 is in charge of the lower membership cases ($i = 6 \rightarrow 11$); the disjunction of these two literals is exactly equal to L_0. We try to formulate the observations in the α-covering methods stated in **Section 5.3**, we choose the fixed-α to be 0.5 and the adaptive-α's are estimated as $\alpha_{L_0} = 0.46$, $\alpha_{L_1} = 0.37$ and $\alpha_{L_2} = 0.59$. The result is amazing : L_2 covers none of the target samples from whatever α covering methods !

L_1 : $i =$	1	2	3	4	5	6	7	8	9	10	11
fixed-α-covered ?	√	√	√	√	√						
adaptive-α-covered ?	√	√	√	√	√						
L_2 : $i =$	1	2	3	4	5	6	7	8	9	10	11
fixed-α-covered ?											
adaptive-α-covered ?											

Table 5.6: Use of α-covering methods

From the viewpoint of the first-order induction algorithm, L_0 and L_1 are equivalent as shown in**Table 5.6**. ; L_1 is considered as "perfectly covering" the target (as it has already "covered" all the "$\oplus$" samples in target), thus, the learning will be terminated as:

$$L_0(X) \quad :- \quad L_1(X) \tag{5.8}$$

Unfortunately, this imperfect result is likely to give high scores by using whatever partial evaluation method. Unless we use a global evaluation method, which compares the result and the target in a sample-by-sample manner, to discover the small amount of error, we don't know that (5.8) could still be improved. In spite of the mentioned deficiency, any global evaluation should be prohibited during the induction process (please refer to **Section 5.1.3** for reasons). All these concerns lead us to a probabilistic covering approach.

The probabilistics covering approach refers to partitioning samples into the $\oplus$ and $\ominus$ *by chance*. Of course, the probability should be dependent on the class memberships. The way we suggest is: the probability of a sample to be classified as "+ve" is equal to the square-root of its class memberships, i.e.

$$P(x \in \oplus) = \sqrt{\mu(x)} \tag{5.9}$$

$$P(x \in \ominus) = 1 - \sqrt{\mu(x)} \tag{5.10}$$

By this way, the probability of a sample x, which has class membership $\mu_C(x)$ on the concept description and $\mu_T(x)$ on the target, to be considered "covered" is:

$$P(x \text{ is covered}) = \sqrt{\mu_C(x)} \times \sqrt{\mu_T(x)} \tag{5.11}$$

There are some important properties behind (5.11) (here we denote $P(x$ is covered) by $P(\bullet)$) :

1. If either μ_C or μ_T equals 0, $P(\bullet)$ equals 0. It is consistent with Boolean cases.

2. If $\mu_C = \mu_T = 1$, $P(\bullet) = 1$. It is also consistent with Boolean cases.

3. If $\mu_C = \mu_T = \mu$, $P(\bullet) = \mu$. It means that any sample with non-zero class membership is *possibly* covering the target, and the possibility is linearly proportional to its class membership.

In implementation, a list of random number (in range of $[0, 1]$) is generated and squared. The list is considered as the probability in (5.10), and it is compared with the class memberships to get the $\oplus$ set. The following steps are as usual as Boolean cases.

i	1	2	3	4	5	6	7	8	9	10	11
$x_i \in \oplus_{\mathtt{L_0}}$?	√		√		√		√		√		
$x_i \in \oplus_{\mathtt{L_1}}$?	√	√	√		√						
$x_i \in \oplus_{\mathtt{L_2}}$?							√				

Table 5.7: A successful example of using the probabilistic covering method

Consider again the case which α-covering fails, the probabilistic covering approach provides the chance to get the perfect solution. One of the possibly is shown on **Table 5.7**: at least one of x_i, $i \in [6, 11]$ is classified as "+ve" in both $\mathtt{L_0}$ and $\mathtt{L_2}$, at the same time at least one of x_i, $i \in [1, 5]$ is "+ve" in $\mathtt{L_0}$ and $\mathtt{L_1}$. This kind of cases will force the system to consider both $\mathtt{L_1}$ and $\mathtt{L_2}$ in the clause, as the system tries to cover all the "positive" cases. In fact, the chance to get a perfect solution of this particular problem by the probabilistic approach is:

$$\begin{aligned} P(\text{perfect solution}) &= (1-(1-0.3)\times(1-0.3)\times(1-0.2)\times(1-0.2)\times(1-0.1)) \\ &\quad \times(1-(1-0.9)\times(1-0.8)\times(1-0.7)\times(1-0.6)\times(1-0.6)) \\ &= 0.718 \times 0.999 \\ &= 0.717 \end{aligned}$$

which shows that the probabilistic approach is very likely to get the perfect solution as in (5.7). But this appealing probability is not common because there is so little choices of literals in this example.

The probabilistic approach offers the chance to cover low class membership samples, at the time it may omit some high class membership samples. If some

"important" samples are omitted, the system cannot distinguish between the useful and the irrelevant literals, which results in long and incorrect solutions.

In practical usage, the α-covering approaches should be applied first. If the result is not satisfactory, carry out the probabilistic covering for several times to see if any better solution exists. The probabilistic approach shares the same belief of genetic algorithm : if the unique solution from a *deterministic* algorithm does not work, gets a pool of solutions and hopefully the optimal one is out there. We would understand more about the performance of these two covering approaches in **Chapter 6**.

Chapter 6

Results and Discussions

This chapter demonstrates FF01 on several types of learning tasks. The results are compared against existing learning algorithms. We will discuss the system at the end of this chapter.

6.1 Experimental Results

6.1.1 Iris Plant Database

Fisher created this well-known attribute-value dataset. The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant, namely setosa, versicolor and virginica. One class is linearly separable from the other two; the latter are not linearly separable from each other. Each instance is described by 4 numeric attributes, namely the sepal length(sl), sepal width(sw), petal length(pl) and petal width(pw). The dataset is originally designed for zeroth-order learning systems, however, with the help of the conversion utility **Appendix A**, it is able to formulate each class as a relation with arity of 4, and each attribute acts as a variable of the relation. We will compare the results get from FOIL and FF01 by doing 10 experiments, each with a 50–50 training–testing ratio.

First, we see the results of FOIL: 9 of the 10 experiments get the following results:

```
    setosa(sl,sw,pl,pw)  :-  pl ≤ 1.9
versicolor(sl,sw,pl,pw)  :-  pl > 1.9
 virginica(sl,sw,pl,pw)  :-  pw > 1.6
 virginica(sl,sw,pl,pw)  :-  pl > 4.9
```

The results of FOIL is similar, however, the expression is different:

```
setosa(sl,sw,pl,pw)     :- very_small(pl)(1.0,1.0,1.7,2.40)
versicolor(sl,sw,pl,pw) :- ¬very_small(pl)(1.0,1.0,1.7,2.4)
virginica(sl,sw,pl,pw)  :- very_big(pw)(1.4,1.6,2.5,2.5)
virginica(sl,sw,pl,pw)  :- very_big(pl)(4.5,5.1,6.9,6.9)
```

where the 4 numbers following the literals represent the parameters of the trapezoidal membership function as defined in (3.5).

Clearly, the results of FF01 is more readable and comprehensive than that of FOIL because we have no common sense on the exact value of some attributes. Do you know a sepal of length 1.9cm is long or not? It shows one great advantage of fuzzy expressions: it is closely related to human language.

Note that we are using the FF99 algorithm to learn this result, as all data are numeric, and the problem is zeroth-order. To analyze the classification accuracy, we compare it against C4.5, FOIL and another fuzzy learning system FAQR [18]. The results are shown in **Table 6.1**.

	C4.5	FOIL	FAQR	FF01
Setosa	100%	100%	100%	100%
Versicolor	95%	95%	97%	97%
Virginica	95%	95%	97%	97%

Table 6.1: Classification accuracy of iris classes

We find that the our system and FAQR achieve higher accuracy compared to C4.5 and FOIL. It demonstrates another main advantage of fuzzy learning systems: they prevent sharp classification boundaries, which are not avoidable in crisp learning systems. A fuzzy boundary always allows more robust classification, plus higher accuracy can be achieved.

6.1.2 Kinship Relational Domain

Introduction

Hinton developed this relational database, which consists of 24 unique names in two families (they have equivalent structures). Hinton used one unique output unit for each person and was interested in predicting the following relations: wife, husband, mother, father, daughter, son, sister, brother, aunt, uncle, niece, and nephew. Hinton used 104 input-output vector pairs (from a space of 12x24=288 possible pairs). The prediction task is as follows: given a name and a relation, have the outputs be on for only those individuals (among the 24) that satisfy the relation. The outputs for all other individuals should be off.

Clearly, using a neural network representation is not natural to define the kinship concepts. Quinlan repeat the learning task using FOIL. He gets this kind of Horn-clause results:

```
brother(X,Y) :-father(Z,X), son(Y,Z)
brother(X,Y) :-mother(Z,X), son(Y,Z)
```

where the predicate `P(A,B)` is read as "A is the P of B". Our system represents that concepts as the same way as FOIL does.

From the result comparison in [11], Hinton used 100 vectors as input and 4 for testing, his results on two passes yielded 7 correct responses out of 8. His network of 36 input units, 3 layers of hidden units, and 24 output units used 500 sweeps of the training set during training. While Quinlan repeated the experiment 20 times (rather than Hinton's 2 times). FOIL was correct 78 out of 80 times on the test cases. The capped **Table 6.2** summarize the classification results.

This summary seems to be too brief for us to test out our system, though we adapt the FOIL algorithm as the core flow in FF01. More than that, we

System	Hinton's	FOIL
Accuracy	7/8 (87.5%)	78/80 (97.5%)

Table 6.2: Classification accuracy of kinship domain on existing systems

are using this kinship domain to analyze several important aspects of FF01: they are accuracy, complexity and implementation. After carrying out a comprehensive set of experiments, we find that our implementation of the FOIL algorithm is not exactly the same as the one written by Quinlan. By varying the number of individuals n from 24 down to 12, we get 13 sets of experiments (which depending on n); each experiment is done for 5 times to get average running time, yielding a total of 65 sets of experiments. We implement our system by `Matlab` scripts (for details, please visit `www.mathworks.com`); the experiments are carried out on a `Sun Ultra 5/400` machine with 512MB of memory.

Accuracy

First we look at the accuracy of FF01 numerically. In Boolean classification tasks, four cases of classification are possible: true-positive T^+ (both the target and classification are true), true-negative T^- (both the target and classification are false), false-positive F^+ (classification is true but the target is false), false-negative F^- (classification is false but the target is true). Unlike attribute-value classification systems, which concerns on accuracy separately on the training sets and the testing (unseen) sets, testing sets are seldomly employed in first-order learning systems, since the number of cases are so large enough and we always need to *omit most training samples* (by appropriate pruning) throughout the loopings. FOIL measures the accuracy by considering merely the true classification cases, as shown in (6.1) :

$$A^+ = \frac{|T^+|}{|T^+| + |F^+|} \times 100\% \tag{6.1}$$

where $|X|$ is the size of the set X.

As the number of cases of false classification, as well as false target, are *numerous*, it is time consuming to consider all the false cases in the accuracy calculation. However, since we are working out specifying the training set by the positive samples only (please refer to **Section 5.2.1** for details), it's possible that for a concept description perfectly covers the positive target samples yet some false classification samples are positive target too, i.e. $A^+ = 100\%$ but $F^- \neq 0$. In other words, the negative classification samples are acting as the "unseen" samples as in attribute-value systems. Desprite it is computationally expensive, we still carry out the test on *all samples*, just to include all possible test cases. The complete accuracy measure is stated as in (6.2):

$$A = \frac{|T^+| + |T^-|}{|T^+| + |F^+| + |T^-| + |F^-|} \times 100\% \tag{6.2}$$

The results of FF01 running on different n are summerized in **Table 6.3**. Note that since FF01 is deterministic for Boolean data, all the 5 experiments run for the same n yield the same set of results.

n	12	13	14	15	16
A^+	$91\%(\frac{32}{35})$	$92\%(\frac{34}{37})$	$93\%(\frac{38}{41})$	$95\%(\frac{40}{42})$	$88\%(\frac{46}{52})$
A	$100\%(\frac{1725}{1728})$	$100\%(\frac{2025}{2028})$	$100\%(\frac{2349}{2352})$	$100\%(\frac{2698}{2700})$	$100\%(\frac{3066}{3072})$
n	17	18	19	20	21
A^+	$88\%(\frac{50}{57})$	$98\%(\frac{54}{55})$	$93\%(\frac{64}{69})$	$82\%(\frac{70}{85})$	$91\%(\frac{82}{90})$
A	$100\%(\frac{3461}{3468})$	$100\%(\frac{3887}{3888})$	$100\%(\frac{4327}{4332})$	$100\%(\frac{4785}{4800})$	$100\%(\frac{5284}{5292})$
n	22	23	24		**Total**
A^+	$94\%(\frac{92}{98})$	$94\%(\frac{98}{104})$	$93\%(\frac{112}{120})$		$91.8\%(\frac{812}{885})$
A	$100\%(\frac{5802}{5808})$	$100\%(\frac{6342}{6348})$	$100\%(\frac{6904}{6912})$		$99.9\%(\frac{52655}{52728})$

Table 6.3: FF01 classification accuracies on different n

We can see that the overall accuracy fluctuates when n changes, but there is no correlation between the fluctuation and the changes in n. Since the original experiment is designed by the description of 2 *complete* family trees, which

consists of 24 people, when we remove several members in the family trees, some "kinship links" might be broken. It makes the system being impossible to induce perfect solutions (so that the accuracy fluctuates), and it takes more time to learn a lengthy (the best possible) solution (so that we would see the running time fluctuates also). According to the A accuracy readings in **Table 6.3**, it shows that F^- bares only a very small amount of the total negative classification cases, that's why the A accuracy is seldomly reported in other systems. Go back to the usual accuracy measure, A^+, FFO1 performs very well *on average* by yielding a 91.8% correctness. Unfortunately, it could never achieve 100% accuracy even $n = 24$. We would leave this issue behind to be discussed in the "implementation" part.

Complexity

n	12	13	14	15	16	17	18
FF01 (sec)	50.1	55.5	51.1	57.9	66.0	64.2	93.6
FOIL (sec)	0.22	0.20	0.22	0.26	0.28	0.30	0.42
n	19	20	21	22	23	24	
FF01 (sec)	107.5	132.9	128.7	133.1	139.5	127.1	
FOIL (sec)	0.40	0.54	0.50	0.52	0.56	0.46	

Table 6.4: Running time on FOIL and FF01 on the kinship domain

From **Fig. 6.1**, we see that the dependence of t on n are pretty much the same in FOIL and FF01. But the fact is, FOIL runs much faster than FF01, before we explain the reasons accounting for the difference in speed, a complexity analysis is performed: Assumed that the running time t is a function of the number of individuals n in the form :

$$t(n) = \alpha + \beta n^{\theta} + e_n \tag{6.3}$$

where α, β and θ are the constants to estimate, and e_n is the error between the model and the actual reading. Using model in (6.3), we clain the complexity

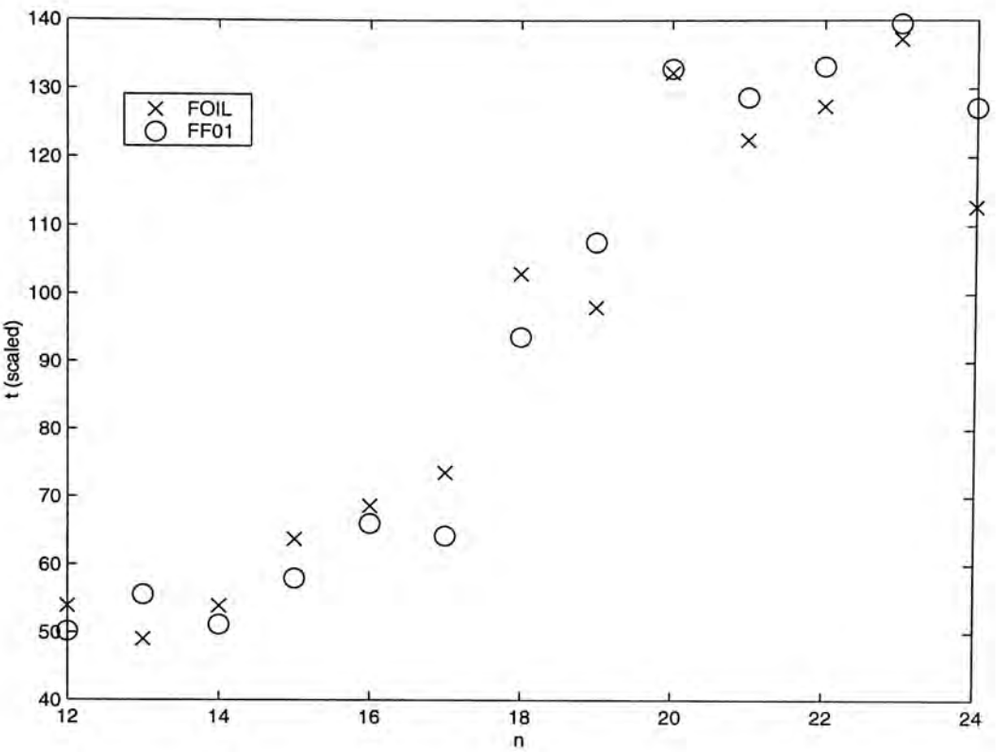


Figure 6.1: Average running time t vs. number of individuals n

of this kind system be $O(n^{\theta})$. The technical problem is how we estimate the parameters. By subtracting t by e, and taking logarithmic from both sides, we get a linear model with $\log\beta$ being the intercept and θ being the slope; this model could be easily solved by least square error. However, the whole estimation depends heavily on the choice of α. More precisely, for each α, we would get a optimal set of $\{\beta\ \theta\}$ and that set of parameters give rise to a set of e_n's.

$$RMSE(\alpha) = \sqrt{\frac{\sum_n {e_n}^2}{N}} \qquad \text{where } N \text{ is the number of different } n \quad (6.4)$$

By formulating a square-root mean-square error value as in (6.4), we expect a concave (U-shape) error function to minimize. As there is no analytical $RMSE$ guaranteed, we used a numerical bisection method to locate the optimal α, with respect to a minimum $RMSE$. The real data in **Fig. 6.2** is supporting our assumptions, as concave $RMSE$ functions are observed from both sets of

experiments. The optimal parameter sets are hence estimated and recorded in **Table 6.5**.

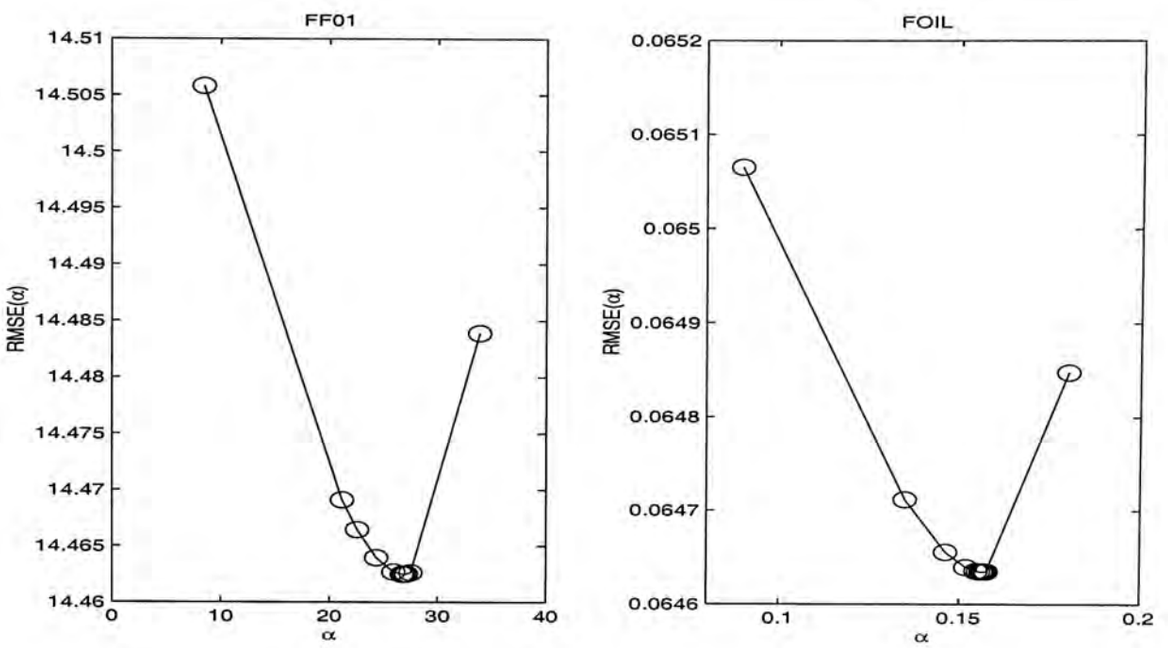


Figure 6.2: $RMSE$ as a concave function of α

	α	β	θ	$RMSE$
FF01	26.8	0.56	1.76	14.5
FOIL	0.157	0.0035	1.60	0.0646

Table 6.5: Optimal parameters estimated by minimizing $RMSE$

Insprite of the big difference in α, β and $RMSE$, the θ from both systems are basically the same. However, the results make us comfortable to claim that FF01 runs in $O(n^{1.8})$ *in this domain.* **Fig. 6.3** shows how the suggested running-time model describes the actual data. Note that, without suitable pruning and the use of the covering method, the complexity should be $O(n^V)$ (assuming all other factors kept constant), where V is the number of different variables allowed simultaneously. For example, our system allows up to 5 variables, which implies a $O(n^5)$ complexity ! In fact, before constructing

FF01, we tried to globally evaluate each literal and it spent days to explore the correct solutions. We would further discuss the difference in α, β and the running time between FF01 and FOIL in **Section 6.2.1**.

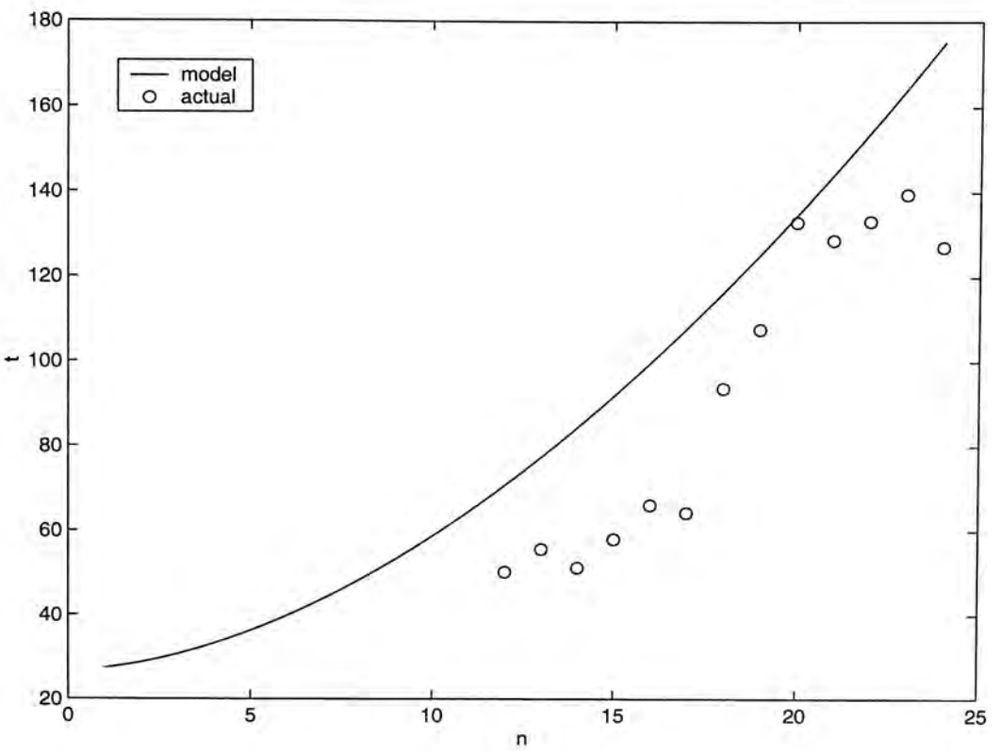


Figure 6.3: The actual running time and the expected running time from model

Implementation

The last two categories of analysis shows that there are some essential differences in the implementation of FF01 and FOIL.

First of all, FF01 could never learn the "correct" (by common sense in family relations) definitions of two relations, namely `Nephew` and `Niece`. We found that the basic FOIL algorithm, as described in **Section 5.2.1**, is not capable of inducing those relations. There are two main reasons:

1. The gain heuristics G is not as simple as stated in **Section 5.2.1**. In fact, for different literal forms, e.g. for those which introduce new variables,

G is multiplied by *some bias*. Unlike the basic gain heuristics, which has solid foundation in information theory, the bias are purely *experimental*. An optimal set of bias requires a large set of experiments and a lengthy fine-tuning process.

2. This point is more important: a greedy search is hardly perfect in all situations. The latest implementation of FOIL, the one we test, allows a limited backtracking to "guarantee" the best solutions. Whenever it checks that a concept description that is too "lengthy" and is "not likely" to get the perfect covering, it backtracks to select some alternatives. The problem is how could we determine what is "lengthy" and "not likely": it also requires a fine-tuning.

Second, despite the complexities are similar, the actual running time of FF01 is much longer that of FOIL. Reminded that FF01 is running on a `Matlab` *interpreted* environment, while FOIL is a *compiled*, highly optimized `C` executable. Nevertheless, we still see room for speed improvement, the possibilities would be discussed in **Section 6.2.1**.

In conclusion, this experiment domain shows that FF01 performs well in inducing Boolean first-order concepts. Thus we are confident in building on top of this basic FOIL algorithm to induce fuzzy first-order concepts.

6.1.3 The Fuzzy Relation Domain

We construct this artificial domain to test FF01 on the induction of fuzzy first-order concepts. This domain consists of 8 fuzzy relations, some are unary and some are binary, defining on a discrete type comprising up to 26 entities. The task is to induce the definition of $\mathtt{R}_7$ and $\mathtt{R}_8$, which are perfectly defined as :

$$\mathtt{R}_7(\mathtt{X},\mathtt{Y}) \quad :- \quad \mathtt{R}_1(\mathtt{X},\mathtt{Y}), \mathtt{R}_3(\mathtt{Y},\mathtt{X}) \tag{6.5}$$

$$\mathtt{R_8(X)} \quad :- \quad \mathtt{R_3(X,Y)}, \mathtt{R_2(Y)} \tag{6.6}$$

The literals $\mathtt{R_4}$, $\mathtt{R_5}$ and $\mathtt{R_6}$ are acted as "noise" as they are irrelevant to the two targets. (6.5) tests if FF01 could learn definition in correct variable orderings and (6.6) tests the performance on fuzzy variable projection (see **Section 2.4.4**). Also, the three fuzzy methods (fixed-α (fa), adaptive-α (aa), probabilistic (p)) will be compared and evaluated.

First, we have to formulate a evaluation method to quantify the solutions, as the accuracy measures described in **Section 6.1.2** are only valid for Boolean cases. In particular, the quantities T^+, T^-, F^+ and F^- have to be fuzzified. Following the rules of fuzzification in **Section 5.2.2**, we propose the following measurements:

$$\widetilde{T^+} = \sum_i \mu_C(i) \times (1 - |\mu_T(i) - \mu_C(i)|) \tag{6.7}$$

$$\widetilde{T^-} = \sum_i (1 - \mu_C(i)) \times (1 - |\mu_T(i) - \mu_C(i)|) \tag{6.8}$$

$$\widetilde{F^+} = \sum_i \mu_C(i) \times |\mu_T(i) - \mu_C(i)| \tag{6.9}$$

$$\widetilde{F^-} = \sum_i (1 - \mu_C(i)) \times (1 - |\mu_T(i) - \mu_C(i)|) \tag{6.10}$$

where $\mu_C(i)$ and $\mu_T(i)$ are the class membership of the i-th sample of the concept description and the target respectively, and $\tilde{\bullet}$ refers to the fuzzified version of $\bullet$.

As shown in **Section 6.1.2**, the A measurement shows little information, so we ignore it in this experiment and concentrate in fuzzifying the A^+ measurement, which is denoted as $\tilde{A}$:

$$\tilde{A} = \frac{|\widetilde{T^+}|}{|\widetilde{T^+}| + |\widetilde{F^+}|} \times 100\% \tag{6.11}$$

We adjust the size of the type, n, from 16 to 26. For each n, we carry out the fixed-α (fa) approach once, the adaptive-α (aa) approach once, and the probabilistic approach for 10 times. The *overall* accuracies (over the two

concept definitions) are reported in **Table 6.6** and are plotted in **Fig. 6.4**. The best (worst) performance of the 10 probabilistic experiments are denoted as $max\ \tilde{A}_p$ ($min\ \tilde{A}_p$).

n	16	17	18	19	20	21
$\tilde{A}_{fa}$	$88\%(\frac{15.6}{17.7})$	$87\%(\frac{16.2}{18.6})$	$87\%(\frac{17.0}{19.5})$	$87\%(\frac{17.8}{20.6})$	$88\%(\frac{18.7}{21.4})$	$87\%(\frac{20.1}{23.0})$
$\tilde{A}_{aa}$	$83\%(\frac{17.4}{20.9})$	$74\%(\frac{22.4}{30.2})$	$79\%(\frac{20.7}{26.0})$	$79\%(\frac{22.0}{27.7})$	$79\%(\frac{22.1}{28.0})$	$87\%(\frac{20.1}{23.0})$
$min\ \tilde{A}_p$	$72\%(\frac{16.8}{23.4})$	$70\%(\frac{15.2}{21.9})$	$75\%(\frac{18.3}{24.3})$	$70\%(\frac{29.9}{42.9})$	$77\%(\frac{24.6}{31.9})$	$75\%(\frac{23.6}{31.4})$
$max\ \tilde{A}_p$	$81\%(\frac{19.1}{23.5})$	$81\%(\frac{19.8}{24.3})$	$84\%(\frac{19.7}{23.4})$	$78\%(\frac{18.0}{23.1})$	$87\%(\frac{19.7}{22.6})$	$81\%(\frac{22.5}{27.7})$

n	22	23	24	25	26	**Total**
$\tilde{A}_{fa}$	$87\%(\frac{20.8}{23.8})$	$87\%(\frac{21.5}{24.7})$	$87\%(\frac{21.5}{24.7})$	$87\%(\frac{22.1}{25.5})$	$86\%(\frac{23.2}{26.8})$	$87.1\%(\frac{214.5}{246.3})$
$\tilde{A}_{aa}$	$87\%(\frac{20.8}{23.8})$	$87\%(\frac{21.5}{24.7})$	$87\%(\frac{21.5}{24.7})$	$87\%(\frac{22.1}{25.5})$	$79\%(\frac{30.6}{38.7})$	$82.3\%(\frac{241.2}{293.2})$
$min\ \tilde{A}_p$	$74\%(\frac{25.9}{35.2})$	$77\%(\frac{32.4}{42.1})$	$79\%(\frac{31.9}{40.4})$	$76\%(\frac{34.6}{45.7})$	$78\%(\frac{35.6}{45.5})$	$75.1\%(\frac{288.8}{384.7})$
$max\ \tilde{A}_p$	$86\%(\frac{21.7}{25.2})$	$83\%(\frac{23.8}{28.7})$	$85\%(\frac{23.8}{27.9})$	$82\%(\frac{23.1}{28.1})$	$82\%(\frac{26.3}{32.0})$	$82.9\%(\frac{237.5}{286.5})$

Table 6.6: Accuracies of different methods in FF01 on different n

The accuracy obtained is decent: an average of 87% correctness is achieved by the fixed-α approach. Suprisingly, the adaptive α method is always inferior to the fixed α method. It really violates our expectation and the result is unknown yet. Consider the normal solutions get by the fixed-α :

$$\mathtt{R_7(X,Y)} \quad :- \quad \mathtt{R_1(X,Y), R_3(Y,X)} \tag{6.12}$$

$$\mathtt{R_8(X)} \quad :- \quad \mathtt{R_3(X,Y)} \tag{6.13}$$

in which (6.12) matches (6.5), but (6.13) does not match (6.6). Moreover, by using whatever α methods, (6.6) is never achieved. It shows the deficiency of the α-covering methods as discussed in **Section 5.4**. Despite that the probabilistic approach always performs worse in term of *overall* accuracies, it does successfully induce the correct solution as in (6.6) *in some occasions.* But don't expect that the probabilistic approach could induce both definitions correctly at the same time, the opportunity is endlessly vast. However, it

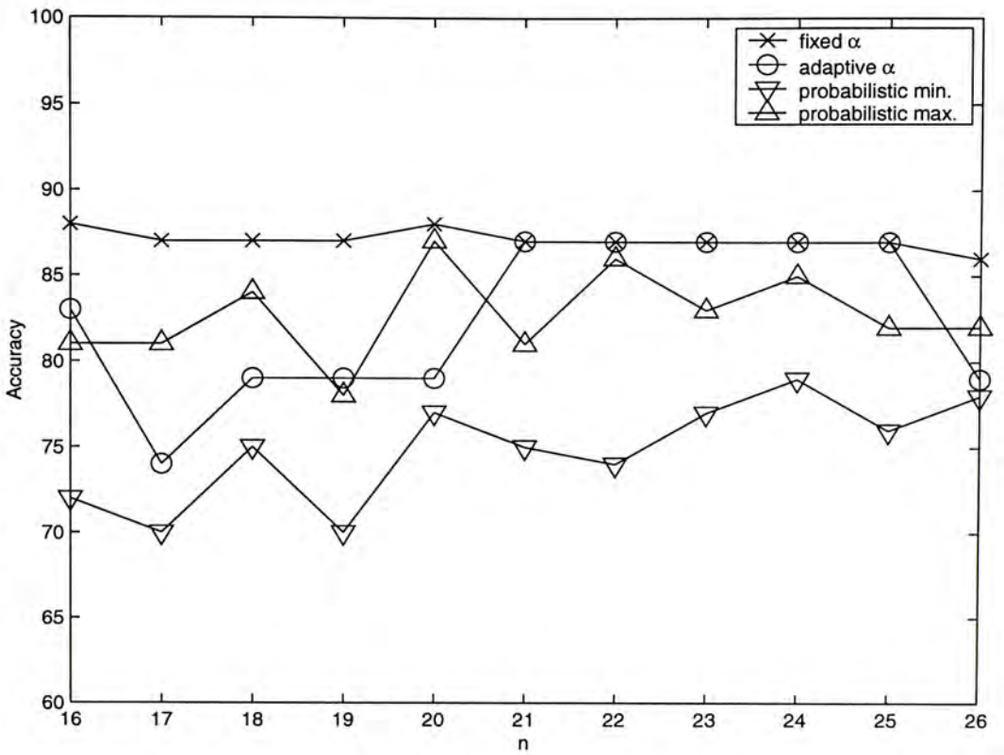


Figure 6.4: Accuracy of different methods

demonstrates that the probabilistic approach is an alternative to seek a better solution if the deterministic methods do not work.

Next, we look at the running time of this experiment set as reported in **Table 6.7**. By observing **Fig. 6.5**, we see that the most accurate method spends the least time. And it is very encouraging to use only 15.4 sec in the $n = 26$ case. However, we must go further to analysis its complexity.

n	16	17	18	19	20	21	22	23	24	25	26
t_{fa}	4.3	4.8	5.5	6.1	6.9	7.9	9.0	10.3	11.7	13.6	15.4
t_{aa}	8.5	11.5	11.6	13.4	10.5	7.8	8.9	10.1	11.5	13.3	85.2
$min\ t_p$	12.3	16.9	11.9	10.8	22.2	23.3	27.5	25.0	41.1	33.0	63.7
$max\ t_p$	27.3	25.0	22.1	29.3	37.8	53.7	53.0	50.8	68.0	122.2	131.4

Table 6.7: Running time on different methods in FF01 domain

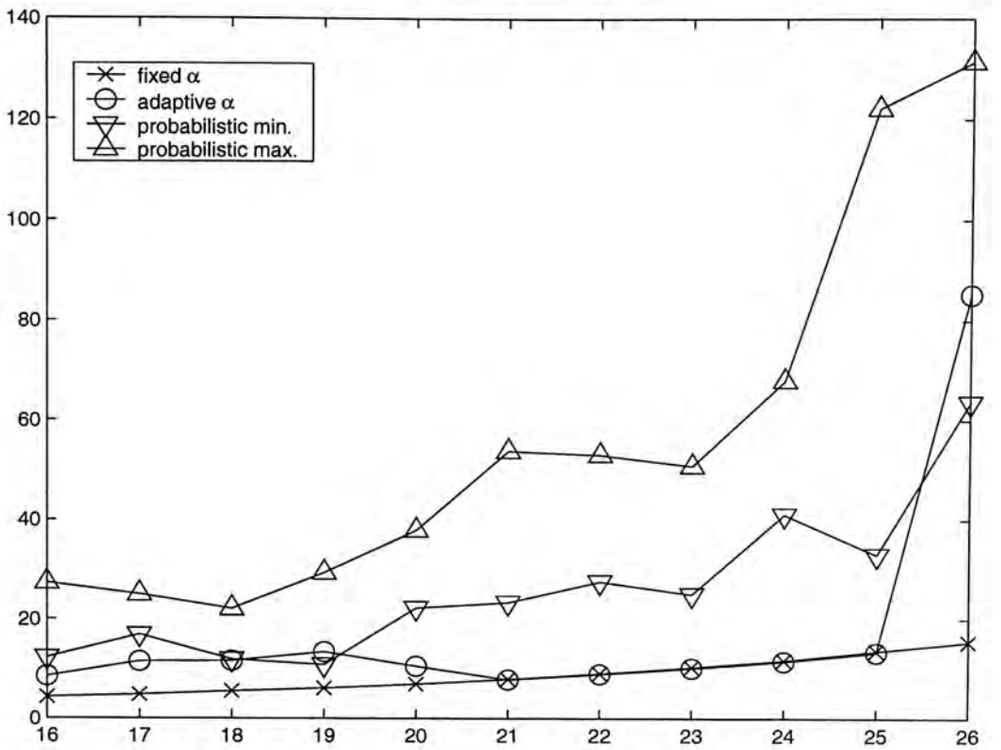


Figure 6.5: Running time of different methods t vs. number of individuals n

Following the same procedures as described in **Section 6.1.2**, we find that (6.14) fits the actual running time very well as shown in **Fig. 6.6**.

$$t = 1.94 + 1.11 \times 10^{-4} \times n^{3.59} \tag{6.14}$$

Unlike that in the kinship experiment, FF01 has a much higher complexity, $O(n^{3.59})$, instead of $O(n^{1.76})$. The reason accounts for the grow is that whenever the system could not learn a perfect definition, it spends much more time in constructing a much bigger (more, longer clauses) solution. We would suggest ways to cut down the complexity in **Section 6.2.1**.

Among the suggested fuzzy covering methods, fixed-α seems to be the best. The probabilistic approach gives the opportunity to get a perfect solution if other methods fail. The overall accuracy and the speed of the fuzzy first-order induction task in satisfactory.

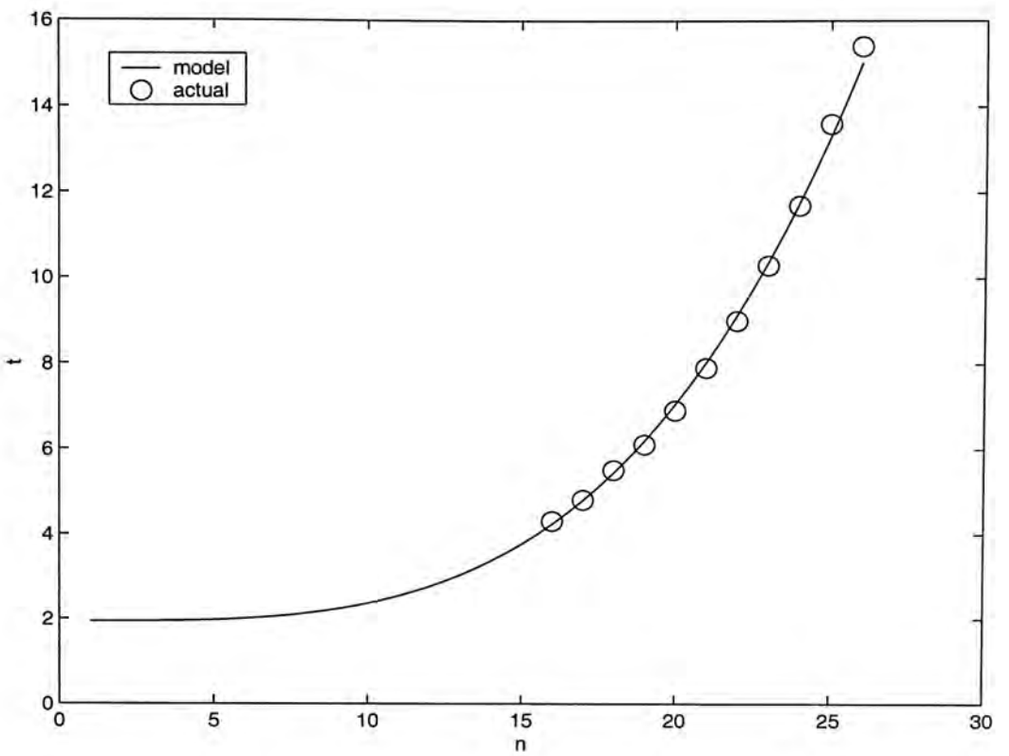


Figure 6.6: The actual running time and the expected running time from model

6.1.4 Age Group Domain

The previous two experiments show that our system is compatible to existing learning tasks, no matter they are attribute-value based or relational. Now, we step forward to learn concepts from fuzzy data. This artificially constructed domain consists of the age of 20 people and the degree of membership of the fuzzy relations: `middle_aged`, `much_older_than`, `young` and `old`. If we directly run FF01, we get the definition of `middle_aged` as:

```
middle_aged(X) :-medium(X)(10,30,48,60)
```

where `medium(X)` is automatically generated for the numeric universe `X` (which means the age in this experiment). However, this concept is quite useless: we repeat the experiment by suppressing the automatic generation of literals

(please refer to **Table 3.6**), we get a more meaning result:

```
middle_aged(X) :-much_older_than(Y,X),much_older_than(X,Z)
```

which is read as "someone is middle-aged if he is much older than some guy and another guy is much older than him". This time, the concept can be applied universely on all situations.

This experiment demonstrates that FF01 is able to learn both fuzzy and first-order concepts. It opens a new area of machine learning tasks. We can see its numerous potential applications, for example, data mining of questionnaire results, structural image understanding, etc.

6.1.5 The NBA Domain

From of the official site of the Nation Basketball Association (NBA), `www.nba.com`, we collect the data for this domain to analyze the *relative* performance between players and terms. The data are recorded into three spreadsheets: one for the career statistics for individual player (denoted as S1), the other two for the results of 55 games, including the statistics of the game (denoted as S2) and the performance of the starters (denoted as S3).

Since the raw data spreadsheets are presented in a attribute-value (zeroth-order) manner, we have to transform them into relational format. First of all, we have to set the types where the relations defined on. In this dataset, all the attributes are describing the performance of a player, a team, or a particular player on a particular team. Therefore, we set two types, namely "player" and "team", which are discrete and disordered. The "player" type contains 61 entities while there are 12 different "team" type values.

First we consider S1, there are 16 numeric attributes as listed in **Table 6.8**. Note that the fields from `MIN` to `PTS` are the averages of those attributes, for example, `PTS` = 9.1 means that a player owns a career average of 9.1 points per game.

Name	Meaning	Name	Meaning	Name	Meaning
H	height in inch	W	weight in lbs	AGE	age
EXP	year played	G	game played	MIN	minutes
FGP	field-goal percentage	3PP	3-pointer FGP	FTP	free-throw percentage
AST	assists	ST	steals	BLK	blocks
TO	turn-overs	PF	personal fouls	REB	rebounds
PTS	points				

Table 6.8: Name of numeric fields and their meaning in S1

Since FF01 offers the functionality to fuzzify numeric attributes into linguistic terms (fuzzy unary relations), the attributes in S1 need no special treatments. For example, from the PTS field, FF01 would automatically generates 5 predicates, namely very_large_PTS to very_small_PTS, by the method discussed in **Section 3.1.3**. The predicates generated are unary relations defining on the type "player", for instance, some instances of the predicate "very_large_PTS" are shown in **Table 6.9**. Although one might prefer the name high_scorer to very_large_PTS, we can just accept this tradeoff between predicate naming and automatic predicate generation.

player	JORDAN	CARTER	SPREWELL	MOURNING	HAMILTON
$\mu_{\texttt{very\_large\_PTS}}$(player)	1.0	0.9	0.1	0.1	0.0

Table 6.9: Some instances of very_large_PTS(player)

In comparison with S1, we could get more information from S2. First of all, we gather the statistics for each team, calculate the average numbers over the games, and then the difference between the average and each individual game is calculated. All these calculated fields are considered are fed into the system as unary relations. The statistics for players (S3) are processed in the same way.

After gathering these vast amount of information, FF01 "discovers" several

interesting information, for example:

```
very_small_PTS(X) :-matchup(X,Y), very_large_H(Y)
```

which means "a player would score few if he matchups with a very tall guy". While the clause :

```
very_large_FGP(X) :-sameterm(X,Y), very_large_AST(Y)
```

which means "a player would obtain very high field-goal percentage if one of his teammate dishes many". Also :

```
very_large_PTS(X) :-very_large_REB(X), very_large_FGP(X)
```

reads as "a team gets very high scores if it rebounds well and shoots accurately".

Note that not the results are as results as the above ones. In fact, many results are nonsense, mainly due to the small amount of data. Also, the "good" solution are usually not obtained by deterministic methods, but by the probabilistic approach, and it require human intervention to pick up the "useful" solution out from the pool of possible solutions.

6.2 Future Development Directions

This section discuss a few possible improvements on our system to solve the problems we encountered in previous sections.

6.2.1 Speed Improvement

In **Section 6.1.2**, we found that FF01 spends long in initialization (as shown in a large α). It is simple and easy to maintain to define all the literals in the initialization stage, however, it's time consuming. A better method is to generate a literal once it's needed, and save it for later use. The difference of

β comes from the lack of use of "determinate literals" [11]. This idea could be incorporated in our system to further prune the search space.

In **Section 6.1.3**, we know that the speed slows down whenever perfect solution is not achieved. To avoid this, we have to keep track of the number of clauses learned and the length of the current clause, if the concept definition tends to grow with no "significant" improvement, it should be stopped and backtracked. The issue to concern is how we determine which we should stop and backtrack.

Obviously, plotting our code from Matlab to C would greatly increase the speed. However, as Matlab facilities us to develop a potential-user interface, we don't see the necessity of plotting as long as the running time is acceptable.

6.2.2 Accuracy Improvement

By now, once we could not get the perfect solution, we go for the probabilistic approach. However, once we effectively use the backtracking techniques, we may not need to do so.

Moreover, the current implementation of the probabilistic approach is too simple: it has no memory among each running of the probabilistic method. If we could keep track of the partitions in each run, we may know that which samples should really be covered or not. With that knowledge, we could further adjust the opportunity of which a sample is picked up. This idea is similar to genetic algorithm, which "remembers" the good solution and at the same time allows the possibility to try a better solution.

6.2.3 Others

Generalizing slightly, we can identify te following features that will be required by any robust system for learning fuzzy logic programs:

User interface: As shown in **Section 6.1.5**, to use FF01 in real-life requires

a large amount of human intervention. Not including the basic data entry process, we have to analysis, process each process and create some useful relations. The more we need to do this, the less useful is our system in automatic data mining. So, a friendly user-interface is needed to help the user to create the relational dataset from attribute-value data.

Extended treatment of numeric fields: Not many first-order systems seem to have addressed the issue of using continuous-valued information. FF01 already extended the use of numeric fields by several new literal forms. Remember that FOIL's use of numeric fields is limited to thresholding and comparisons of known values rather than computing new values. Since many practical Prolog programs involve computation, learning systems that are intended to generate these programs must somehow come to grips with computational clauses.

Treatment of recursive definitions: We currently follow [31] to handle recursive relations (the concept description involving the target relation). However, the theory behind the paper cannot be applied in fuzzy systems. Particularly, we cannot order the fuzzy constants as stated in the paper. Thus, a new treatment of recursive relations is needed.

In conclusion, throughout these experiment sets, we have shown that FF01 is capable to due with a wide variety of induction problems, from Boolean attribute-value data to fuzzy first-order concepts. We have also performed detailed analysis on the accuracy and the complexity of FOIL, both of them are shown to be satisfactory.

Chapter 7

Conclusion

This thesis shows that the fuzzy first-order logic is a powerful representation language. With the definition of several new literal forms, FF01 is capable on a diverse range of learning tasks, which include the conventional attribute-value classification problems, the crisp relational concept learning problem and the robust, comprehensive manipulation of numeric attributes. And the system has high potential on real world tasks that, while currently lacking the structure necessitating a fuzzy first-order learner, may acquire such structure in future.

We solve the problem by first introducing a set of literal forms to describe fuzzy relational concepts and the forms are shown to be very powerful in representing human-like knowledge. The next step is to explore the ways to induce concepts on such literal forms. We have developed FF99, though it could not fulfill our goal completely, it acts as an alternative: It is a novel fuzzy zeroth-order learning system and the nodal characteristics we have discovered are worth discussing.

To achieve our objective, we try to fuzzify FOIL. We first analysis the uniqueness and reasons of success of it, then we explore the ways to fuzzify it. After formulating the constraints, we have developed the α-covering method, which is then improved to the adaptive-α-covering method. The method is good, yet still fails sometimes. Thus we propose a probabilistic covering approach to improve the situation.

Experimental results have shown that FF01 is generally accurate and robust; the complexity would be kept in $O(n^{1.8})$ if perfect solution could be found. Finally, we have proposed some ways to further improve FF01.

Bibliography

[1] J. H. Kim, *CONVINCE: A Conversational Inference Consolidation Engine.* PhD thesis, University of California, Los Angeles, 1983.

[2] D. Heckerman, *Probabilistic Similarity Networks.* Cambridge, Massachusetts: MIT Press, 1991.

[3] E. Shortliffe, "Consultation system," in *Computer-Based Medical Consultations: MYCIN*, pp. 79–158, Elsevier Computer Science Library, Artificial Intelligence Series, Book2, 1976.

[4] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[5] G. F. L. and W. A. S., *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Addison Wesley, 2nd ed.

[6] T. M. Mitchell, R. M. Keller, and S. T. Kedar, "Explanation-Based Generalization: A unifying view," *Machine Learning*, pp. 47–80, 1986.

[7] D. G. and R. Mooney, "Explanation-Based Learning: An alternative view," *Machine Learning*, pp. 147–176, 1986.

[8] M. Minsky and S. Papert, *Perceptrons.* Cambridge, MA: MIT Press, 1969.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing:*

Explorations in the Microstructure of Cognition (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, pp. 318–362, Cambridge, MA: MIT Press, 1986.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, New Jersey: Prentice-Hall International, Inc, 1995.

[11] Q. J. R., "Learning Logical Definitions from Relations," *Machine Learning*, pp. 239–266, 1990.

[12] S. Muggleton and F. Cao, "Efficient Induction of Logic Programs," in *Proceedings of the Workshop on Algorithmic Learning Theory*, (Tokyo), 1990.

[13] R. S. Michalski, "Pattern Recognition as Rule-guided Inductive Inference," in *Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 349–361, 1980.

[14] M. Pazzani and D. Kibler, "The utility of knowledge in inductive learning.," *Machine Learning*, vol. 9, pp. 57–94, 1992.

[15] J. R. Quinlan and R. M. Cameron-Jones, "FOIL: A Midterm Report," in *Proceedings of the European Conference on Machine Learning (ECML-93)* (P. B. Brazdil, ed.), vol. 667 of *LNAI*, (Vienna, Austria), pp. 3–20, Springer Verlag, Apr. 1993.

[16] R. Cameron-Jones and J. Quinlan, "First Order Learning, Zeroth Order Data," 1993.

[17] J. Kacprzyk and C. Iwanski, "Fuzzy Logic with Linguistic Quantifiers in Inductive Learning," *Fuzzy Logic for the Management of Uncertainty*, pp. 465–478, 1992.

[18] C. J. Tsai, S. S. Tseng, C. H. Wang, and C. T. Y. M. F. Jiang, "A Fuzzy Iductive Learning Algorithm for Parallel Loop Scheduling," Master's thesis, Institute of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, 1997.

[19] C. Wang, J. Liu, T. Hong, and S. Tseng, "A Fuzzy Inductive Learning Strategy for Modular Rules," 1999.

[20] C. M. Higgins and R. M. Goodman, "Learning Fuzzy Rule-Based Neural Networks for Control," in *Advances in Neural Information Processing Systems* (S. J. Hanson, J. D. Cowan, and C. L. Giles, eds.), vol. 5, pp. 350–357, Morgan Kaufmann, San Mateo, CA, 1993.

[21] P. Smyth and R. M. Goodman, "An Information Theoretic Approach to Rule Induction from Databases," *IEEE Trans. On Knowledge And Data Engineering*, vol. 4, pp. 301–316, 1992.

[22] G. Chen, *FUZZY LOGIC IN DATA MODELING, Semnatics, Constraints, and Database Design.* Boston: Kluwer Academic Publishers, 1998.

[23] T. Munakata and Y. Jani, "Fuzzy Systems: An Overview," *Communications of the ACM*, vol. 37, pp. 69–76, Mar. 1994.

[24] M. M. H. Yasui, Y. Hamada, "Fuzzy Prolog based on Lukasiewicz implication and bounded product," *IEEE Transactions on Fuzzy Systems*, pp. 949–954, 1995.

[25] L. A. Zadeh, "A Computational Approach to Fuzzy Quantifiers in Natural Language," *Computers and Mathematics with Applications*, vol. 9, no. 1, pp. 149–184, 1983.

[26] C. Hinde, "Fuzzy Prolog," *International Journal of Man-Machine Studies*, vol. 24, pp. 569–595, 1986.

[27] T. P. Martin, J. F. Baldwin, and B. W. Pilsworth, "The Implementation of FPROLOG – A Fuzzy Prolog Interpreter," *Fuzzy Sets and Systems*, vol. 23, no. 1, pp. 119–129, 1987.

[28] J. Baldwin, T. Martin, and B. Pilsworth, "FRIL - Fuzzy and Evidential Reasoning in Artificial Intelligence," 1995.

[29] M. Mukaidono, Z. Shen, and L. Ding, "Fundamentals of Fuzzy Prolog," *International Journal of Approximate Reasoning*, vol. 3, no. 2, pp. 179–193, 1989.

[30] Q. J. R., *C4.5: Programs for Machine Learning.* San Mateo, California: Morgan Kaufmann, 1993.

[31] R. Cameron-Jones and J. Quinlan, "Avoiding pitfalls when learning recursive theories," 1993.

[32] M. F. T. K.S. Leung, Irwin King, "FF99: A Novel Fuzzy First-Order Logic Learning System," *IEEE International Conference on Systems, Man, and Cybernetics*, 1999.

[33] L. A. Zadeh, "Calculus of fuzzy restrictions," in *Fuzzy Sets and their Applications to Cognitive and Decision Processes* (L. A. Zadeh, K.-S. Fu, K. Tanaka, and M. Shimura, eds.), (New York), pp. 1–40, Academic Press, 1975.

[34] He and Xin-Gui, *Fuzzy Database Systems.* China: Tsinghua University Publishing House, 1994.

[35] Y. T. S. K. S. Leung, "Consistency Checking for Fuzzy Expert Systems," *International Journal of Approximate Reasoning*, vol. 9, pp. 263–282, 1993.

[36] S. C. E., "A Mathematical Theory of Communication," *Bell System Tech. J.*, pp. 379–423, 623–656, 1948.

[37] S. Ihara, *Information Theory for Continuous Systems.* World Scientific.

[38] E. B. Hunt, J. Marin, and P. T. Stone, *Experiments in Induction.* New York: Academic Press, 1966.

[39] J. R. Quinlan, "Induction of Decision Trees," in *Readings in Machine Learning* (J. W. Shavlik and T. G. Dietterich, eds.), Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[40] R. L. Cannon, J. V. Dave, and J. C. Bezdek, "Efficient implementation of the fuzzy C-means clustering algorithms," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, pp. 248–255, Mar. 1986.

[41] C. Wang, T. Hong, and S. Tseng, "Inductive learning from fuzzy examples," *IEEE Trans. on Knowledge and Data Engineering*, pp. 13 – 18, 1996.

Appendix A

C4.5 to FOIL File Format Conversion

Mike Cameron-Jones wrote a program named 'c4tofoil.c' to convert files from the standard C4.5 input format (*.data, *.names and *.test) to a form that can be used by FOIL (*.d). Our system, FF01, can also read the *.d input format. To use this utility, we only need to modify the *.names file to specify the type names to be used in our system. The utility is embedded in the FOIL package.

To illustrate, we include a credit approval domain dataset in C4.5 format. The dataset contains three files: 'crx.names', 'crx.data' and 'crx.test'. They are partially listed as follows. Note that type names are appended in 'crx.names'.

'crx.names' contains the description of classes and attributes:

```
+, -. | classes

A1: b, a. |type: A
A2: continuous. |type: B
A3: continuous. |type: B
A4: u, y, l, t. |type: C
A5: g, p, gg. |type: D
A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff. |type: E
A7: v, h, bb, j, n, z, dd, ff, o. |type: E
A8: continuous. |type: B
A9: t, f. |type: F
A10: t, f. |type: F
A11: continuous. |type: B
A12: t, f. |type: F
A13: g, p, s. |type: G
A14: continuous. |type: H
A15: continuous. |type: H
```

'crx.data' contains the training data :

```
b,30.83,0,u,g,w,v,1.25,t,t,01,f,g,00202,0,+
a,58.67,4.46,u,g,q,h,3.04,t,t,06,f,g,00043,560,+
a,24.50,0.5,u,g,q,h,1.5,t,f,0,f,g,00280,824,+
b,27.83,1.54,u,g,w,v,3.75,t,t,05,t,g,00100,3,+
b,20.17,5.625,u,g,w,v,1.71,t,f,0,f,s,00120,0,+
b,32.08,4,u,g,m,v,2.5,t,f,0,t,g,00360,0,+
b,33.17,1.04,u,g,r,h,6.5,t,f,0,t,g,00164,31285,+
a,22.92,11.585,u,g,cc,v,0.04,t,f,0,f,g,00080,1349,+
b,54.42,0.5,y,p,k,h,3.96,t,f,0,f,g,00180,314,+
b,42.50,4.915,y,p,w,v,3.165,t,f,0,t,g,00052,1442,+
b,22.08,0.83,u,g,c,h,2.165,f,f,0,t,g,00128,0,+
b,29.92,1.835,u,g,c,h,4.335,t,f,0,f,g,00260,200,+
a,38.25,6,u,g,k,v,1,t,f,0,t,g,00000,0,+
...
```

'crx.test' contains the testing data :

```
a,47.25,0.75,u,g,q,h,2.75,t,t,01,f,g,00333,892,+
b,24.17,0.875,u,g,q,v,4.625,t,t,02,t,g,00520,2000,+
```

```
b,39.25,9.5,u,g,m,v,6.5,t,t,14,f,g,00240,4607,+
a,20.50,11.835,u,g,c,h,6,t,f,0,f,g,00340,0,+
...
```

The utility converts the files into the FOIL and FF01 compatible 'crx.d' as follows:

```
#A: *Ab, *Aa.
B: continuous.
#C: *Cu, *Cy, *Cl, Ct.
#D: *Dg, *Dp, *Dgg.
#E: *Ec, *Ed, *Ecc, *Ei, *Ej, *Ek, *Em, *Er, *Eq, *Ew, *Ex, *Ee, *Eaa, *Eff, *Ev, *Eh, *Ebb, *En, *Ez, *Edd, *Eo
#F: *Ft, *Ff.
#G: *Gg, *Gp, *Gs.
H: continuous.

is_+(A,B,B,C,D,E,E,B,F,F,B,F,G,H,H)
Ab,30.83,0,Cu,Dg,Ew,Ev,1.25,Ft,Ft,01,Ff,Gg,00202,0
Aa,58.67,4.46,Cu,Dg,Eq,Eh,3.04,Ft,Ft,06,Ff,Gg,00043,560
Aa,24.50,0.5,Cu,Dg,Eq,Eh,1.5,Ft,Ff,0,Ff,Gg,00280,824
Ab,27.83,1.54,Cu,Dg,Ew,Ev,3.75,Ft,Ft,05,Ft,Gg,00100,3
Ab,20.17,5.625,Cu,Dg,Ew,Ev,1.71,Ft,Ff,0,Ff,Gs,00120,0
Ab,32.08,4,Cu,Dg,Em,Ev,2.5,Ft,Ff,0,Ft,Gg,00360,0
Ab,33.17,1.04,Cu,Dg,Er,Eh,6.5,Ft,Ff,0,Ft,Gg,00164,31285
...
Aa,33.67,0.375,Cu,Dg,Ecc,Ev,0.375,Ff,Ff,0,Ff,Gg,00300,44
Ab,48.58,0.205,Cy,Dp,Ek,Ev,0.25,Ft,Ft,11,Ff,Gg,00380,2732
;
Ab,32.33,7.5,Cu,Dg,Ee,Ebb,1.585,Ft,Ff,0,Ft,Gs,00420,0
Ab,34.83,4,Cu,Dg,Ed,Ebb,12.5,Ft,Ff,0,Ft,Gg,?,0
...
Ab,18.83,3.54,Cy,Dp,Eff,Eff,0,Ff,Ff,0,Ft,Gg,00180,1
?,45.33,1,Cu,Dg,Eq,Ev,0.125,Ff,Ff,0,Ft,Gg,00263,0
.

is_+
Aa,47.25,0.75,Cu,Dg,Eq,Eh,2.75,Ft,Ft,01,Ff,Gg,00333,892:+
Ab,24.17,0.875,Cu,Dg,Eq,Ev,4.625,Ft,Ft,02,Ft,Gg,00520,2000:+
...
Ab,17.92,0.205,Cu,Dg,Eaa,Ev,0.04,Ff,Ff,0,Ff,Gg,00280,750:-
Ab,35.00,3.375,Cu,Dg,Ec,Eh,8.29,Ff,Ff,0,Ft,Gg,00000,0:-
.
```

Note that 'crx.d' contains three main blocks as discussed in **Section 3.2.1**. The blocks specify the types, the training relations and the test cases all in one file.

Appendix B

FF99 EXAMPLE

The following MATLAB script FF99exam.m contains the example in **Section 4.5**. It includes the definitions of literals and the whole learning loop.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FF99exam.m --- FF99 example
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function FF99exam
clear all
clc
tic

% define the literals and the sample space
X = 1:10000;
L{1} = pimf(X, [6000 6500 8500 9500]);
L{2} = pimf(X, [6500 7000 9000 9800]);
L{3} = 0.7*pimf(X, [5000 5500 9200 9900]);
L{4} = pimf(X, [1000 2000 2500 3500]);
L{5} = pimf(X, [0 3800 4000 4300]);

L{6} = pimf(X, [500 2300 2500 2900]);
L{7} = pimf(X, [4000 4800 5800 6200]);
L{8} = 1-L{7};
L{9} = pimf(X, [5300 5900 6600 7200]);
L{10} = 1-L{9};


T = max(min(L{4},L{5}),min(min(L{1},L{2}),L{3})); % the target relation
C = [];  % the concept description
N = 10;

figure(1);
hold on;
for i=1:N,
    plot(X,L{i},':');
end
plot(X,T,'r');


% main learning loop
currC = 1;
currL = 1;
n = 1;
C{currC} = [];

while 1,
   for i=1:N,
      if isempty(C{currC}) | isempty(find(C{currC} == i)),
         % form a new C and and a new E
         muC = zeros(size(X));
         for c = 1:currC-1,
            muCcurr = ones(size(X));
            for l = 1:length(C{c}),
               muCcurr = min(muCcurr,L{C{c}(l)});
            end
            muC = max(muC, muCcurr);
         end
         muCcurr = ones(size(X));
         for l = 1:length(C{currC}),
            muCcurr = min(muCcurr,L{C{currC}(l)});
         end
         muCcurr = min(muCcurr,L{i}); % append the current literal
         muC = max(muC, muCcurr);
```

```
        E = T - muC;
        [Enormpos, Enormneg, Xnorm, MSE(i), cepos(i), ceneg(i), cost(i)] = FF99(E,X);
      else
        cost(i) = 1; % exclude the existing literal
      end
   end

   save(['FF99examdat' sprintf('%d',n)], 'MSE', 'cepos', 'ceneg', 'cost');
   mini = find(cost == min(cost));
   C{currC}(currL) = mini;
   n = n + 1;

   if cost(mini) == 0,
      break; % terminate the whole learning
   end

   if ceneg(mini) == 0,% terminate the current clause
      currC = currC + 1;
      currL = 1;
      C{currC} = [];
   else
      currL = currL + 1;
   end
end

% display results
for c = 1:currC,
   txt = 'C :- ';
   for l = 1:length(C{c}),
      txt = [txt sprintf('%d, ', C{c}(l))];
   end
   disp(txt);
end

toc
```

FF99exam.m calls the routines supplied by the MATLAB Fuzzy Logic Toolbox. Also, it invokes the function in FF99.M, which is the procedures to evaluate a literal according to the error function supplied.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The heuristics used in FF99
%
% input: E, X
%   E = a vector of errors (the error function)
%   X = a vector being the universe for E; if X=[], a uniform universe over [0 1] is assumed
%       it should be sorted ascending in advance
%
% output: [Enormpos, Enormneg, Xnorm, MSE, cepos, ceneg, cost]
%   Enormpos = the normalized positive error function (= [] if does not exist)
%   Enormneg = the normalized negative error function (= [] if does not exist)
%   Xnorm = the normlized universe (all sorted)
%   MSE = mean square error
%   cepos = continuous entropy of the normalized positive error function (= 0 if does not exist)
%   ceneg = continuous entropy of the normalized negative error function (= 0 if does not exist)
%           both cepos and ceneg are normalized to [0 1] (0 for the best <=> impulse)
%   cost = (MSE+cepos+ceneg)/3
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Enormpos, Enormneg, Xnorm, MSE, cepos, ceneg, cost] = FF99(E,X)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MSE
MSE = mean(E.^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Normalization
if isempty(X),% X=[], suitable for ordinal universe
   Xnorm = 0:1/(length(E)-1):1;
else
   Xnorm = (X-X(1))/(X(end)-X(1)); % map to [0 1]
end
Epos = zeros(size(E));
I = find(E>0);
if isempty(I),
   Enormpos = [];
else
   Epos(I) = E(I);
   area = trapz(Xnorm,Epos);
   Enormpos = Epos / area;
end
Eneg = zeros(size(E));
I = find(E<0);
if isempty(I),
   Enormneg = [];
else
   Eneg(I) = -E(I);
   area = trapz(Xnorm,Eneg);
   Enormneg = Eneg / area;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
% Continuous entropy estimation
% lower bound of CE set to -20, which supports 2^20 samples
%
if isempty(Enormpos),
   cepos = 0;
else
   info(find(Enormpos==0)) = 0;
   I = find(Enormpos);
   info(I) = Enormpos(I).*log2(Enormpos(I));
   cepos = trapz(Xnorm, -info);
   % normalization from [-20 0] to [0 1]
   cepos = 1 - cepos;
   cepos = log(cepos) / log(21); % 21 comes from 20 + 1
   cepos = 1 - cepos;
end
if isempty(Enormneg),
   ceneg = 0;
else
   info(find(Enormneg==0)) = 0;
   I = find(Enormneg);
   info(I) = Enormneg(I).*log2(Enormneg(I));
   ceneg = trapz(Xnorm, -info);
   % normalization from [-20 0] to [0 1]
   ceneg = 1 - ceneg;
   ceneg = log(ceneg) / log(21); % 21 comes from 20 + 1
   ceneg = 1 - ceneg;
end

cost = (MSE+cepos+ceneg) / 3;

return;
```