LEARNING WITH FEATURE PARTITIONS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

İzzet Şirin

August, 1993

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Halil Altay Güvenir (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Varol Akman

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Kemal Oflazer

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of the Institute

ABSTRACT

LEARNING WITH FEATURE PARTITIONS

İzzet Şirin
M.S. in Computer Engineering and Information Science
Advisor: Asst. Prof. Halil Altay Güvenir
August, 1993

This thesis presents a new methodology of learning from examples, based on *feature partitioning*. *Classification by Feature Partitioning* (CFP) is a particular implementation of this technique, which is an inductive, incremental, and supervised learning method. Learning in CFP is accomplished by storing the objects separately in each feature dimension as disjoint partitions of values. A partition, a basic unit of representation which is initially a point in the feature dimension, is expanded through generalization. The CFP algorithm specializes a partition by subdividing it into two subpartitions. Theoretical (with respect to PAC–model) and empirical evaluation of the CFP is presented and compared with some other similar techniques.

Keywords: Machine learning, inductive learning, incremental learning, supervised learning, feature partitioning.

ÖZET

ÖZNİTELİK BÖLÜNTÜLERİ İLE ÖĞRENME

İzzet Şirin
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Danışman: Y. Doç. Dr. Halil Altay Güvenir
Ağustos, 1993

Bu çalışmada öznitelik bölünmesine dayalı yeni bir mekanik öğrenme yöntemi sunulmuştur. Bu yöntem kullanılarak bir sınıflama algoritması olan *Öznitelik Bölüntüleri ile Sınıflayıcı* CFP'nin yazılımı hazırlanmıstır. CFP algoritması mekanik öğrenmeyi tümevarım ve artırımlı öğrenme yöntemlerini kullanarak sağlar. CFP algoritmasında bölütü elemanları temel gösterim unsurlarıdır. Başlangıçta bölüntü elemanları bir boyutlu uzayda bir noktayı ifade ederken, zaman içinde bu elemanlar genişleyerek bir aralığı ifade ederler. Bölüntü elemanları parçalanarak özelleştirilirler. CFP algoritmasının kuramsal analizi *yaklaşık olarak doğru* kuramına (PAC-model) göre yapılmıştır ve benzer sistemlerle uygulama sonuçları karşılaştırılmıştır.

Anahtar Sözcükler: Mekanik öğrenme, tümevarımsal öğrenme, artırımsal öğrenme, denetimli öğrenme, öznitelik bölüntüleme.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisor Asst. Prof. Halil Altay Güvenir for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis.

I would also like to thank Assoc. Prof. Varol Akman and Asst. Prof. Kemal Oflazer for reading and commenting on the thesis.

I owe special thanks to Prof. Mehmet Baray for providing a pleasant environment for study.

I am grateful to the members of my family for their infinite moral support and patience that they have shown, particularly in times I was not with them.

Contents

List of Figures

List of Tables

List of Symbols

AI	Artificial Intelligence.
CFP	Classification by feature partitioning.
C_i	Label of the ith class.
C4	Decision tree algorithm.
C4.5	Decision tree algorithm.
$C(n,r)$	Number of combinations of r object out of n.
CT	Confidence threshold parameter of the CFP.
D	Probability distribution defined on the instance space.
D_f	Generalization limit for feature f in CFP.
3–DNF	Third disjunctive normal form.
E	An example.
e	An example.
e_{class}	Class of example e.
e_f	Feature value of example e.
E^n	Euclidean n–dimensional space.
EACH	Exemplar-aided constructor of hyperrectangles.
EBG	Explanation-based generalization.
EBL	Explanation-based learning.
GA	Genetic algorithm.
GA-CFP	Hybrid CFP algorithm.
(GA)-CFP	GA-CFP and CFP algorithms.
GA-WKNN	Hybrid WKNN algorithm.
h	Hypothesis.
H	Hyperrectangle.
$H_{k_{lower}}$	Lower boundary of H_k.
$H_{k_{upper}}$	Upper boundary of H_k.

IBL	Instance-based learning.
ID3	Decision tree algorithm.
KA	Knowledge acquisition.
KBS	Knowledge-based Systems.
KNN	K Nearest neighbors algorithm.
ln	Natural logarithm.
log	Logarithm in base two.
M, m	The number of training examples.
ML	Machine learning.
NGE	Nested generalized exemplars.
PAC	Probably approximately correct.
PLS1	Probabilistic learning algorithm.
$vote_c$	Voting power of class c.
WKNN	Weighted K nearest neighbors algorithm.
w_f	Weight of feature f.
w_{H_k}	Weight of hyperrectangle k.
Δ	Weight adjustment rate of the CFP algorithm.
δ	Confidence parameter of the PAC-model.
ε	Error parameter of the PAC-model.
γ	Probability of distance between to adjacent point is greater than ε.
$\|\|$	Absolute value.
{}	Comment in the algorithm of CFP.
$[x, y]$	Closed interval in which boundaries are x and y.
$\lceil x \rceil$	The smallest integer number greater than or equal to x.
$\times$	Multiplication.
/	Division.

Chapter 1

Introduction

The development of knowledge-based systems (KBS) is a difficult and often time consuming task. The acquisition of the knowledge necessary to perform a certain task (usually through a series of acquisition sessions with a domain expert) is considered as one of the main bottlenecks in building KBS. Knowledge acquisition (KA) and machine learning (ML) have been closely linked by their common application field, namely building up knowledge bases for KBS. Learning and KA can be viewed as two processes that construct a model of a task domain, including the systematic patterns of interaction of an agent situated in a task environment. Learning of an agent involves both learning to solve new problems and learning better ways to solve previously solved problems. Carbonell describes machine learning as follows [6]:

> Perhaps the tenacity of ML researchers in light of the undisputed difficulty of their ultimate objectives, and in light of early disappointments, is best explained by the very nature of the learning process. The ability to learn, to adapt, to modify behavior is an inalienable component of human intelligence.

The motivation for applying ML techniques to *real-world* tasks is strong. ML offers a technology for assisting in the KA process. There is a potential for automatically discovering new knowledge in the available on-line databases which are too large for humans to manually sift through. Furthermore, the ability of computers to automatically adapt to changing expertise would offer huge benefits for the maintenance and evolution of expert systems.

The success of a learning system is highly related to the ability to cope with noisy and incomplete data, an adequate knowledge representation scheme, having low learning and sample complexities, and the effectiveness of the learned knowledge [24].

Tradeoffs between learning and programming can also be examined in terms of their relative utilities. Computer time is now very cheap, whereas human labor is becoming increasingly expensive. This suggests that learning could have an immediate economic edge over manual methods of programming the same information. Of course, there are many situations in which the potential benefit of developing a knowledge base far exceeds the cost of its capture.

Evaluating the utility of programmed and learned knowledge cannot stop with a simple human level of effort analysis. However, computational efficiency of algorithms and the representations they use dramatically affect both the size of the computers that are required and the size of the problems one can solve.

The most widely studied method for symbolic learning is one of inducing a general concept description from a sequence of instances of the concept and (usually) known counterexamples of the concept. The task is to build a concept description from which all previous positive instances can be rederived by universal instantiation but none of the previous negative instance (counterexample) can be rederived by the same process [6].

Learning from examples has been one of the primary paradigms of ML research since the early days of Artificial Intelligence (AI). Many researchers have observed and documented the fact that human problem solving performance improves with experience. In some domains, the principal source of expertise seems to be a memory to hold a large number of important examples. For example, in chess human experts seem to have a memory of roughly 50,000 to 70,000 specific patterns. The attempts to build an intelligent (i.e., at the level of human) system have often faced the problem of memory for too many specific patterns. Researchers expect to solve this difficulty by building machines that can learn. This reasoning has motivated many machine learning projects [31].

Inductive learning is a process of acquiring knowledge by drawing inductive inference from teacher (or environment) provided facts. Such a process involves operations of generalizing, specializing, transforming, correcting and refining knowledge representations. There are several different methods by which a

human (or machine) can acquire knowledge, such as rote learning (or learning by being programmed), learning from instruction (or learning by being told), learning from teacher provided examples (concept acquisition), and learning by observing the environment and making discoveries (learning from observation and discovery).

Learning a concept usually means to learn its description, i.e., a relation between the name of the concept and a given set of features. Several different representation techniques have been used to describe concepts for supervised learning tasks. One of the widely used representation technique is the *exemplar-based* representation. The representation of the concepts learned by the exemplar-based learning techniques stores only specific examples that are representatives of other several similar instances. Exemplar-based learning was originally proposed as a model of human learning by Medin and Schaffer [25].

There are many different exemplar-based learning models in the literature (see Fig. 1.1). All of these models share the property that they use verbatim examples as the basis of learning. For example, instance-based learning [1] retains examples in memory as points, and never changes them. The only decisions to be made are what points to store and how to measure similarity. Aha, Kibler, and Albert [1] have created several variants of this model, and they are experimenting with how far they can go with strict point-storage model. Another example is the nested-generalized exemplars model of Salzberg [34]. This model changes the point storage model of the instance-based learning and retains examples in the memory as axis-parallel hyperrectangles.

Previous implementation of the exemplar-based models usually extend the nearest neighbor algorithm in which some kind of similarity (or distance) metric is used for prediction. Hence, prediction complexity of such algorithms is proportional to the number of instances (or objects) stored.

This thesis presents another form of exemplar-based learning, based on the representation of *feature partitioning*. CFP is a particular implementation of the feature partitioning technique. The CFP partitions each feature into segments corresponding to concepts. Therefore, the concept description learned by the CFP is a collection of feature partitions. In other words, the CFP learns a projection of the concept on each feature dimension. The CFP algorithm makes several significant improvements over other exemplar-based learning algorithms, where the examples are stored in memory without any change in

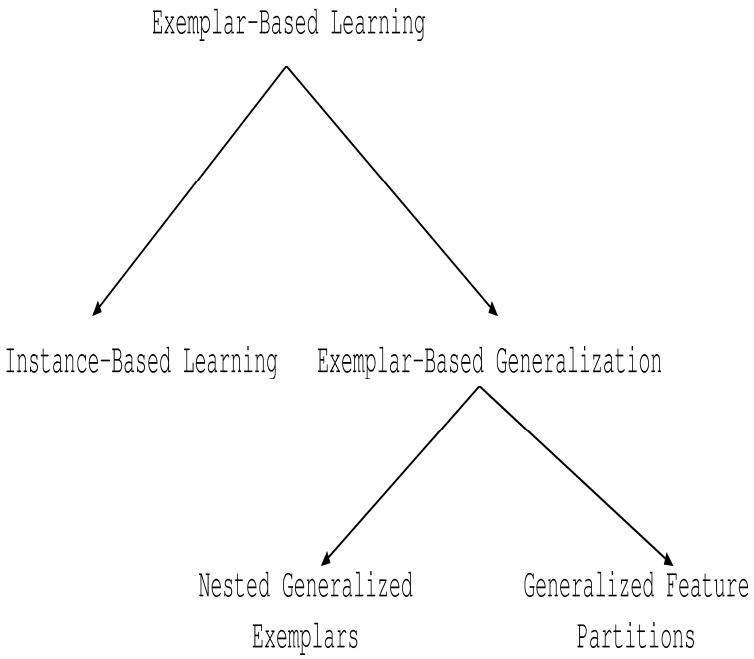

Figure 1.1. Classification of exemplar-based learning algorithms

the representation. For example, IBL algorithms learn a set of instances (a representative subset of all training examples), EACH (Exemplar-Aided Constructor of Hyperrectangles) learns a set of hyperrectangles of the examples. On the other hand, the CFP algorithm stores the instances as factored out by their feature values.

Since the CFP learns projections of the concepts, it does not use any similarity (or distance) metric for prediction. Each feature contributes the prediction process by its local knowledge. Final prediction is based on a voting among the predictions of the features. Since a feature partition can be represented by a sorted list of line segments, the prediction by a feature is simply a search for the partition corresponding to the instance on that sorted list. Therefore, the CFP algorithm significantly reduces the prediction complexity, over other exemplar-based techniques. The power of a feature in the voting process is determined by the weight of that feature. Assigning variable weights to the features enables the CFP to determine the importance of each feature to reflect its relevance for classification. This scheme allows smooth performance degradation when data set contains irrelevant features.

The issue of unknown attribute values is an unfortunate fact of real-world data sets, that data often contain missing attribute values. Most of the learning

systems, usually overcome this problem by either filling in missing attribute values (with most probable value or a value determined by exploiting inter-relationships among the values of different attributes) or by looking at the probability distribution of known values of that attribute. Most common approaches are compared in Quinlan [29], leading to a general conclusion that some approaches are clearly inferior but no one approach is uniformly superior to others. In contrast, CFP solves this problem very naturally. Since CFP treats each attribute value separately, in the case of an unknown attribute value, it simply leaves the partitioning of that feature intact. That is the unknown values of an instance are ignored while only the known values are used.

In the next chapter some of the previous models are presented and several different properties of the learning method are explored. The precise details of the CFP algorithm are described in Chapter 3. The process of partitioning of a feature dimension is illustrated with an example, and several extensions to the CFP algorithm are described. Chapter 4 presents theoretical and empirical evaluation of the CFP algorithm. A theoretical analysis of the CFP algorithm with respect to PAC-learning theory is presented, Performance of the CFP on artificially generated data sets and comparisons with other similar techniques on real-world data sets are also presented. The final chapter discusses the applicability of the CFP and concludes with a general evaluation of the algorithm.

Chapter 2

Previous Models

It is well known that two major directions of AI research, *symbolic* and *subsymbolic* models, exhibit their strengths and weakness in an almost complementary ways. While symbolic models are good in high-level reasoning, they are weak in handling imprecise and uncertain knowledge and data. On the other hand subsymbolic models are good in lower-level reasoning such as imprecise classification and recognition problems. However, they are not good in higher-level reasoning. Nevertheless, both models contribute important insight to our understanding of intelligent systems. Fortunately, over the last few years these two approaches have become less separate, and there has been an increasing amount of research that can be considered a hybrid of the two approaches [4, 13, 37].

2.1 Symbolic Models

Two of the main types of learning from examples in the history of AI research are *concept learning* and *explanation-based learning*.

Concept Learning: Concept learning tackles the problem of learning concept *definitions*. A definition is usually a formal description in terms of a set of attribute-value pairs, often called *features*. More recently, approaches using decision trees, connectionist architectures, representative instances, and hyperrectangles (exemplar-based learning) have appeared in the literature. These approaches construct concept description by examining a series of examples,

each of which is categorized as either an example of the concept or a counterexample. A learning system can refine its concept description until it matches the correct description.

Exemplar-based learning is a kind of concept learning in which the concept definition is constructed from the examples themselves, using the same representation language. In the instance-based learning in fact, the examples are the concept (no generalizations are formed). Like other concept learning theories, exemplar-based learning requires little or no domain specific knowledge.

2.1.1 Explanation-Based Learning (EBL)

Dejong and Mooney [9] present a general technique for learning by generalizing explanations. They discuss the differences of their model with the *Explanation-Based Generalization* (EBG) [21]. The border term EBL better describes the approach than does EBG. It seems both possible and desirable to apply the approach to concept refinement (i.e., specialization) as well as concept generalization. The explanation-based approach uses domain-specific knowledge as much as possible. This approach has been more commonly used in ML. An explanation consists of an inference chain (which may be a proof, but may also be merely plausible reasoning) that identifies one or more of these variables as the cause of the wrong prediction. This subset of variables is linked by the inference chain to correct the prediction. Once the variables have been identified, the system must revise its prediction model in some way to reflect the fact that a particular set of variables is now associated with the new prediction.

The difficult part of the EBL is the identification of the proper set of the variables that cause the wrong prediction. The number of possible subsets of the variables increases exponentially with the number of variables, so this is intractable. Hence, EBL systems use domain-specific knowledge to select, from a large set of possible explanations, a few plausible explanations. The domain knowledge is used to construct a proof that specific variables caused the wrong prediction. The knowledge that must be provided includes detailed knowledge about how each variable affect the prediction, and how input variables interact. Another goal of the EBL methods is that they attempt to construct as concise a description as possible of the input examples.

The generalization method described in Mitchell [21] EBG, must be provided with the following information:

1. *Goal concept*: A definition of the concept to be learned in terms of high-level or functional properties which are not directly available in the representation of an example.
2. *Training example*: A representation of a specific example of the concept in terms of lower level features.
3. *Domain theory*: A set of inference rules and facts sufficient for proving that a training example meet the high-level definition of the concept.
4. *Operationality criterion*: A specification of how a definition of a concept must be represented so that the concept can be efficiently recognized.

Given this information, EBG constructs an explanation of why the training example satisfies the goal concept by using the inference rules in the domain theory. This explanation takes the form of a proof tree composed of Horn-clause inference rules which proves that the training example is a member of the concept.

2.1.2 Instance-Based Learning

The primary output of IBL algorithms is a *concept description*, which is a function that maps instances to concepts. Instance-based learning technique [1], has been implemented in three different algorithms, namely IB1, IB2, and IB3. IB1 stores all the training instances, IB2 stores only the instances for which the prediction was wrong. Neither IB1 nor IB2 remove any instance from concept description after it had been stored. IB3 employs a significance test (i.e., acceptable or significantly poor) to determine which instances are good classifiers and which ones are believed to be noisy.

An instance-based concept description includes a set of stored instances and some information concerning their past performance during the training process, e.g., the number of correct and incorrect classification predictions. The final set of instances can change after each training process. However, IBL algorithms do not construct intensional concept descriptions Instead, concept

descriptions are determined by how IBL algorithm's *similarity* and *classification* functions use the current set of saved instances. The similarity and classification functions determine how the set of saved instances in the concept description are used to predict values for the category attribute. Therefore, IBL concept descriptions contain these two functions along with the set of stored instances.

Three components of IBL algorithms are:

1. *Similarity function*: computes the similarity between an instance and instances in concept description.
2. *Classification function*: yields the classification for an instance by using the result of the similarity function and performance record of the concept description.
3. *Concept description updater*: maintains records on classification performance and decides which instances should be included in the concept description.

IBL algorithms assume that, instances that have high similarity values according to the similarity function, have similar classifications. This leads to their local bias for classifying novel instances according to their most similar neighbor's classification. They also assume that, without prior knowledge, attributes will have equal relevance for classification decisions (i.e. each feature has equal weight in similarity function). This assumption may lead to significant performance degradation if the data set contains many irrelevant features.

IB3 is the noise tolerant version of the IBL algorithms. It employs *wait and see* evidence gathering method to determine which of the saved instances are expected to perform well during classification. In all IBL algorithms, the similarity between instances x and y is computed as:

$$similarity(x, y) = -\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

IB3 maintains a classification record (i.e. number of correct and incorrect classification attempts) with each saved instance. A classification record summarizes an instances's classification performance on subsequently presented

training instances and suggests how it will perform in the future. IB3 employs a significance test (i.e. acceptable and significantly poor) to determine which instances are good classifiers and which ones are believed to be noisy. IB3 accepts an instance if its classification accuracy is significantly greater than its class's observed frequency and removes an instance from concept description if its accuracy is significantly less. Confidence intervals are used to determine whether an instance is acceptable, mediocre, or noisy. Confidence intervals are constructed around both the current classification accuracy of the instance and current observed relative frequency of its class.

2.1.3 Nested Generalized Exemplars

In *Nested Generalized Exemplars* (NGE) theory, learning is accomplished by storing objects in Euclidean n-space, E^n, as hyperrectangles [34]. NGE is also a variation of exemplar-based learning. In the simplest form of the exemplar-based learning, every example is stored in memory, with no change in representation (or without generalization), as in IB1 algorithm presented above. NGE adds generalization on top of the simple exemplar-based learning. It adopts the position that exemplars, once stored, should be generalized. The learner compares a new example to those it has seen before and finds the most similar, according to a similarity metric, which is inversely related to the distance metric (Euclidean distance in n-space). The term *exemplar* (or *hyperrectangle*) is used to denote an example stored in memory. Over time, exemplars may be modified (due to generalization) from their original forms. This is similar to the generalizations of partitions in the CFP algorithm.

Once a theory moves from a symbolic space to Euclidean space, it becomes possible to *nest* one generalization inside the other. Its generalizations, which take the form of hyperrectangles in E^n, can be nested to an arbitrary depth, where inner rectangles act as *exceptions* to the outer ones.

EACH (Exemplar-Aided Constructor of Hyperrectangles) is a particular implementation of the NGE technique [34], where an exemplar is represented by a hyperrectangle. EACH uses numeric slots for feature values of exemplar. The generalizations in EACH take the form of *hyperrectangles in Euclidean n-space*, where the space is defined by the feature values for each example. Therefore, the generalization process simply replaces the slot values with more general values (i.e., replacing the range of values [a, b] with another range [c,

$d]$, where $c \leq a$ and $d \geq b$). EACH compares the class of a new example with the most similar (shortest distance) exemplar in the memory. The distance between an example and an exemplar is computed according to the following distance function:

$$Distance(E, H_k) = w_{H_k} \sqrt{\sum_{i=1}^{n} (w_{f_i} \frac{dif_i}{max_i - min_i})^2}$$

where

H_k:	hyperrectangle k
$H_k.reference$:	the number of reference to H_k
$H_k.correct$:	the number of correct prediction made by H_k
$H_{k_{lower}}$:	lower boundary of H_k
$H_{k_{upper}}$:	upper boundary of H_k
E:	an example
f_i:	ith feature
E_{f_i}:	ith feature value of exampleE
w_{H_k}:	weight of $H_k(H_k.reference/H_k.correct)$
w_{f_i}:	weight of f_i
max_i, min_i:	maximum and minimum feature values, respectively
dif_i:	the distance between E and H on the ith dimension

$$dif_i = \begin{cases} E_{f_i} - H_{k_{upper}} & \text{if } E_{f_i} > H_{k_{upper}} \\ H_{k_{lower}} - E_{f_i} & \text{if } E_{f_i} < H_{k_{lower}} \\ 0 & \text{otherwise.} \end{cases}$$

If a training example and the nearest exemplar are the same (i.e. a correct prediction has been made) the exemplar is generalized to include the new example if it is not already contained in the exemplar. However, if the closest example has a different class then that of the example, then the second closest exemplar is tried in the similar way. The idea behind the second minimum is to apply the *second chance* heuristic. This heuristic is useful to reduce the number of exemplars in the memory. If none the closest two exemplars has the same class as the example, then the algorithm modifies the weights of features so that the weights of the features that caused the wrong prediction is increased (in terms of distance).

2.1.4 K Nearest Neighbors (KNN) Algorithm

The nearest neighbor classification algorithm is based on the idea that, given a data set of classified examples, an unclassified instance should belong to the same class as its nearest neighbors in the data set. A common extension to the nearest neighbor algorithm is to classify a new instance by looking at its k *nearest neighbors* ($k > 1$). The k nearest neighbors algorithm classifies a new instance by noting its distance from each of the stored instances and assigning the new instance to the class of the majority of its nearest neighbors. Different ways of computing similarity or distances between instances are compared by Salzberg [33]. This algorithm can be quite effective when the attributes of the data are equally important. However, if the attributes are not equally important, performance of the algorithm degrades. Usually to solve this problem feature weights are introduced, resulting in the WKNN algorithm [22]. Assigning variable weights to the attributes of the instances before applying the KNN algorithm distorts the feature space, modifying the importance of each attribute to reflect its relevance for classification. In this way, closeness or similarity with respect to important attributes becomes more critical than similarity with respect to irrelevant attributes.

The GA-WKNN algorithm [22] combines the optimization capabilities of a genetic algorithm with the classification capabilities of the WKNN algorithm. The goal of the GA-WKNN algorithm is to learn an attribute weight vector which improves the WKNN classification performance. Chromosomes are vectors of real-valued weights. A vector value is associated with each attribute and one is associated with each of the k neighbors. Thus the length of the chromosome is the number of features plus k. Another extension to the KNN is to combine simple KNN with genetic algorithms (GAs)

2.1.5 Decision Tree

The decision tree is a well-known representation for classification tasks. This representation has been used in a variety of systems. Among them the most famous are ID3 [27] and its extension C4.5 [29] of Quinlan.

A decision tree can be used to classify a case by starting at the root of the tree and moving through it until a leaf is encountered. At each non-leaf

decision node, the outcome of the case for the test at the node is determined and attention shifts to the root of the subtree corresponding to this outcome. When this process finally leads to a leaf, the class of the case is predicted to be that record at the leaf.

A decision tree is global for each attributes, in other words each non-leaf decision node may specify some test on any one of the attributes. The CFP algorithm can be seen to produce special kind of decision trees. Unlike ID3, the CFP probes each feature exactly once. An important difference between decision tree approach and other approaches mentioned above, including CFP, is that the classification performance of these systems does not depend critically on any small part of the model. In contrast, decision trees are much more susceptible to small alterations.

2.1.6 The PLS1 Algorithm

Both ID3 [27] and *probabilistic learning system* (PLS1) [30, 31] use probabilistic criteria to specialize hypotheses, and start with a single general description and split into two or more parts. The process of splitting continues, using one attribute for each split, until some stopping criterion is satisfied.

The PLS1 accepts instances of known class membership, and based on their frequency, divides the instance space into mutually exclusive regions or *probability classes.* Like ID3, PLS1 also uses specialization. It represents input instances as points in a k-dimensional space creates orthogonal hyperrectangles by inserting boundaries parallel to instance space axes. Each hyperrectangle r is annotated with values: (1) the probability u of finding a positive example within r, and (2) an error measure e of u. These annotated hyperrectangles (r, u, e), called *regions*, are like nodes of a decision tree annotated with probability and error measures. PLS1 was compared with ID3 and C4 by Rendell [31, 32]. These comparisons show that two algorithms are similar, although they differ in some striking ways. For example, the splitting criteria are different. C4 prunes its decision tree to eliminate noise, whereas PLS1 faces that problem by splitting only if the statistical significance is high (and never prunes).

2.2 Subsymbolic Models

Both symbolic and subsymbolic models contribute an important insight to our understanding of intelligent systems. Connectionist models and Genetic Algorithms (GAs) are the best known examples of subsymbolic computation. This section presents a brief description of these subsymbolic models. I will not attempt to define precisely the essential differences between symbolic and subsymbolic approaches. This is beyond the scope of this thesis.

2.2.1 Connectionist Paradigm

A neural network (or connectionism) is a kind of computation system in which the state of a system is represented as a numerical distribution pattern with many processing units and connections among those units. Learning by neural networks uses an algorithm for transforming distribution patterns, which quite different from learning based on symbolic representations. Connectionist systems have stirred a great deal of excitement for number of reasons.

1. They are novel. Connectionism seems to be a good candidate for a major new paradigm in AI where there have only been a handful of paradigms.
2. They have cognitive science potential. While connectionist neural nets are not accurate models of neurons, they do seem brain-like and capable of modeling a substantial range of cognitive phenomena.
3. Connectionist systems have exhibited non-trivial learning. They are able to self-organize, given only examples as inputs.
4. Connectionist systems can be made fault-tolerant and error-correcting, degrading gracefully for cases not encountered previously.
5. An appropriate and scalable connectionist hardware is rapidly becoming available. This is important for applicability of the connectionist systems to the large-scale cognitive phenomena.
6. Connectionist architectures also scale well, in that modules can be interconnected rather easily. This is because message passed between modules are generally activation level, not symbolic messages.

However, there are considerable difficulties still ahead for connectionist models. Many different connectionist architectures were proposed in the literature. It is important to note that there are connectionist architectures beyond the simple feed-forward, single-hidden-layer neural networks. In particular, recurrent [10] with their feedback loops and "memory", are especially appealing for application to symbolic tasks that have a sequential nature.

2.2.2 Genetic Algorithms

Genetic algorithms (GAs) are adaptive *generate-and-test* procedures derived from principles of natural population genetics. This section presents a high-level description of one formulation of genetic algorithms. GAs represents a class of adaptive search techniques that have been intensively studied in recent years. The key feature of GAs is that adaptation proceeds, not by making incremental changes to a single *structure* but by maintaining a *population* (or database) of structures from which each structure in the population has an associated *fitness* (goal-oriented evaluation). These fitness values are used in *competition* to determine which structures are used to form new ones [19]. Genetic algorithms are best viewed as another tool for the designer of learning systems. The selection of a good feedback mechanism that facilitates the adaptive search strategy, is critical issue for the effectiveness of GAs. Detailed descriptions, of genetic algorithms, are given by Goldberg [14]. A skeleton of a simple genetic algorithm is shown in Fig. 2.1.

During iteration (*generation*) t, the genetic algorithm maintains a *population* $P(t)$ of structures $\{x_1^t, x_2^t, ..., x_N^t\}$ chosen from the domain of the objective function f. The initial population $P(0)$ is usually chosen at random. The population size N remains fixed for the duration of the search. Each structure x_i^t is *evaluated* by computing $f(x_i^t)$. Usually, the term *trial* is used for each such evaluation. This provides a measure of fitness of the evaluated structure for the given problem. When each structure in the population has been evaluated, a new population of structures is formed in two steps.

First, structures in the current population are selected to reproduce on the basis of their relative fitness. That is, the *selection* algorithm chooses structures for replication by stochastic procedure that ensures that the expected number of offspring associated with a given structure x_i^t if $f(x_i^t/\mu(P,t)$, where $f(x_i^t)$

```
Genetic-Algorithm:
begin
        t = 0
        initialize P(t)
        evaluate P(t)
        while( not terminating condition)
                t = t + 1
                select P(t) from P(t − 1)
                recombine P(t)
                evaluate P(t)
        end
end
```

Figure 2.1. A skeleton of a simple genetic algorithm

is the observed performance of x_i^t and $\mu(P,t)$ is the average performance of all structures in the population. Structures that perform well may be chosen several times for replication and structures that perform poorly may not be chosen at all. In the absence of any other mechanisms, this selective pressure would cause the best performing structures in the initial population, to occupy a larger portion of the population over time.

In the second step, the selected structures are recombined using idealized *genetic operators* to form a new population. The most important genetic operator is *crossover*, which combines the features of two *parent* structures to form two similar *offsprings*. Crossover operates by swapping corresponding segments of the structures representing the parents. In generating new structures for testing, the crossover operator draws only on the information present in the structures of the current population. If specific information is missing, due to storage limitations or loss incurred during the selection process of a previous generation, then crossover operator is unable to produce new structures that contain that information. A *mutation* operator arbitrarily alters one or more components of a selected structure. It provides a means for introducing new information into the population. Usually, mutation operator is treated as a background operator (i.e. its probability of application is kept very low). Its presence ensures that all points in the search space can be reached.

2.3 Comparison of CFP with Other Models

To characterize the feature partitioning method, I will identify several different properties of the learning methods, and show how CFP differs or is similar to the other methods. We will use the terms *instance* and *example* interchangeably.

Knowledge Representation Schemes: One of the most useful and interesting dimensions in classifying ML methods is the way they represent the knowledge they acquire. Many systems acquire *rules* during their learning process. These rules are often expressed in logical form (i.e., Prolog), but also in other forms, such as schemata. Typically such systems will try to generalize the left hand side of the rules (the antecedent in an *if-then* rule) so that those rules apply to as large number of situations as possible. Some systems try to generalize right hand side of the rules. Another way to represent what is learned is with decision trees. For example, ID3 [27], and several successors. Decision trees seem to lack of clarity as representations of knowledge. Another knowledge representation is set of representative instances [1] or hyperrectangles [30, 34]. On the other hand, in CFP algorithm partition is a basic unit of representation. Learning in CFP is accomplished by storing objects separately in each feature dimension as partitions of the set of values that it can take.

Underlining Learning Strategies: Most systems fall into one of two main categories according to their learning strategies. Namely, *incremental* and *non-incremental* learning strategies. Systems that employs incremental learning strategy attempt to improve an internal model (whatever the representation is) with each example they process. However, in non-incremental strategies, system must see all the training examples before constructing a model of the domain. Most concept learning systems follow an incremental learning strategy, since the idea is to begin with a rough definition of a concept, and modify that definition over time. The characteristic problem of these system is that their performance is sensitive to the order of the instances they process. The CFP algorithm falls in to the incremental learning category, which means that CFP's behavior sensitive to the order of examples.

A non-incremental learning strategy usually assumes random access to the examples in the training set. The learning systems which follows this strategy (including ID3 of Quinlan and INDUCE system of Larson) search for patterns and regularities in the training set in order to formulate decision trees or rules.

This approach offers the advantage of not being sensitive to the order of the examples. However, it introduces additional complexity by requiring the program to decide when it should stop its analysis.

Domain Independent Learning: EBG requires considerable amounts of domain specific knowledge to construct explanations. This results from the fact that explanation based systems must construct explanations each time they experience a prediction failure.

Exemplar-based learning, on the other hand, does not construct explanations at all. Instead, it incorporates new examples into its experience by storing them verbatim in memory. Since it does not convert examples into another representational form, it does not need any domain knowledge to explain what conversions are legal, or even what the representations mean. Interpretation is left to the user or domain experts. Consequently, exemplar-based systems like CFP, can be quickly adapted to new domains, with a minimal amount of programming.

Multiple Concept Learning: Machine learning methods have gradually increased the number of concepts that they can learn and the number of variables they could process. Many early programs could learn exactly one concept (e.g. initial ID3 can learn only one concept (positive and negative). Successors of the ID3 can learn multiple concepts). Some of the theories that handle multiple concepts need to be told exactly how many concepts they are learning.

Binary, discrete, and continuous variables: One shortcoming of some learning programs is that they handle only binary variables, or only continuous variables, but not both. In Catlett [7] a method of changing continuous-valued attributes into ordered discrete attributes is presented for the systems that can only use discrete attributes. The CFP learning program handles variables which take on any number of values, from two (binary) to infinity (continuous). However, if most of the features are binary or discrete, the probability of constructing overlapping partitions is high. In this case performance of the CFP depends on the observed frequency of the concepts. In general, CFP learning system outperforms if the domain has continuous variables.

Problem Domain Characteristic: In addition to characterizing the dimensions along which CFP system offers advantages over other methods, it's worthwhile to consider the sorts of problem domains it may or may not handle. The CFP system is domain independent. However, there are some domains in which

the target concepts are very difficult for exemplar-based learning, and other learning techniques will perform better. In general, exemplar-based learning is best suited for domains in which the exemplars are clusters in feature space. The CFP algorithm is applicable to concepts, where each feature, independent of other features, can be used in the classification of the concept. The CFP algorithm is a variant of algorithms that learn by projecting into one feature dimension at a time. For example, ID3 learns that in greedy manner while building conjunctions. The CFP algorithm retains feature-by-feature representation and uses voting to categorize. if concept boundaries are nonrectangular, or projection of the concepts into a feature dimension are overlapping then CFP performance degrades.

Noise Tolerance: The ability to form general concept description on the basis of particular examples is an essential ingredient of intelligent behavior. If examples contain errors, the task of useful generalization becomes harder. The cause of these errors or "noise", may be either systematic or random. There two sorts of noise: (1) *classification noise*, and (2) *attribute noise* [3]. Classification noise involves corruption of the class value of an instance, and attributes noise involves replacing of the attribute value of an instance. Missing attribute values are also treated as attribute noise.

Applicability of a learning algorithm highly depends on the capability of the algorithm handling noisy instances. Therefore, most of the learning algorithms try to cope with noisy data. For example, the IB3 algorithm utilizes classification performance of stored instances to cope with noisy data. It removes instances from concept description that are believed to be noisy [1]. EACH also uses classification performance of hyperrectangles. However, it does not remove any hyperrectangle from concept description [34]. Decision tree algorithms utilize statistical measurements and tree pruning to cope with noisy data [27, 29].

Connectionist models and Genetic Algorithms (GAs) are relatively noise tolerant. Robustness of connectionist models naturally arises from distributed representation. The learned concept is represented as a set of weighted connections between neuron-like units. The key feature of GAs is that adaptation proceeds with population of structures. The selection of a good feedback mechanism is a critical issue for the effectiveness of GAs [14].

The CFP algorithm utilizes representativeness values of partitions, observed

frequency of concepts, and a voting scheme to cope with noisy data. Section 3.2 presents noise tolerant version of the CFP algorithm which removes partitions that are believed to be noisy. Section 4.2.2 presents empirical comparisons of the CFP with C4.5 on artificially generated noisy data.

Chapter 3

Learning with Feature Partitions

This chapter discuss a new incremental learning technique, based on *feature partitioning*. First section describes the CFP, that can be used on real-world data sets. The process of partitioning of a feature dimension is illustrated with an example. Section 2 presents an extension to the CFP for handling noisy instances. Section 3 explores a possible parallelization scheme of the CFP and Section 4 presents a hybrid system (GA-CFP) that combines the optimization capability of GAs and the classification capability of the CFP. Finally, limitations of the CFP presented in Section 5.

3.1 The CFP Algorithm

This section describes the details of the feature partitioning algorithm used by CFP. Learning in CFP is accomplished by storing the objects separately in each feature dimension as disjoint partitions of values. A partition is the basic unit of representation in the CFP algorithm. For each partition, lower and upper bounds of the feature values, the associated class, and the number of instances it represents are maintained. The CFP program learns the projection of the concepts over each feature dimension. In other words, the CFP learns partitions of the set of possible values for each feature. An example is defined as a vector of features values plus a label that represents the class of the example.

Initially, a partition is a point (lower and upper limits are equal) on the line representing the feature dimension [38]. For instance, suppose that the first example e_1 of class C_1 is given during the training phase (Fig. 3.1.a). If the value

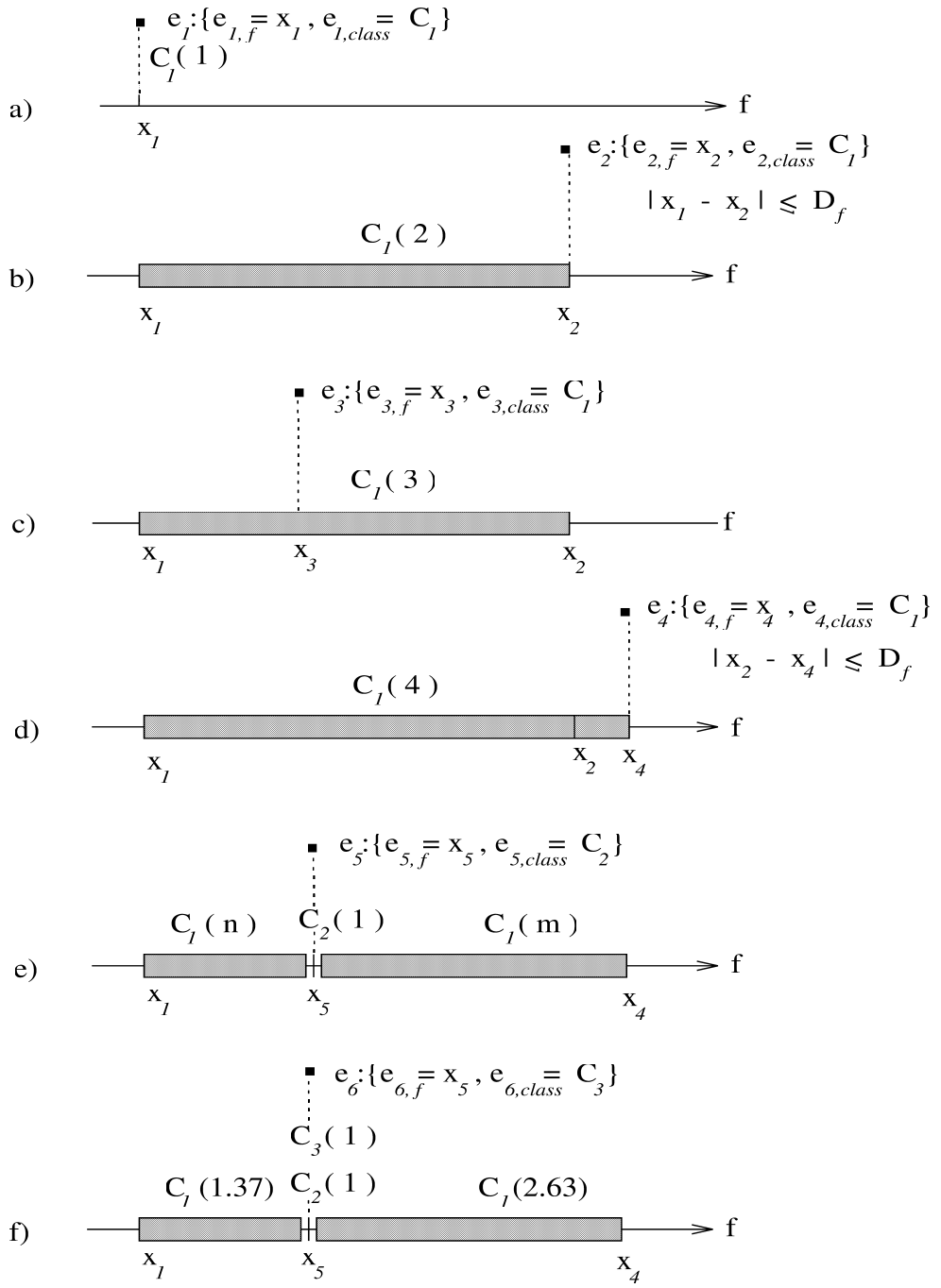

Figure 3.1. Partitioning of a feature dimension

of e_1 for feature f is x_1, that is $e_{1,f} = x_1$ then the set of possible values for feature f will be partitioned into three partitions: $< [-\infty, x_1], undetermined, 0 >$, $< [x_1, x_1], C_1, 1 >$, $< [x_1, \infty], undetermined, 0 >$; where the first element of the triple indicates the range of the partition, the second its class, and the third, called the representativeness value, the number of examples represented by the partition.

A partition can be extended through generalization with other neighboring points in the same feature dimension. The CFP algorithm pays attention to the disjointness of the partitions to avoid *over-generalization.* In order to generalize a partition in feature f to cover a point, the distance between them must be less than a given generalization limit (D_f). Otherwise, the new example is stored as another point partition in the feature dimension f. Assume that the second example e_2 is close to e_1 (i.e., $|x_1 - x_2| \leq D_f$) in feature f and also of the same class. In that case the CFP algorithm will generalize the partition for x_1 into an extended partition $< [x_1, x_2], C_1, 2 >$ which now represents two examples (see Fig. 3.1.b). Generalization of a range partition is illustrated in Fig. 3.1.d.

If the feature value of a training example falls in a partition with the same class, then simply the representativeness value (number representing the examples in the partition) is incremented by one (Fig. 3.1.c).

If the new training example falls in a partition with a different class than that of the example, the CFP algorithm specializes the existing partition by dividing it into two subpartitions and inserting a point partition (corresponding to the new example) in between them (see Fig. 3.1.e, f). When a partition is divided into two partitions, it is important to distribute the representativeness value of the old partition to the newly formed partitions. The CFP distributes the representativeness of the old partition among the new ones in proportional to their sizes. For instance, the representativeness value of the newly formed partitions in Fig. 3.1.e will be

$$n = 4 \times \frac{x_5 - x_1}{x_4 - x_1},$$

$$m = 4 \times \frac{x_4 - x_5}{x_4 - x_1}.$$

For instances, if $x_4 - x_1 = 6.7$ and $x_4 - x_5 = 4.4$, then $n = 1.37$ and $m = 2.63$.

```
train(Training Set):
begin
    foreach e in Training Set
        foreach feature f
            if class of partition(f, e_f) = e_class
              then w_f = (1 + Δ)w_f
              else w_f = (1 − Δ)w_f
            update-feature-partitioning(f, e_f)
    end
```

Figure 3.2. Training algorithm of the CFP

In terms of production rules, the partitioning in Fig. 3.1.f can be represented as:

if $e_f \geq x_1$ and $e_f < x_5$
then $e_{class} = C_1$

if $e_f = x_5$
then $e_{class} = C_1$

if $e_f > x_5$ and $e_f \leq x_4$
then $e_{class} = C_1$

The CFP algorithm pays attention to the disjointness of the partitions. However, partitions may have common boundaries. In this case the representativeness values of the partitions are used to determine class value. For example, in Fig. 3.1.f at $e_f = x_5$, three classes C_1, C_2 and C_3 are possible, but since the total representativeness of the class C_1 is 4 and that of the other classes is 1, the prediction for the feature f is C_1.

The training process in CFP algorithm has two steps: learning of feature weights and feature partitions (Fig. 3.2). For each training example, the prediction of each feature is compared with the actual class of the example. If the prediction of a feature is correct, then the weight of that feature is incremented by Δ (global feature weight adjustment rate) percent; otherwise, it is decremented by the same amount.

prediction(e):
begin
    $vote_c = 0$
    foreach feature f
        c = class of partition(f, e_f)
        $vote_c = vote_c + w_f$
    return class c with highest $vote_c$.
end

Figure 3.3. The Prediction process of CFP

The prediction in the CFP is based on a vote taken among the predictions made by each feature separately (Fig. 3.3). For a given instance e, the prediction based on a feature f is determined by the value of e_f. If e_f falls properly within a partition with a known class then the prediction is the class of that partition. If e_f falls in a point partition then among all the partitions at this point the one with the highest representativeness value is chosen. If e_f falls in a partition with no known class value, then no prediction for that feature is made. The effect of the prediction of a feature in the voting is proportional with the weight of that feature. All feature weights are initialized to one before the training process begins. The predicted class of a given instance is the one which receives the highest amount of votes among all feature predictions.

Fig. 3.4 shows an example of classification process of the CFP on a domain with four features and two classes. Assume that the test example e has a class value C_1 and features values are x_1, x_2, x_3, and x_4 respectively. The prediction of the first feature is C_1. The second feature predicts *undetermined* as a class value. The prediction of the third and fourth features are C_2. The fourth feature value x_4 of e falls into two partitions. In this case the representativeness values are used to determine the class value (e.g., C_2 partition has greater representativeness value than C_1 partition, so that prediction of the fourth feature is C_2). Final prediction of the CFP depends on the values of the feature weights (w_i's). If $w_1 > (w_3 + w_4)$ then CFP will classify e as a member of C_1 class which is a correct prediction. Otherwise, CFP predicts the class of the e as C_2, which would be a wrong prediction.

The second step in the training process is to update the partitioning of each feature using the given training example (Fig. 3.5). If the feature value

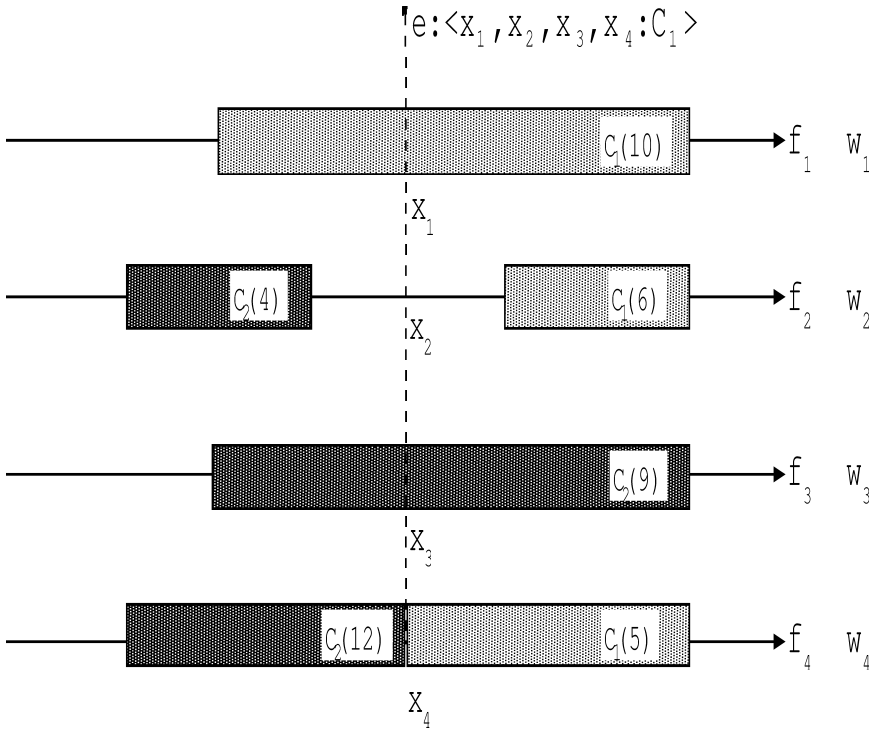

Figure 3.4. The Classification process of CFP

of a training example falls in a partition with the same class, then simply its representativeness value is incremented. If the new feature value falls in a partition with a different class than that of the example and this partition is a point partition, then a new point partition (corresponding to the new feature value) is inserted next to the old one. Otherwise, if the class of the partition is not undetermined, then the CFP algorithm specializes the existing partition by dividing it into two subpartitions and inserting a point partition (corresponding to the new feature value) in between them. On the other hand, if the example falls in an undetermined partition, the CFP algorithm tries to generalize the nearest partition of the same class with the new point. If there exist a partition with the same class in D_f distance, then it is generalized to cover the new feature value. Otherwise, a new point partition that corresponds to the new feature value is inserted.

In order to illustrate the form of the resulting concept descriptions learned by the CFP algorithm, consider a domain with two features f_1 and f_2. Assume that during the training phase, positive $(+)$ instances with f_1 values in $[x_{11}, x_{12}]$ and f_2 values in $[x_{23}, x_{24}]$, and negative $(-)$ instances with f_1 values in $[x_{13}, x_{14}]$ and f_2 values in $[x_{21}, x_{22}]$ are given. The resulting concept description is shown

```
update-feature-partitioning(f, e_f):
begin
    if class of partition(f, e_f) = e_class
      increment representativeness value of partition(f, e_f)
    else {different class}
      if partition(f, e_f) is a point partition
        insert-new-partition(f, e_f)
      else {partition(f, e_f) is a range partition}
          if class of partition(f, e_f) is not undetermined
            subdivide-partition(partition(f, e_f), e_f)
          else {try to generalize}
              if the nearest partition to left or right in D_f distance
              has the class e_class
                generalize(partition,e_f)
              else {there is no partition in D_f distance with the
              same class as e}
                  insert-new-partition(f, e_f)
end
```

Figure 3.5. Updating a feature partition

in Fig. 3.6.

For test instances which fall into the region $[-\infty, x_{11}][x_{23}, x_{24}]$, for example, feature f_1 has no prediction, while feature f_2 predicts as class (+). Therefore, any instance falling in this region will be classified as (+). On the other hand, for instances falling into the region $[-\infty, x_{11}][-\infty, x_{21}]$, for example, the CFP algorithm does not commit itself to any prediction.

If both features have equal weight ($w_1 = w_2$) then, the description of the concept corresponding to the class + shown in Fig. 3.6 can be written in 3-DNF as:

class +:

$$(x_{11} \leq f_1 \ \& \ f_1 \leq x_{12} \ \& \ f_2 < x_{21}) \text{ or}$$
$$(x_{11} \leq f_1 \ \& \ f_1 \leq x_{12} \ \& \ f_2 > x_{22}) \text{ or}$$
$$(x_{23} \leq f_2 \ \& \ f_2 \leq x_{24} \ \& \ f_1 < x_{13}) \text{ or}$$
$$(x_{23} \leq f_2 \ \& \ f_2 \leq x_{24} \ \& \ f_1 > x_{14})$$

More compactly:

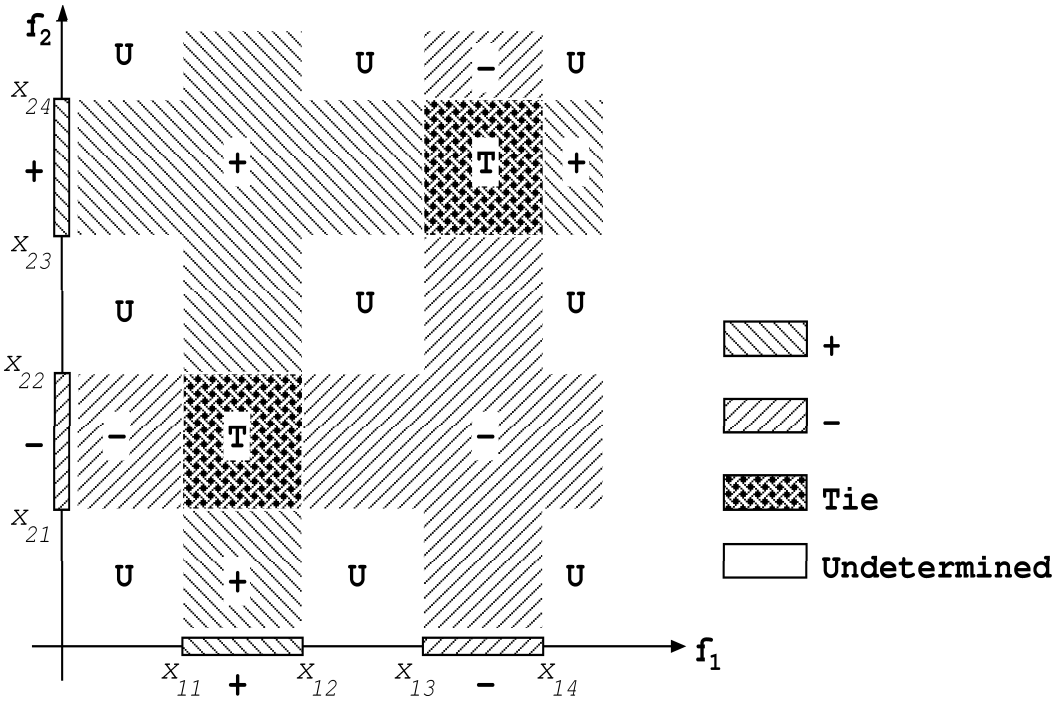

Figure 3.6. An example concept description in a domain with two features

class $+$:

$$[(x_{11} \leq f_1 \leq x_{12}) \ \& \ (f_2 < x_{21} \text{ or } x_{22} < f_2)] \text{ or}$$
$$[(x_{23} \leq f_2 \leq x_{24}) \ \& \ (f_1 < x_{13} \text{ or } x_{14} < f_1)]$$

Similarly, the description for the negative examples can be written as:

class $-$:

$$[(x_{13} \leq f_1 \leq x_{14}) \ \& \ (f_2 < x_{23} \text{ or } x_{24} < f_2)] \text{ or}$$
$$[(x_{21} \leq f_2 \leq x_{22}) \ \& \ (f_1 < x_{11} \text{ or } x_{12} < f_1)]$$

The CFP does not assign any classification to an instance if it could not determine the appropriate class value for that instance. This may result from having seen no instances for a given set of values or having a tie between two or more possible contradicting classifications. In case of different weight values for the features, the ties are broken in favor of the class predicted by the features with the highest weights during the voting process (with equal feature weights, it corresponds to the majority voting scheme of feature predictions).

If $w_1 > w_2$ then the ties will be broken in favor of f_1 during the voting process. In that case the concept description of the class + will be as follows.

class +:

$[x_{11} \leq f_1 \leq x_{12}]$ or
$[(x_{23} \leq f_2 \leq x_{24})$ & $(f_1 < x_{13}$ or $x_{14} < f_1)]$

The description of the class − will be as follows.

class −:

$[x_{13} \leq f_1 \leq x_{14}]$ or
$[(x_{21} \leq f_2 \leq x_{22})$ & $(f_1 < x_{11}$ or $x_{12} < f_1)]$

Similarly, if $w_2 > w_1$ then the ties will be broken in the favor of f_2 during the voting process. In that case the concept description of the class + will be as follows.

class +:

$[x_{23} \leq f_2 \leq x_{24}]$ or
$[(x_{11} \leq f_1 \leq x_{12})$ & $(f_2 < x_{21}$ or $x_{22} < f_2)]$

Similarly, the description for the negative examples can be written as:

class −:

$[x_{21} \leq f_2 \leq x_{22}]$ or
$[(x_{13} \leq f_1 \leq x_{14})$ & $(f_2 < x_{23}$ or $x_{24} < f_2)]$

Having described the basic CFP algorithm, we will now describe several extensions. These extensions are the noise tolerant version, parallelized version, and the version that uses a genetic algorithm to determine the domain dependent parameters of the CFP algorithm.

3.2 A noise tolerant version of the CFP

There are several types of noise that may exist in a data. One possible type of noise is the *classification noise*. Here the attribute values of an instance represent a valid point in the instance space, however the associated classification is incorrect. In order to cope with this type of noise one must be able to differentiate misclassified instances from correctly classified ones.

This section presents an extension to the CFP algorithm to remove the partitions that are believed to be introduced by noisy instances. A new parameter, called *confidence threshold (or level)* (CT), is introduced to control the process of removing the partitions from the concept description. The confidence threshold and observed frequency of the classes are used together to decide that a partition is noisy. The CT is also a domain dependent parameter of the CFP.

Partition removing operation is used as a background operator and is activated when specialization of a partition is performed. Thus, this operation is simple and does not introduce additional computational cost to the training process. If the new training example falls in a partition with a different class than that of the example in a feature, the CFP algorithm specializes the existing partition by dividing it into two subpartitions and inserting a point partition, corresponding to the new example, in between and distributes the representativeness value of the old partition to the newly formed partitions. If the representativeness values of any of the resulting subpartitions drop below the confidence threshold times the observed frequency of its class, then that subpartition is removed from partition list of the feature.

Depending on the noise level of the data set and the number of the irrelevant attributes, the value of the confidence threshold changes between 0 (do not remove any partition) and 1 (remove a partition if its representativeness value drops below the observed frequency of the its class).

The confidence threshold is used to improve accuracy of the CFP and also to reduce the memory requirement. Empirical evaluation of the confidence threshold and other domain dependent parameters of the CFP on artificially generated data sets are given in Chapter 4.

3.3 Parallelization of the CFP

Achieving speed-up through parallelism on multicomputer architectures on multicomputer architectures is not straightforward. An algorithm must be designed so that both computations and data can be distributed to the processors in such a way that computational task can be run in parallel. The computational load should be balanced as much as possible. Communication between processors, in order to exchange data, must be considered as part of the algorithm design. Unless parallel algorithms reduce both the number and the volume of the interprocessor communication through problem partitioning with judicious use of communication, the benefits of parallel processing can be easily offset by the communication overhead.

If we consider the above issues, the CFP algorithm is naturally suitable for parallel implementation. Since, the CFP learns one feature at a time, it is very natural for mapping the training process to parallel architectures. A parallel algorithm can be developed to learn feature partitions in any fixed number of dimensions. The formal analysis of the parallelization of the CFP is presented in Section 4.1.

The main problem in the parallel implementation of the CFP is load-balancing. If the CFP constructs small number of partitions for some features and too many partitions for others, then the utilization of the processors, corresponding to the features with small number of partitions, will be low. However, if number of the partitions on each feature dimensions are nearly equal, then parallelization will result in high efficiency.

3.4 The GA-CFP Algorithm

Kelly and Davis have developed a hybrid genetic algorithm for the K Nearest Neighbors (KNN) classification algorithm [22]. Their algorithm, called the GA-WKNN (for Genetic Algorithm with Weighted K Nearest Neighbor), combines the optimization capabilities of a genetic algorithm with the classification capabilities of the weighted KNN algorithm (WKNN). The goal of the GA-WKNN algorithm is to learn a feature weight vector which improves the common k nearest neighbor algorithm.

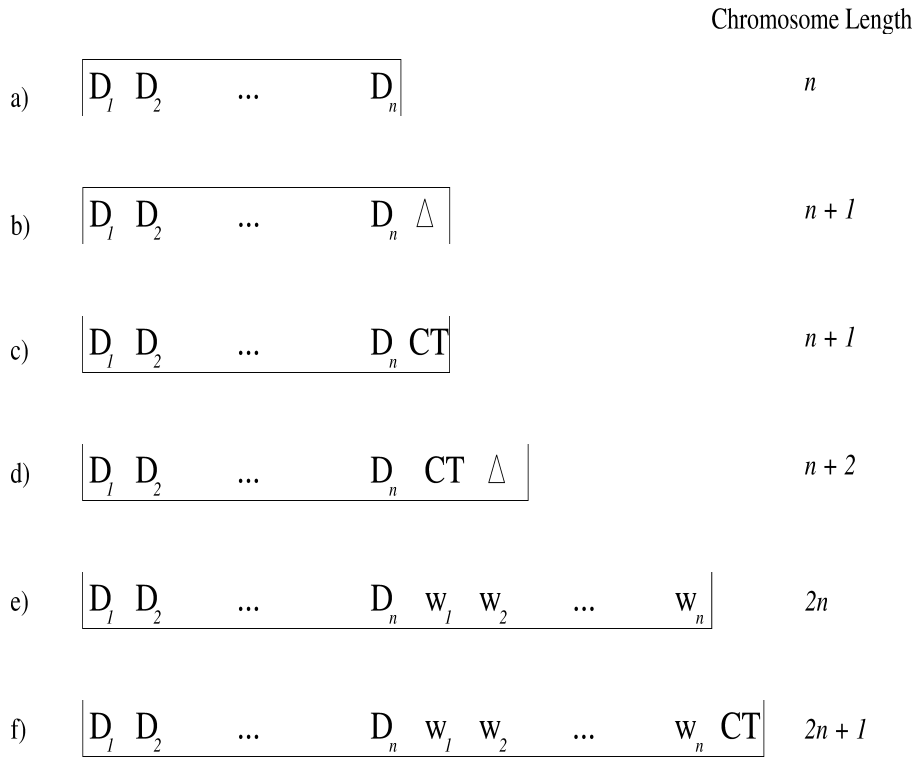

Figure 3.7. Parameter encoding schemes of the GA-CFP

This section describes a hybrid system, called GA-CFP,[1] which combines a genetic algorithm with the CFP algorithm. The GAUCSD 1.4 genetic algorithm package [36] is used to implement the GA-CFP algorithm. The standard operators of genetic algorithms were used, namely, reproduction, crossover and mutation [14]. The chromosomes are treated as rings and crossover is done by exchanging the sections between two crossover points. The genetic algorithm is used to determine the domain dependent parameters of the CFP algorithm. It is a difficult problem to find an optimum setting of these parameters. Empirical results of the GA-CFP are presented in Chapter 4. These results show that the genetic algorithm was able to find good setting for these parameters. They are good in the sense that they outperform the cases where the feature weights are identical, the generalization limits are set to two extremes, and the confidence threshold is set to zero.

We investigated six different parameter encoding schemes with GA-CFP shown in Fig. 3.7. In Fig. 3.7 D_f and w_f represents the generalization limit and weight of the fth feature respectively. Δ represents weight adjustment

[1] A preliminary version of this implementation appeared in [17].

rate and CT represents confidence threshold. The basic difference between these encoding schemes is that the learning of the parameters is achieved by GA or set by the user, consequently size of the search space of the GA-CFP differs. Chromosomes are vectors of real-valued, and represents feature weights, generalization limits, and confidence threshold depending on the employed encoding scheme. Each chromosome is a vector of decimal numbers between 0 and 1 inclusive. A vector value is associated with the weight, the generalization limit for each feature, and confidence threshold. Thus the length of the vector changes between n and $n+1$ on a domain with n features.

In all encoding schemes, generalization limits are learned by GA. Fig. 3.7.a illustrates a chromosome of the GA-CFP, in which generalization limits are learned by GA, while Δ and CT are set by user. A vector value is associated with the generalization limit for each feature. Thus the length of the vector is equal to the number of features. With encoding scheme illustrated in Fig. 3.7.b Δ is also learned by the GA. On the other hand, in Fig. 3.7.c confidence threshold is learned by the GA and Δ is set by user. In both encoding schemes, the length of the chromosome is the number of features plus one. Fig. 3.7.d illustrates another encoding scheme, where both Δ and confidence threshold are learned by the GA with chromosome length number of features plus two. Fig. 3.7.e illustrates another encoding scheme where feature weights are learned by the GA directly and used by CFP. With this encoding scheme incremental learning of the feature weights is not employed by the CFP. Most general encoding scheme is shown in Fig. 3.7.f in which, in addition to the feature weights, the confidence threshold is also learned by the GA. Thus length of the vector is twice the number of features plus one. This encoding scheme achieves the best accuracy among the others. When learning the feature weights using Δ increment the CFP can not find the intermediate values between the Δ increments, whereas GA-CFP can find those values. The tradeoff is that, this encoding scheme doubles the size of the search space, so it requires more trials to find the good setting of the parameters. If application domain has many attributes encoding scheme b,c, or d can be used with GA-CFP. Empirical results indicate that CFP is not very sensitive to the changes in these parameters. Reasonable settings can be found by trial and error.

The initial population of chromosomes in each run of the GA-CFP algorithm was randomly generated. The fitness function used to evaluate the chromosomes is the accuracy of the CFP algorithm with the weights, generalization limits, and confidence threshold encoded in the chromosome (see Fig. 3.8). In

```
CFP(Training Set, Test Set):
begin
      { train }
      foreach e in Training Set
         foreach feature f
            update-feature-partitioning(f, e_f)
      { test }
      correct-count = 0
      foreach e in Test Set
         P=prediction(e)
         if P == e_class then
            increment correct-count
      return correct-count / Test Set size
end
```

Figure 3.8. The CFP fitness function of the GA-CFP

order to compute the fitness value of a chromosome, the CFP algorithm is trained with the examples in the training set using the feature weights, generalization limits, and confidence threshold then tested with the examples in the test set. The fitness value is computed as the ratio of the correctly predicted test examples to the size of the test set.

In Chapter 4 the performance of the GA-CFP is tested with various real-world data sets and compared with regular CFP and other similar algorithms. Although the results of the GA-CFP algorithm are better than other classification systems, the use of a genetic algorithm is costly. The computation of the fitness function requires the execution of the CFP algorithm several times due to the cross-validation method. However, an important characteristic of the feature weight and generalization limit parameters of the CFP algorithm is that these parameters are domain dependent. Therefore, the genetic algorithm can be used only with a portion of all the data available. As an experiment, the GA-CFP system was trained with only 1/10, 1/5, 1/4, 1/3, 1/2, 2/3, and 3/4 of the Iris data set, which contains 150 instances. Then, the accuracy of the CFP algorithm was measured on the complete set of data using the parameter settings learned by the GA-CFP (Fig. 3.9). The cross-validation method was used in the test of CFP. For example, 0.92 is the average of 10 runs in each of which a disjoint set of 1/10 of the data is used with GA-CFP

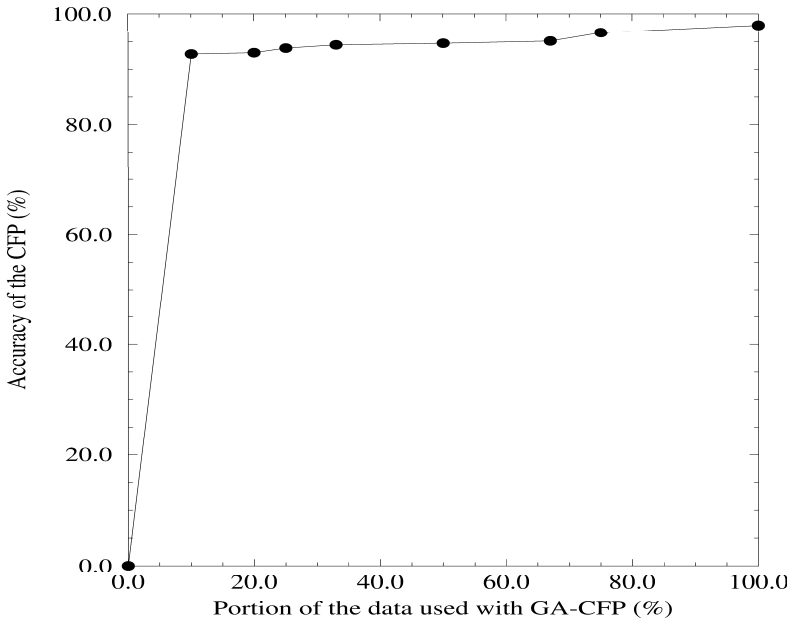

Figure 3.9. Learning curve of domain dependent parameters

and the remaining 9/10 in the test with CFP. In determining the fitness value of a chromosome, to avoid the dependency on the order of the examples in the data set, leaving-one-out cross-validation method is used. For example, in the run with 1/10 (15 examples) of data, the fitness value of a chromosome is the average of 15 runs of the CFP with the parameters it encodes; in each run 14 examples were used in training and the remaining one in the test.

It is clear from these experiments that the genetic algorithm can determine a very good set of domain dependent parameters of the CFP, even when trained with a small portion of the data set. Obviously, the larger portion of the data is used, the better parameters are found.

An algorithm that hybridizes the classification power of the feature partitioning CFP algorithm with the search and optimization power of the genetic algorithm is presented. The resulting algorithm GA-CFP requires more computational capabilities than the CFP algorithm, but achieves improved classification performance in reasonable time. Experimental results indicate that in many real-world domains the GA-CFP algorithm outperforms other classification techniques such as IBL, C4.5, and GA-WKNN.

We have also noticed that the genetic algorithm can be trained with only a small portion of the data to learn the domain dependent parameters of the CFP algorithm with satisfactory prediction accuracy. We anticipate that extensions to the research will improve the algorithm's performance.

3.5 Limitations of the CFP

The CFP learns a partitioning of values for each feature of the application domain. The CFP algorithm is applicable to concepts where each feature, independent of other features, can contribute to the classification of the concept.

This approach is a variant of algorithms that learn by projecting into one feature dimension at a time. For example, ID3 learns in that greedy manner while building a conjunction. The novelty of CFP is that it retains a feature-by-feature representation and uses a voting scheme for categorization. Algorithms that learn by projecting into one dimension at a time are limited in their ability to find complex concepts.

3.5.1 Nonrectangular Concept Descriptions

Each partition represents two (one if lower and upper values of the partition are equal) parallel surfaces (hyperplanes) in the description space; these are orthogonal to the axis of the partition and parallel to all other axes. Consequently, the regions constructed by the CFP are disjoint hyperrectangles.

When actual class regions are not hyperrectangles, the best that CFP can do is to approximate the regions by small hyperrectangles. This is illustrated[2] in Fig. 3.10 by artificial data in which 50 instances of two classes (represented by $-$ and $+$) are described by two continuous attributes F1 and F2, respectively. Here, the class symbols of range partitions are shown above/right of the corresponding partition, class symbol of point partitions (exceptions) are shown below/left of the point. Since the instances are symmetrically distributed in the feature space and the weight of the features are close to each other as expected. For higher values of F1 class $+$ will be predicted and for higher values of F2 class $-$ will be predicted.

[2]Fig. 3.10 is a snapshot of the program called "show concepts" which is developed to visualize the example points and constructed partitions on a two-dimensional plane.

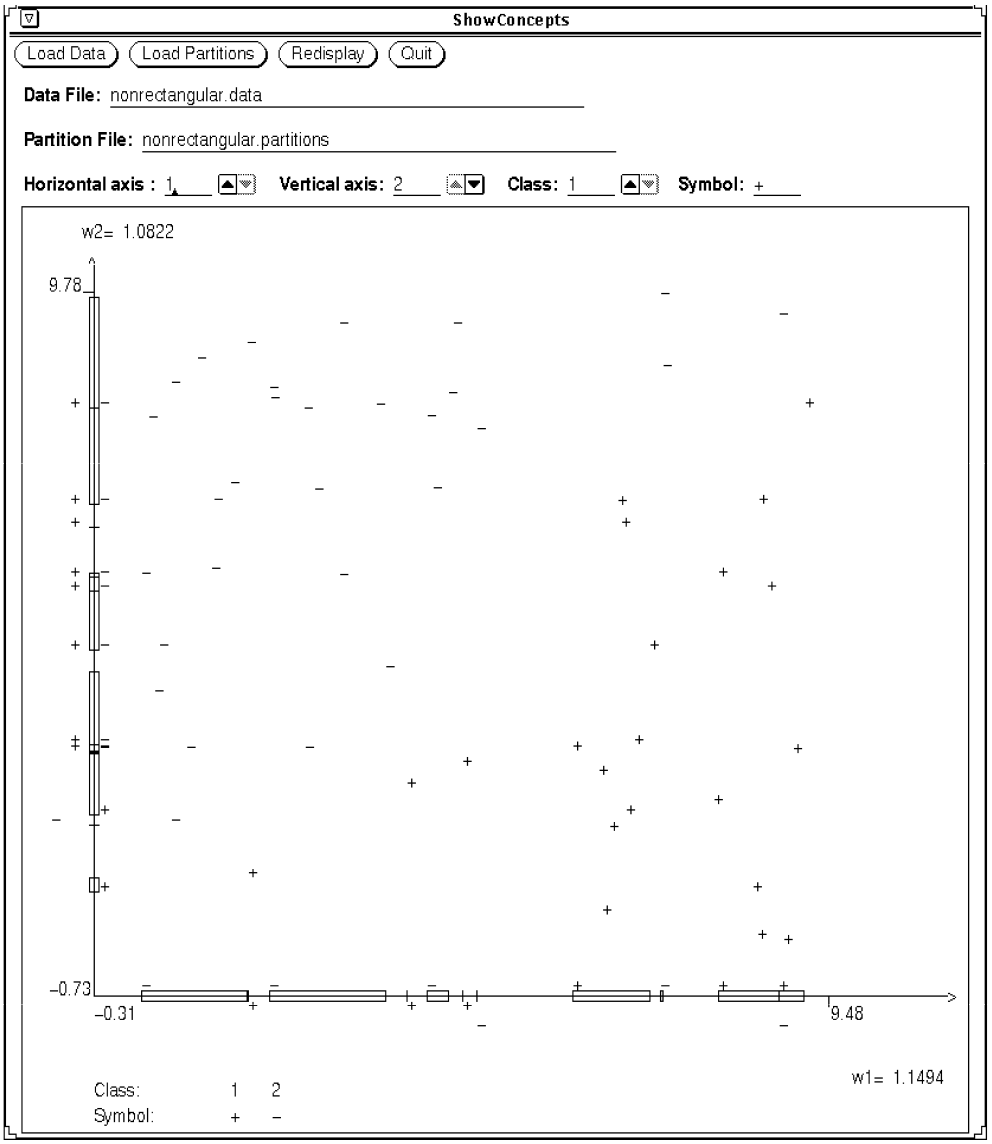

Figure 3.10. An example of nonrectangular concept descriptions

3.5.2 Overlapping Concept Description Projections

CFP learns by projecting into one feature dimension at a time. Therefore, it loses the *n-dimensional* information of the description space. Fortunately, this weakness is compensated with the voting scheme and yields significant reduction in the learning complexity and memory requirement of the CFP.

If the projections of concepts on feature dimensions do not overlap, then the CFP will classify an instance with a high confidence. However, in some circumstances, e.g., in the nested concept description, this may not hold. This is illustrated in Fig. 3.11 on artificial data set in which 100 instances of two

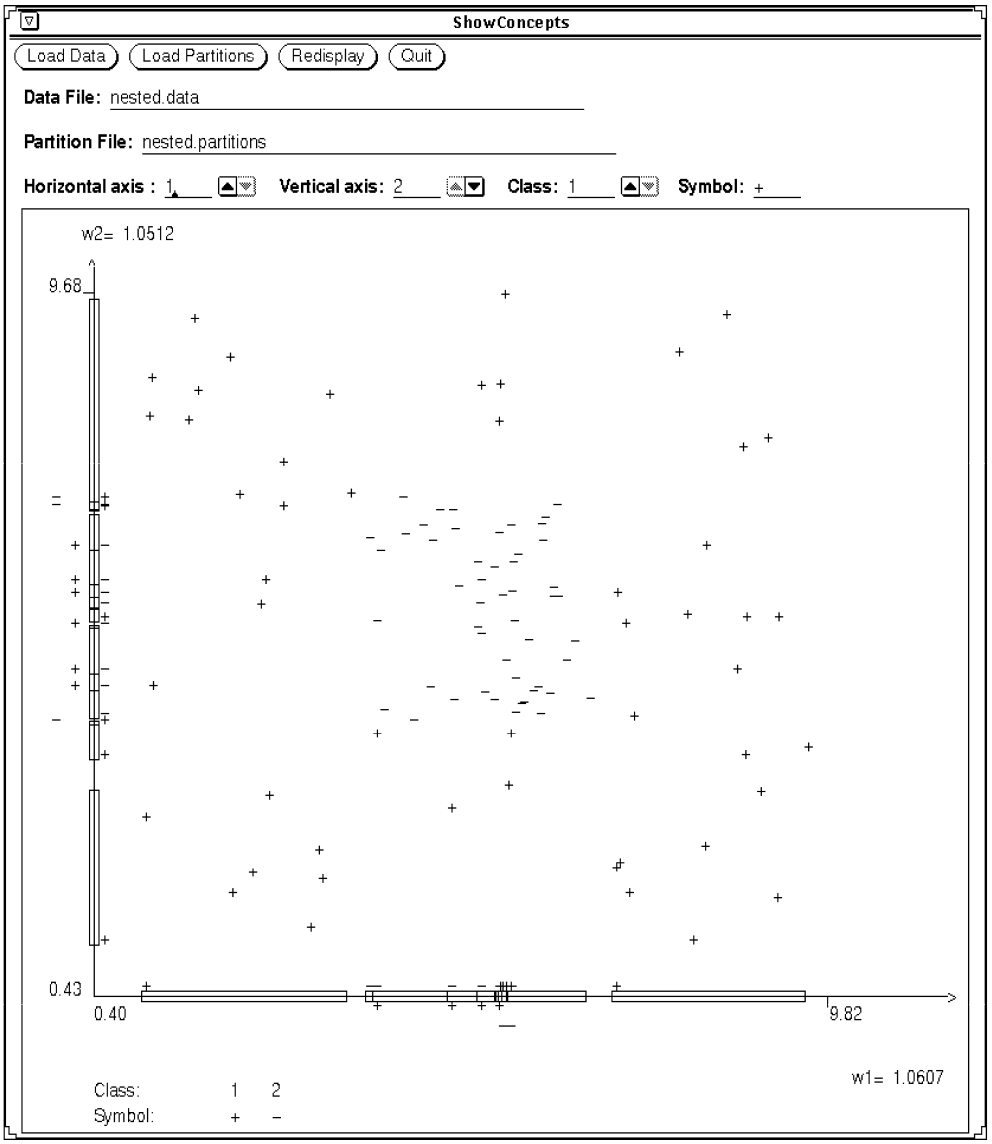

Figure 3.11. An example of nested concept descriptions

classes (represented by $-$ and $+$) are described by two continuous attributes F1 and F2, respectively. Instances falling into the region $< [4,7],[4,7] >$ will be correctly classified, because this region contains 50 instance of class $-$. The reset of the feature space contains 50 instances of class $+$. Consequently, the representativeness value of the class $-$ partitions will be greater than the class $+$ partitions. Therefore, the nested concept will be correctly identified. However, for example, the instances falling into the region $< [4,7],[1,4] >$ be classified as class $-$ (since $w_1 > w_2$).

In some cases, even when concepts descriptions are not nested, the projection of the concept descriptions overlap. This is illustrated in Fig. 3.12 and

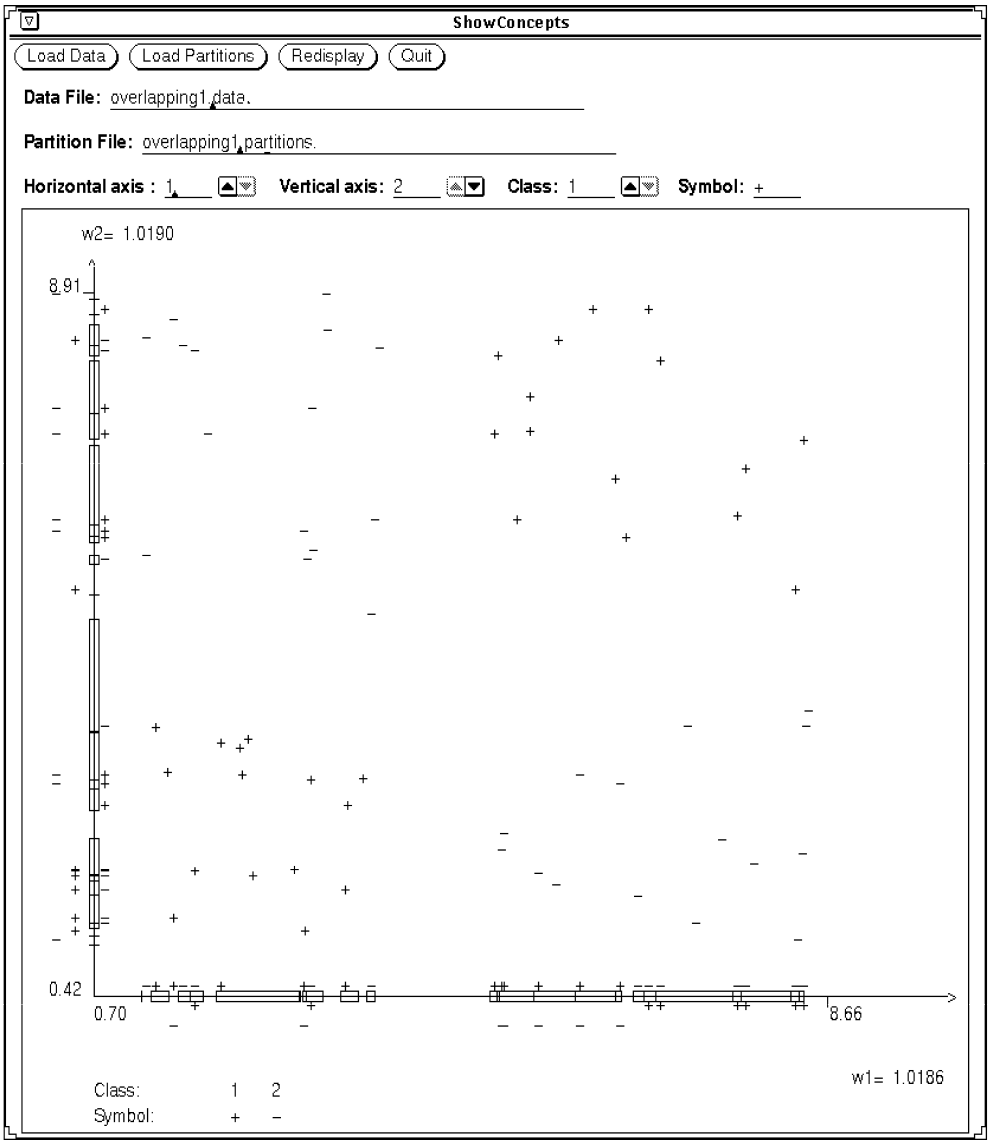

Figure 3.12. An example of overlapping projection of concept descriptions

Fig. 3.13 on artificial data in which 80 and 60 instances respectively, of two classes (represented by $-$ and $+$) are described by two continuous attributes F1 and F2. In Fig. 3.12 the instances are symmetrically distributed in the feature space and the weights of the features are close to each other as expected.

An example of partially overlapping projection of concept descriptions is shown in Fig. 3.13. This artificial data set contains 60 instances which are members of class $+$ or class $-$. The CFP algorithm constructs two range partitions for the non-overlapping projections of the class $-$. Therefore, the CFP will correctly classify the class $-$ examples. However, both projections of the

class + are overlapping with class − projections. Therefore, the classification of class + examples depends on representativeness values of constructed partitions for that part of the feature space.

3.5.3 Domain Dependent Parameters of the CFP

The CFP uses feature weights to cope with unequally relevant attributes. In the CFP the feature weights are dynamically adjusted according to the global weight adjustment rate (Δ), which is an important parameter for the predictive accuracy of the algorithm. The generalization process of the CFP is controlled by the generalization limits for each feature. Another important component of the CFP is the confidence threshold (CT) parameter, which controls the process of removing the partitions that are believed to be noisy from the concept description. The Δ, D_f, and CT are domain dependent parameters of the CFP, and their selection affects the performance of the algorithm. Determining the best values for these parameters is an optimization problem for a given domain. Fortunately, this problem is solved by GA-CFP algorithm as explained in Section 3.4.

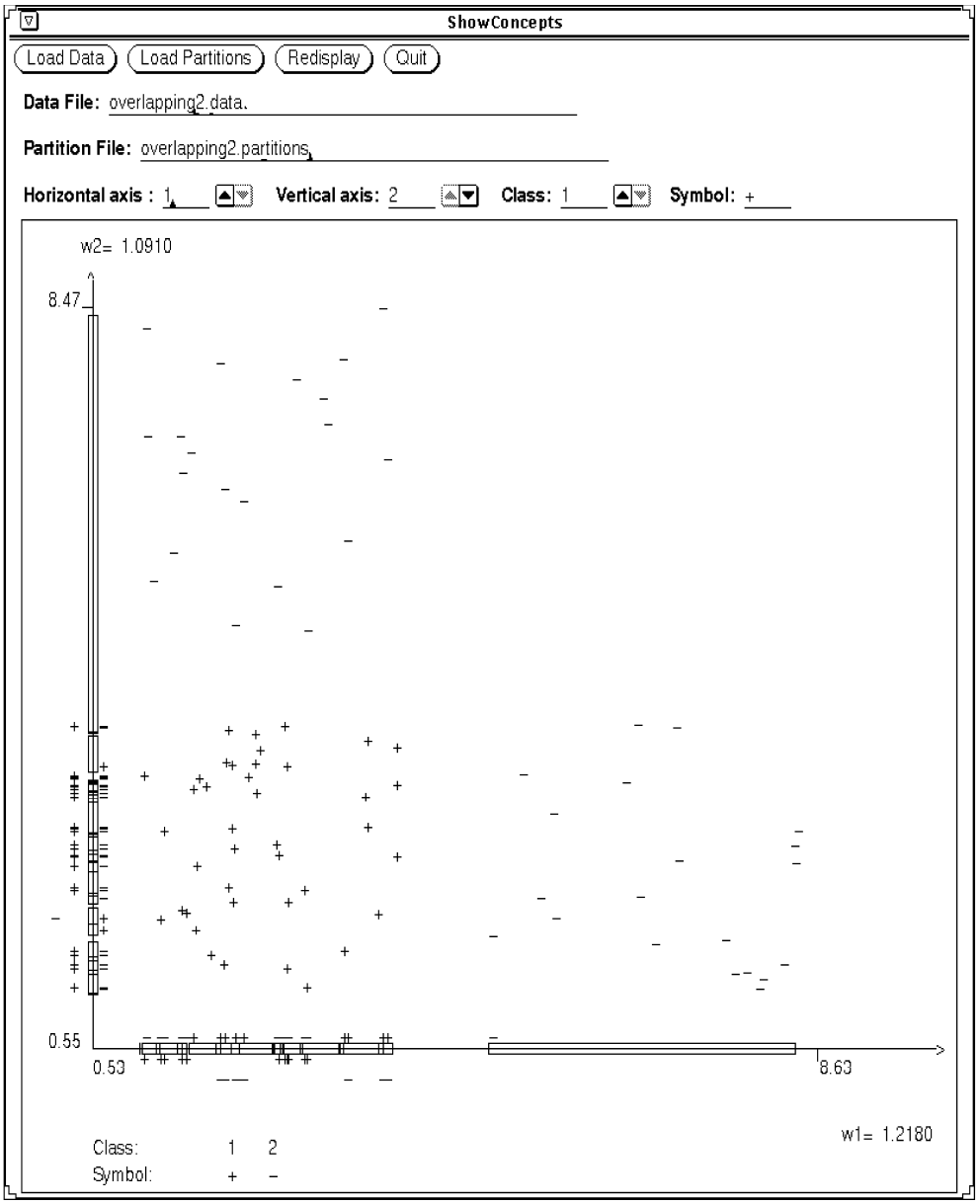

Figure 3.13. An example of partially overlapping projection of concept descriptions

Chapter 4

Evaluation of the CFP

This chapter presents theoretical and empirical analysis of the CFP algorithm. From the viewpoint of empirical research, one of the main difficulty in comparing various algorithms which learn from examples is the lack of a formally specified model on which the algorithms may be evaluated. Typically, different learning algorithms and theories are given together with examples of their performance, but without a precise definition of *learnability*, it is difficult to characterize the scope of applicability of an algorithm or analyze the success of different approaches and techniques.

Informally speaking, a concept is a subset of the objects in a predefined domain and the problem of learning a concept from examples is the following: Given some examples for an unknown concept and/or some prior information on it, compute a good approximation to the concept. The concept for which examples are provided is known as *target concept* [26].

4.1 Theoretical Evaluation of the CFP

This section presents an analysis of the CFP algorithm[1] with respect to *PAC-learning* theory [41]. Valiant introduced this theory in 1984, by taking some simplified notions from statistical pattern recognition, decision theory, and combining them with approaches from computational complexity theory, he came up with a notion of learning problems that are feasible in the sense that there is a polynomial time algorithm that *solves* them. Valiant was successful

[1] A preliminary version of this section will appear in [16].

in his efforts. Since 1984 many theoretical computer scientists and AI researchers have either obtained results in this theory, or criticized about it and proposed modified theories, or both [20]. The intent of the PAC (Probably Approximately Correct) model is that successful learning of an unknown target concept should entail obtaining, with high probability, that it is a good approximation of the concept. In the basic model, the instance space is assumed to be $\{0,1\}^n$, the set of all possible assignments to n binary variables (or attributes). Concepts and hypotheses are subset of $\{0,1\}^n$. The notion of approximation is defined by assuming that there is some probability distribution D defined on the instance space $\{0,1\}^n$, giving the probability of each instance. Then the *error* of hypothesis h with respect to a fixed target concept c is defined as:

$$error(h) = \sum_{x \in h \Delta c} D(x).$$

where Δ denotes the symmetric difference. Thus *error(h)* is the probability that h and c will disagree on an instance drawn randomly according to D. The hypothesis h is a good approximation of the target concept c, if *error(h)* is small.

How does one obtain a good hypothesis? In the simplest case one does this by looking at independent random examples of the target concept c [5]. Each example consists of an instance selected randomly according to D and a label that is + (*positive example*) if that instance is in the target concept c, otherwise − (*negative example*). Training and testing phases use the same distribution of examples, and there is no *noise* in either phase.

The two criticisms most often leveled at the PAC model by AI researchers interested in empirical machine learning are (1) the worst-case emphasis in the model makes it unusable in practice, and (2) the notions of target concepts and noise-free training data are too restrictive in practice [18, 26]. Some extensions are proposed to the original PAC-model to handle the noisy sample data [3, 20]. These extensions preserve the generality of the PAC-model, and do not make any assumptions on the nature of the noise.

Since the classification in the CFP is based on a voting taken among the individual classifications of each attribute, it can learn a concept if each attribute, independently from other attributes, can be used in the classification. We will define what we mean by "learning" in a way that preserves the spirit of the Valiant's definition of learnability, but modifies it for the voting based

classification used in the CFP. In order to do this we will first determine the *sample complexity*, that is the minimum number of examples required to learn a given concept. Then, using this sample complexity, we will derive the *training complexity* of the CFP algorithm. In the following analysis we assume that all feature values are normalized to the interval [0,1], and features are equally relevant that is they have equal weights.

Definition. Let X be a subset of $\Re^n$ with a fixed probability distribution and d is positive integer less than or equal to n. A subset S of X is an $< \varepsilon, \gamma, d > -net$ for X if, for all x in X, with probability greater than γ, there exist an s in S such that $|s_f - x_f| < \varepsilon$ at least for d values of f ($1 \leq f \leq n$).

Lemma 1. Let ε, δ, and γ be fixed positive numbers less than one and d is positive integer less than or equal to n. A random sample S containing $m > (\lceil 1/\varepsilon \rceil / \gamma) \times (n \ln 2 + \ln(\lceil 1/\varepsilon \rceil / \delta))$ instances, drawn according to any fixed probability distribution from $[0,1]^n$, will form an $< \varepsilon, \gamma, d >$-net with confidence greater than $1 - \delta$.

Proof. We prove this lemma by partitioning the unit interval for each feature dimension, into k equal length sub-intervals, each with length less than ε, such that all pairs of points[2] in the sub-interval are within ε distance of each other. The idea of the proof is to guarantee that, with high confidence, at least for d dimensions out of n, each of k sub-intervals contains at least one point of m instances, with sufficient probability.

Let $k = \lceil 1/\varepsilon \rceil$, S_{1f} be the set of sub-intervals with probability greater or equal to γ/k and S_{2f} be the set of remaining sub-intervals of a dimension f. The probability that an arbitrary point in $[0,1]$ will not lie in a selected sub-interval of S_{1f} is $(1 - \gamma/k)$. The probability that none of the m sample points will lie in a selected sub-interval of S_{1f} is $(1 - \gamma/k)^m$. Therefore, the probability that any sub-interval of S_{1f} is excluded by all m instances is at most $p = k(1 - \gamma/k)^m$.

The probability that, for more than $n - d$ dimensions, any sub-interval of S_1's are excluded by all m instances is at most $\sum_{i=n-d+1}^{n} C(n,i)p^i$.[3] To make sure that this probability is small, we force it to be less than δ, that is,

$$\sum_{i=n-d+1}^{n} C(n,i)p^i < \delta.$$

[2] A point here represents the value of an instance for a feature for that dimension.

[3] $C(n,r)$ represents the number of combinations of r objects out of n.

Recall the *binomial theorem*: $(a+b)^n = \sum_{i=0}^{n} C(n,i)a^i b^{n-i}$. With $a = p$ and $b = 1$, $\sum_{i=0}^{n} C(n,i)p^i = (p+1)^n$. Since n is a positive integer, $(p+1)^n - 1 = \sum_{i=1}^{n} C(n,i)p^i$ and it is greater than $\sum_{i=n-d+1}^{n} C(n,i)p^i$, our requirement can be written as

$$(p+1)^n - 1 < \delta.$$

On the other hand, $(1-\gamma/k)^m < e^{-m\gamma/k}$ and, since the value of p is greater than zero and less than one, $2^n p > (p+1)^n - 1$. If we solve the requirement $2^n k e^{-m\gamma/k} < \delta$, for m, and substitute $\lceil 1/\varepsilon \rceil$ for k, it yields

$$m > \lceil 1/\varepsilon \rceil / \gamma \times (n \ln 2 + \ln(\lceil 1/\varepsilon \rceil / \delta)).$$

Consequently, with confidence greater than $1-\delta$, each sub-interval in S_{1f} of d or more dimensions, contains some sample point of an instance of S. □

Theorem 1. Let ε, δ, and γ be fixed positive numbers less than one and a sample set S with n features. If $\lceil \frac{n+1}{2} \rceil$ of features of the elements of S form an $< \varepsilon, \gamma, \lceil \frac{n+1}{2} \rceil >$-net then, the CFP algorithm with equal feature weights and generalization limit $D_f \geq 2\varepsilon$ for all features, will learn a concept C for S with confidence $1-\delta$.

Proof. Since, the CFP algorithm does not use a distance metric for classification, the idea of the proof is to ensure that the CFP can construct ε length partitions with high confidence so that at least one of the m sample instances lies in each sub-intervals of $\lceil \frac{n+1}{2} \rceil$ features with sufficient probability. The CFP algorithm employs a majority voting scheme in the classification. Hence, only $d = \lceil \frac{n+1}{2} \rceil$ of the features must agree on the classification. If we follow the proof of the *Lemma 1*, if S form an $< \varepsilon, \gamma, d >$-net, then it guarantees that each sub-interval contains at least one instance of S with high confidence. The CFP algorithm will generalize two points into one partition, if the distance between them is less than or equal to D_f. Therefore, if $D_f \geq 2\varepsilon$ then the points will be generalized into one partition, corresponding to a projection of the concept on that feature. □

Theorem 2. Let ε, δ, and γ be fixed positive numbers less than one. If random sample S with n features forms an $< \varepsilon, \gamma, \lceil \frac{n+1}{2} \rceil >$-net with confidence greater than $1-\delta$, then CFP with $D_f \geq 2\varepsilon$ constructs at most $n\lceil 1/\varepsilon \rceil$ partitions.

Proof. Since S is an $< \varepsilon, \gamma, \lceil \frac{n+1}{2} \rceil >$-net with with confidence greater than $1 - \delta$, each feature line is divided in to ε length sub-intervals and each one contains at least one sample point and the CFP algorithm constructs at most one (due to $D_f \geq 2\varepsilon$) partition for each sub-interval. Thus, for n features, the CFP constructs at most $n\lceil 1/\varepsilon \rceil$ partitions. □

Theorem 3. Let ε, δ, and γ be fixed positive numbers less than one. If random sample S is an $< \varepsilon, \gamma, \lceil \frac{n+1}{2} \rceil >$-net with confidence greater than $1 - \delta$, then classification complexity of the CFP with $D_f \geq 2\varepsilon$ is $O(n \log(\lceil 1/\varepsilon \rceil))$ and the training complexity is for m sample instances is $O(mn \log(\lceil 1/\varepsilon \rceil))$.

Proof. Proof of the Theorem 2 shows, that the CFP constructs at most $\lceil 1/\varepsilon \rceil$ partitions for each feature. In the CFP algorithm the classification is composed of a search and a voting. The complexity of the search operation is $O(\log(\lceil 1/\varepsilon \rceil))$ for each feature. Since the complexity of voting is $O(n)$, the classification complexity of the CFP algorithm is $O(n \log(\lceil 1/\varepsilon \rceil))$ for n features. Consequently, with m training instances, the training complexity of the CFP algorithm is $O(mn \log(\lceil 1/\varepsilon \rceil))$. □

The analysis of the CFP shows that, it is applicable to a large class of concepts, and requires small number of examples ($m > \lceil 1/\varepsilon \rceil/\gamma \times (n\ln 2 + \ln(\lceil 1/\varepsilon \rceil/\delta))$ examples) and a small amount of memory ($n\lceil 1/\varepsilon \rceil$ partitions) to learn a given concept, compare to many other similar algorithms. Another outcome of the analysis is that, the CFP has a lower learning complexity ($O(mn \log(\lceil 1/\varepsilon \rceil))$) than other similar techniques. For example, sample complexity of the IB1 is $m > \lceil \sqrt{n}/\varepsilon \rceil^n/\gamma \times (\ln(\lceil \sqrt{n}/\varepsilon \rceil^n/\delta))$. IB1 stores all the training instances (m instances). Therefore, the training complexity of the IB1 is $O(nm^2)$ for m training instances [1].

The classification process in exemplar-based learning algorithms which use some form of the nearest neighbor algorithm (such as EACH and IBL) involves computing the Euclidean distance (or similarity) of the instance to each stored exemplar in each dimension. If M exemplars are stored in the memory, and n features are used, then the complexity of the classification is $O(nM)$. On the other hand, since the partitions are naturally sorted for each feature dimension, the classification process in the CFP algorithm is only $O(n \log M)$, which significantly reduces the classification complexity.

Another important feature of the CFP algorithm is its low memory requirement. Since, the CFP learns each feature partition independently of the

others, number of partitions for each feature may be different. If appropriate D_f generalization limits are chosen, CFP may significantly reduce the memory requirement. The selection of D_f values is domain dependent, and it is an optimization problem.

The CFP algorithm is naturally suitable for parallel implementation. Since, the CFP learns one feature at a time, it is very natural for mapping the training process to the parallel architectures. A parallel algorithm can be developed to learn feature partitions in any fixed number of dimension.

Theorem 4. Let ε, δ, and γ be fixed positive numbers less than one. If random sample S is an $< \varepsilon, \gamma, \lceil \frac{n+1}{2} \rceil >$-net with confidence greater than $1 - \delta$, then classification complexity of the CFP, with $D_f \geq 2\varepsilon$ and using n processors, is $O(\log(\lceil 1/\varepsilon \rceil))$ and the training complexity for m sample instances is $O(m \log(\lceil 1/\varepsilon \rceil))$.

Proof. Proof of the Theorem 2 shows, that the CFP constructs at most $\lceil 1/\varepsilon \rceil$ partitions for each feature. In CFP algorithm the classification is composed of a search and a voting. The complexity of the search operation is $O(\log(\lceil 1/\varepsilon \rceil))$ for each feature. Since, the CFP learns feature at a time, search of the each feature dimension can be done in parallel. Training process does not require voting, because in the training process each feature needs a local feedback. However, the result of the voting can be used as a global feedback. Hence, the classification complexity of the CFP algorithm is $O(\log(\lceil 1/\varepsilon \rceil))$ for n processors. Consequently, with m training instances, the training complexity of the CFP algorithm is $O(m \log(\lceil 1/\varepsilon \rceil))$. □

4.2 Empirical Evaluation of the CFP

This section presents the experimental results of the CFP and GA-CFP algorithms. Both algorithms are tested on widely used real data sets and also on artificially generated data sets. The use of the artificially generated data sets, allowed me to test the system in a more controlled way, while the real data sets allowed comparisons with other systems.

The first section describes the methodologies used in the experiments. In the second section, the performance of the CFP and GA-CFP algorithms on artificially generated data sets are presented. In the artificially generated data

sets some of the domain variables (such as number of features, number of examples, amount of noise, unknown attribute values, irrelevant attributes) are changed to test the behavior of the system under different conditions [24]. In Section 2 the effects of the domain dependent parameters of the CFP are also investigated under different settings. Third section presents the performance of the CFP and GA-CFP on real data sets, and comparisons with other similar systems.

Parameters of the Genetic Algorithm used by GA-CFP: Each run of the genetic algorithm maintained a population of size 50–100. Crossover (two point) probability was 0.6 and mutation probability was 0.008. The maximum number of fitness function evaluations in each GA run changed between 2000 and 8000 depending on the size of the search-space that is, the length of the chromosomes.

4.2.1 Testing Methodology

This section briefly describes the methodologies used for testing. In order to compare the CFP and GA-CFP with other learning systems, we used the same testing methodologies used in the reported results of these systems. These methodologies are cross-validation, leave-one-out, and average of randomized runs. we used the same testing methodologies for precise comparison purpose with the other learning algorithms.

Cross-validation: Cross-validation involves removing mutually exclusive test sets of examples from the data set. For each test set, the remaining examples serve as a training set, and classification accuracy is measured as the average accuracy on all the test sets. The union of the all test sets equals to the whole data set.

Leave-one-out: The leave-one-out method involves removing exactly one example from the data and training on the remaining examples and measuring the accuracy by using that single instance as the test instance. The technique is repeated for every example in the data set and accuracy is measured across all examples.

Average of Randomized Runs: This method involves selection of randomly selected training and test sets. Training and test sets are disjoint. The test is

repeated for a fixed number of times (usually 50 trials). The final result is the average accuracy across all trials.

4.2.2 Experiments with Artificial Data Sets

The success of a learning system is highly dependent on the ability to cope with noisy and incomplete data, an adequate knowledge representation scheme, having a low learning and sample complexities, and effectiveness of the learned knowledge [24]. Most of the real-world data sets are incomplete and inconsistent, therefore handling incomplete and inconsistent data is very important.

The ability to form general concept description on the basis of particular examples is difficult task, especially if examples contain errors or "noise". There are various types of noise (e.i. classification noise, attribute noise etc.) that can be found in real-world data sets. Many researchers tackled this problem [3, 12, 35].

Another type of noise is the unknown (*missing*) attribute values. In order to cope with missing attribute values, many techniques were tried. For example, in [15] additional instances are generated for all possible values of the missing attribute and rough set theory is used to solve the conflicts. Obviously, this approach is only applicable to attributes that have finite number of possible values. However, it is a costly solution to handle unknown attributes values.

Handling irrelevant attributes is another problem that has to be solved by a learning algorithm. Most of the real-world data sets contain attributes that are not equally relevant. To test the performance of the CFP algorithm on domains which have unequally relevant attributes, we introduced additional (*irrelevant*) attributes that are randomly generated (uniform between 0 and 100) into the artificial data sets.

The use of the artificially generated data sets allowed me to test the system in a more controlled way. In the artificially generated data sets, some of the domain variables (such as number of features, number of examples, amount of noise, unknown attribute values, irrelevant attributes) are changed to test the behavior of the system under different settings. These data sets contain 300 examples, with 4 features and 3 classes (100 examples for each class). 240 (80 %) of the examples are used in training and the remaining 60 are used in

testing. All the results are the average of 50 runs of the randomly selected training and test sets. The noise free description of the concepts used in the artificial data are given below:

if $[(0 \leq f_1 \leq 2)\ \&\ (0 \leq f_2 \leq 2)\ \&\ (0 \leq f_3 \leq 2)\ \&\ (0 \leq f_4 \leq 2)]$
then class 1
if $[(0 \leq f_1 \leq 2)\ \&\ (0 \leq f_2 \leq 2)\ \&\ (0 \leq f_3 \leq 2)\ \&\ (8 \leq f_4 \leq 10)]$
then class 1
if $[(4 \leq f_1 \leq 6)\ \&\ (4 \leq f_2 \leq 6)\ \&\ (5 \leq f_3 \leq 7)\ \&\ (4 \leq f_4 \leq 6)]$
then class 2
if $[(7 \leq f_1 \leq 10)\ \&\ (7 \leq f_2 \leq 10)\ \&\ (2 \leq f_3 \leq 4)\ \&\ (2 \leq f_4 \leq 4)]$
then class 3

4.2.2.1 Changing Domain Characteristics

This section presents the results of the experiments conducted to compare the C4.5, CFP, and GA-CFP algorithms, on incomplete and inconsistent data sets. The graphs in the sequel depict the results of the algorithms by changing the domain variables such as classification noise, portion of the unknown attribute values, and the number of irrelevant attributes. The C4.5 results indicate the accuracy achieved after tree pruning. Usually, after pruning C4.5 achieves better results.

Learning from Noisy Data

Induction of a concept description from noisy instances is a difficult task. Applicability of a learning algorithm highly depends on the capability of the algorithm handling noisy instances. There are two sorts of noise: (1) *classification noise*, and (2) *attribute noise*. The cause of these errors may be either *systematic* or *random*. Classification noise involves corruption of the class value of an instance, and attribute noise involves replacing of the attribute value of an instance.

Classification noise is more damaging than attribute noise. One of the reason of this is that corrupting class value does not destroy information but reverses it. Whereas, corrupting an attribute value tends to leave enough information in the uncorrupted attributes for adequate learning. Another reason

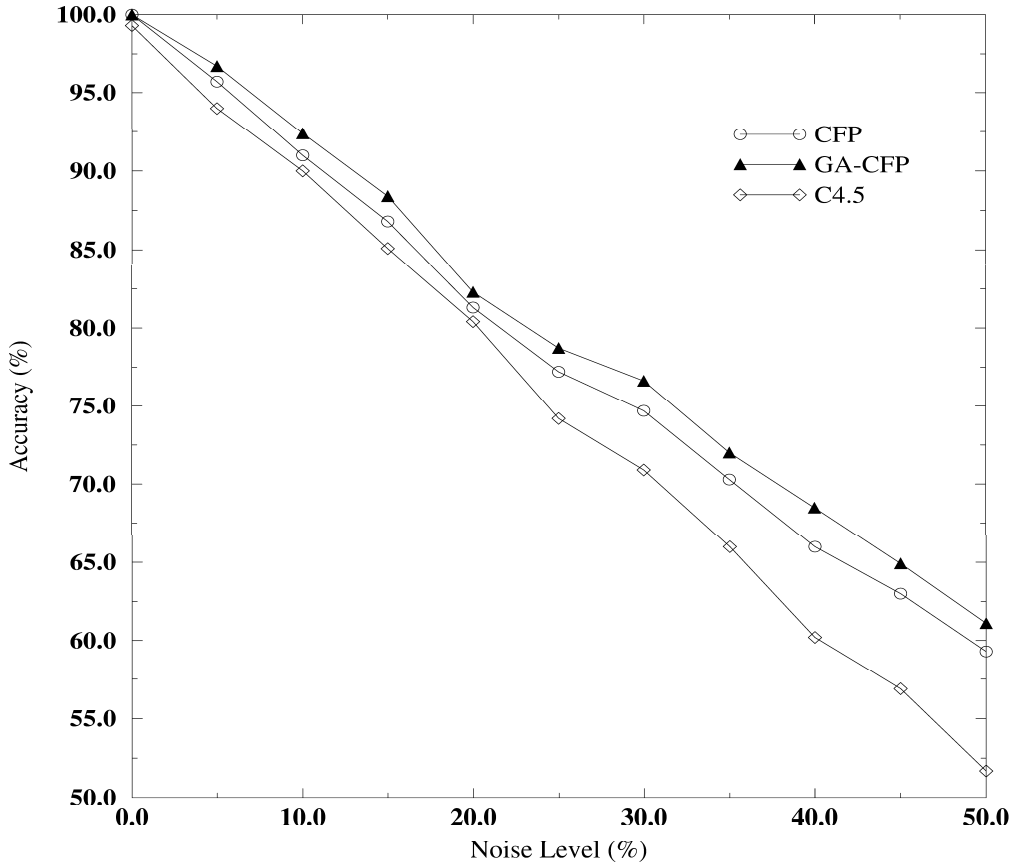

Figure 4.1. Comparison of (GA-)CFP, and C4.5 in terms of accuracy, on a noisy domain

that attribute noise is less damaging is that incorrect attribute values may sometimes produce correct examples especially when the corruption is in the less relevant attributes. In other words, classification noise is worse than attribute noise because changing the class value guarantees that an error is introduced, whereas distorting an attribute value may or may not introduce an error.

Usually, three metrics are used in the comparison of algorithms on noisy domains: (1) accuracy, (2) memory requirement, and (3) percentage of the noise in the learned concept description. Classification noise changing from 0 to 50 percent is introduced into these artificial data sets. Attribute noise is not used because of the reasons mentioned above.

Fig. 4.1 shows the achieved accuracy of the CFP, GA-CFP, and C4.5 algorithms with different amount of classification noise. Results of the GA-CFP and CFP are superior to the result of the C4.5; and the GA-CFP achieved better accuracy than CFP. Classification noise changes the domain characteristics. Consequently, domain dependent parameters of the CFP are changed.

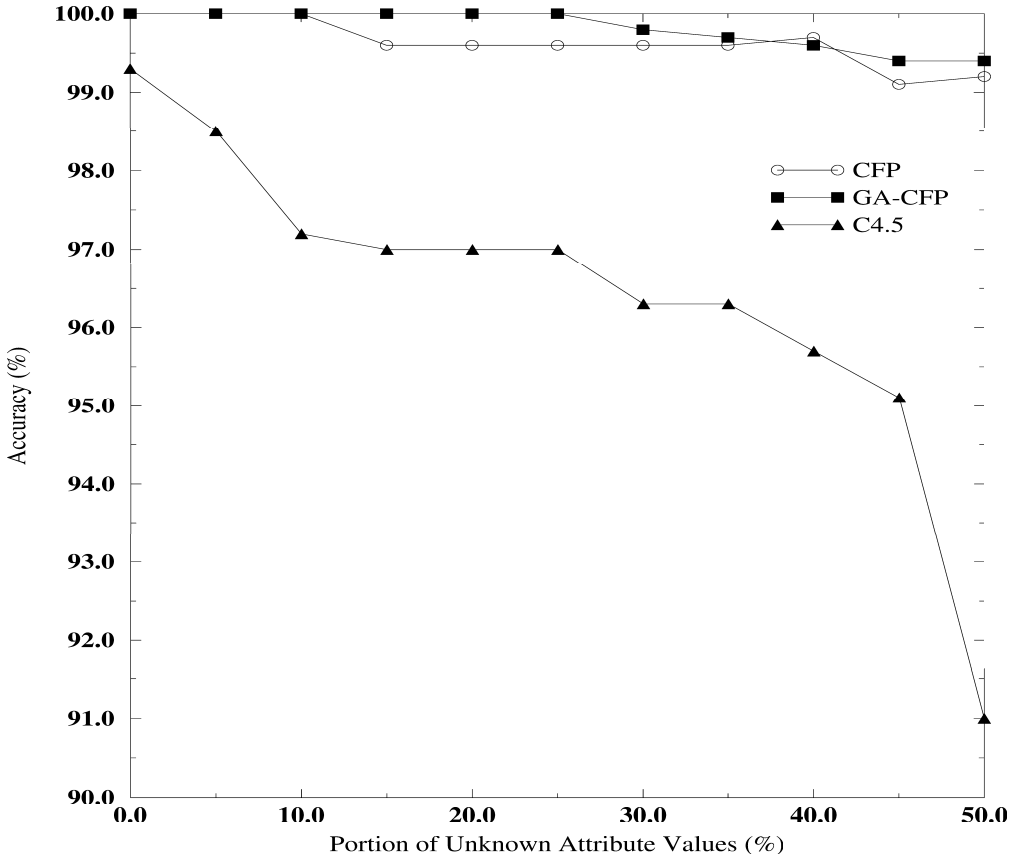

Figure 4.2. Comparison of (GA-)CFP, and C4.5 in terms of accuracy, on a domain that contains unknown attribute values

GA-CFP was able to find new settings of these parameters.

The *confidence level (or threshold)* parameter is introduced into CFP to reduce the percentage of noise in the concept description. Depending of the value of this parameter CFP removes partitions that are believed to be noisy (according to number of instance is represented) from concept description. Higher value of confidence threshold cause removing partitions more aggressively. If confidence threshold is zero then percentage of the noise in the concept description is equal to the noise level of the training set.

Unknown Attribute Values

Most of the real-world data sets contain missing attribute values. Many authors [15, 27, 28, 29] were presented methods for handling unknown attribute values. Most of the methods are based on the following ideas:

1. Ignoring examples which have unknown attribute value.

2. Assuming an additional special value for unknown attribute values. This

can lead to an anomalous situation [27].

3. Using probability theory by utilizing information provided by context.
4. Generating additional instances for all possible values of the unknown attribute [15].
5. Exploring all branches (on decision trees) remembering that some branches are more probable than others [28].

Although these methods for handling unknown attribute values sound promising on paper, they give unconvincing results. However, CFP handles unknown attributes very naturally, since it learns feature-by-feature in the case of an unknown attribute value it simply ignores processing of that feature. The performance degradation of the CFP depends on the information gain of that attribute.

Fig. 4.2 shows the achieved accuracy of the algorithms with different amount of unknown (*missing*) attribute values. GA-CFP and CFP achieved significantly better accuracy than C4.5. These results justify the fact that the classification performance of decision tree based algorithms depends critically on any small part of the model.

Another important point is that CFP simply ignores unknown attribute values which cause reduction in training and testing time. However, C4.5 tries to determine the value of an unknown attribute value using probability distribution of the known values of an attribute and during testing it tries to expand all branches of the tree in the case of a test on unknown attribute value [29], which introduce additional cost in training and testing process.

Learning with Unequally Relevant Attributes

Most of the real-world data sets contains many unequally relevant features. For example, in the task of learning diagnosis rules for several different diseases from the medical records of a large number of patients. These records usually contain more information than is actually required for describing each disease. Another example was given in [2] which involves pattern recognition tasks in which feature detectors automatically extract a large number of features for the learner. Most probably some of these features are not as relevant as the others[23]. Therefore, a learning algorithm should be able to cope with many irrelevant (or unequally relevant) features.

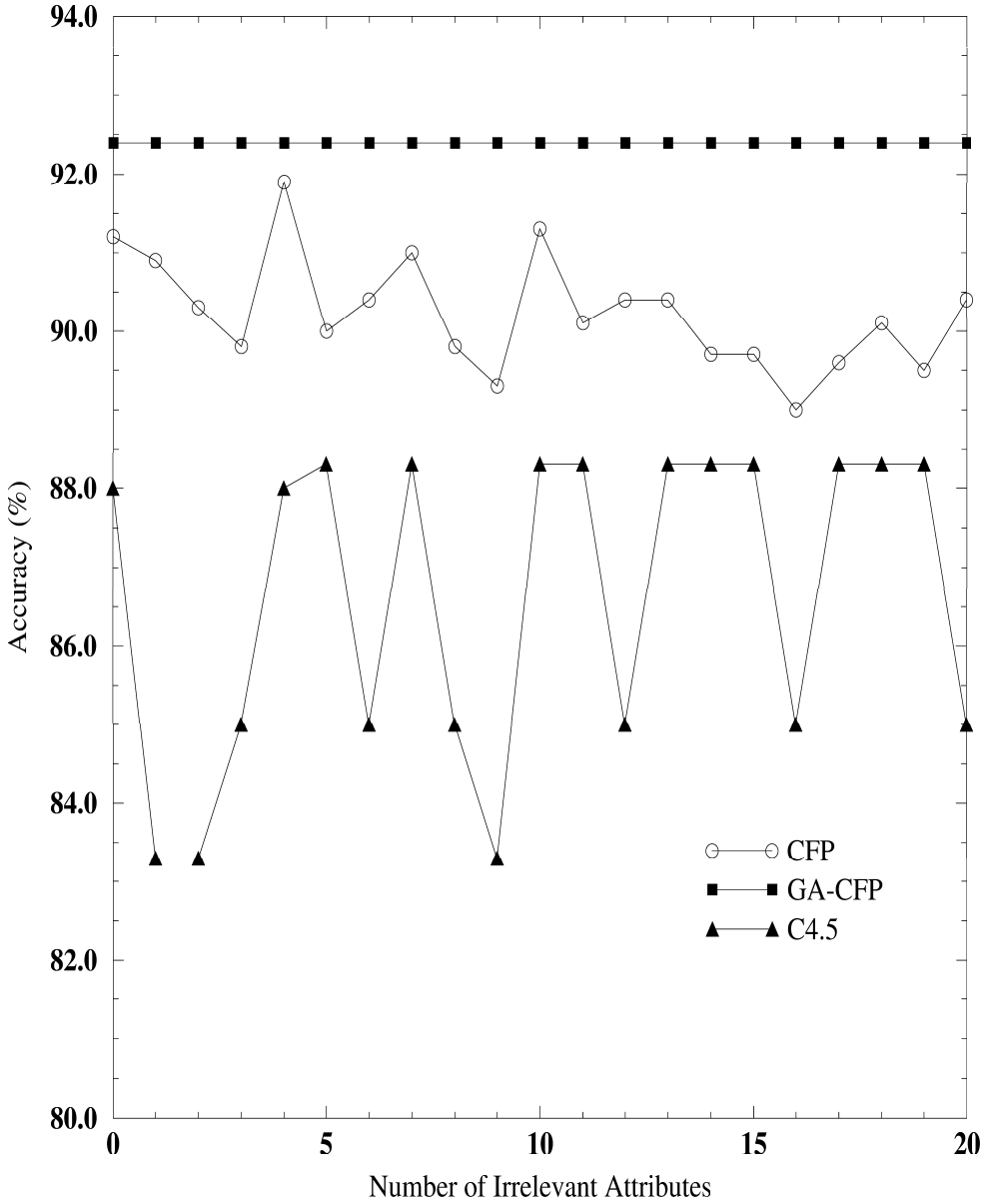

Figure 4.3. Comparison of (GA-)CFP, and C4.5 in terms of accuracy on domains with many irrelevant attributes

The CFP uses feature weights (also successfully used in [22, 34]) to cope with unequally relevant features. In the CFP a feature weight represents the predictive success of that feature. Consequently, final value of the feature weight shows the relevance of that feature in the prediction the projection of concepts on that dimension. With this point of view, the definition of *relevant* is slightly different in this context. A feature may be relevant in the n-dimensional description space. However, in 1-dimensional space it may be less relevant than other features, due to overlapping projection of concept descriptions. This may seem as a weakness of the CFP, but this weakness is compensated with the voting scheme. One or two features with high feature weights is sufficient to correctly classify an instance. This claim is justified with empirical experiments.

Fig. 4.3 shows the achieved accuracy of the algorithms with different number of the irrelevant attributes. This data set contains 10 % classification noise, and 10 % of the all attribute values are unknown. The number of irrelevant attributes changes from 0 to 20.

The GA-CFP achieved robust accuracy for all cases. This shows the power of the hybrid approach. On the other hand, CFP and C4.5 results are fluctuated. This is due to the randomness of the classification noise, and values of the irrelevant attributes. Results of the CFP changes about 2 %. However, results of the C4.5 changes about 5 %.

4.2.2.2 Sensitivity of the CFP to its Domain Dependent Parameters

This section presents the performance of the CFP algorithm, with different settings of the domain dependent parameters, namely weight adjustment rate (Δ), generalization limits (D_f's), and confidence threshold (CT). The performance is measured in terms of achieved accuracy and memory requirement of the CFP.

Fig. 4.4 shows the achieved accuracy of the CFP with different values of the generalization limit and classification noise levels. CFP performed poorly with the zero generalization limit (no generalization). Accuracy of the CFP reaches its optimum with generalization limit is between 2 and 4 (Fig. 4.4). CFP performed worse for both extreme cases (no generalization ($D_f = 0$) and maximum generalization ($D_f = 10$)). However, CFP performs better

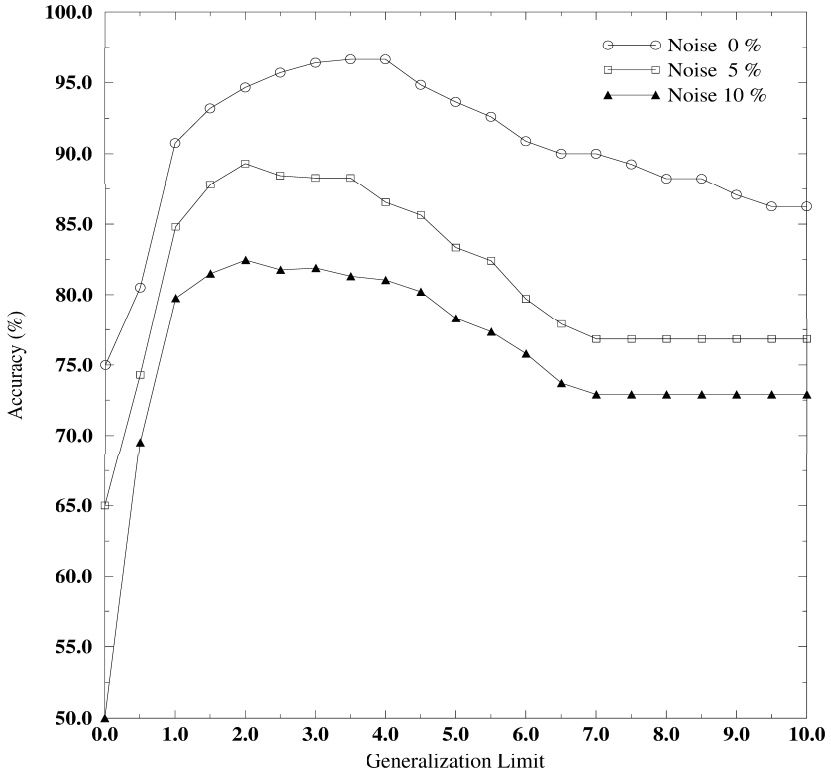

Figure 4.4. Effects of the generalization limit to the accuracy of the CFP

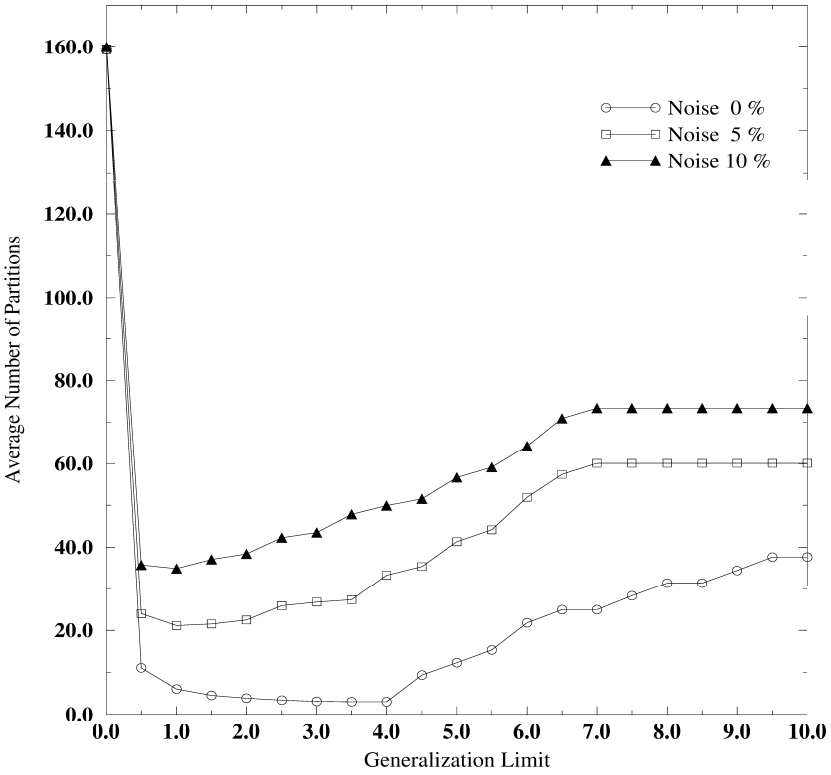

Figure 4.5. Effects of the generalization limit to the memory requirement of the CFP

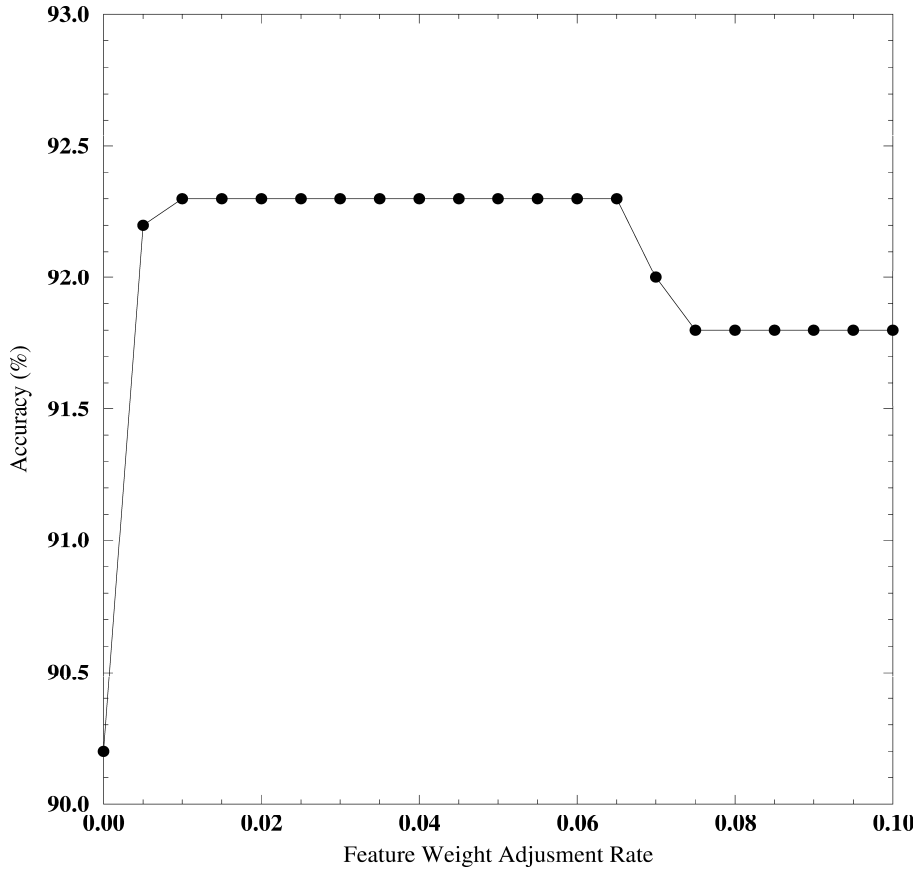

Figure 4.6. Effects of the weight adjustment rate to the accuracy of the CFP

with maximum generalization limit than no generalization. Due to the over-generalization, after some value (e.g. 4) performance of the CFP degrades. As seen from Fig. 4.4 with different amount of classification noise the optimum value of the generalization limit changes, since noise causes a change in domain characteristic.

Fig. 4.5 shows the memory requirements of the CFP in terms of average number of partitions versus different values of the generalization limit for several noise levels. An important point is that, memory requirement of the CFP significantly is reduced when the accuracy reaches its optimum. Therefore, when the best setting for generalization limits are found the best performance in terms of both the prediction accuracy and the memory requirements are achieved.

Fig. 4.6 shows the achieved accuracy by the CFP with different weight adjustment rates (Δ). CFP performs better with small values of the Δ. However, with $\Delta = 0$ (all features have equal weight) performance of the CFP degrades. Although we do not know any general rule for the value of Δ, 0.02-0.05 are good values to start. In my experiments with real and artificial data, we have

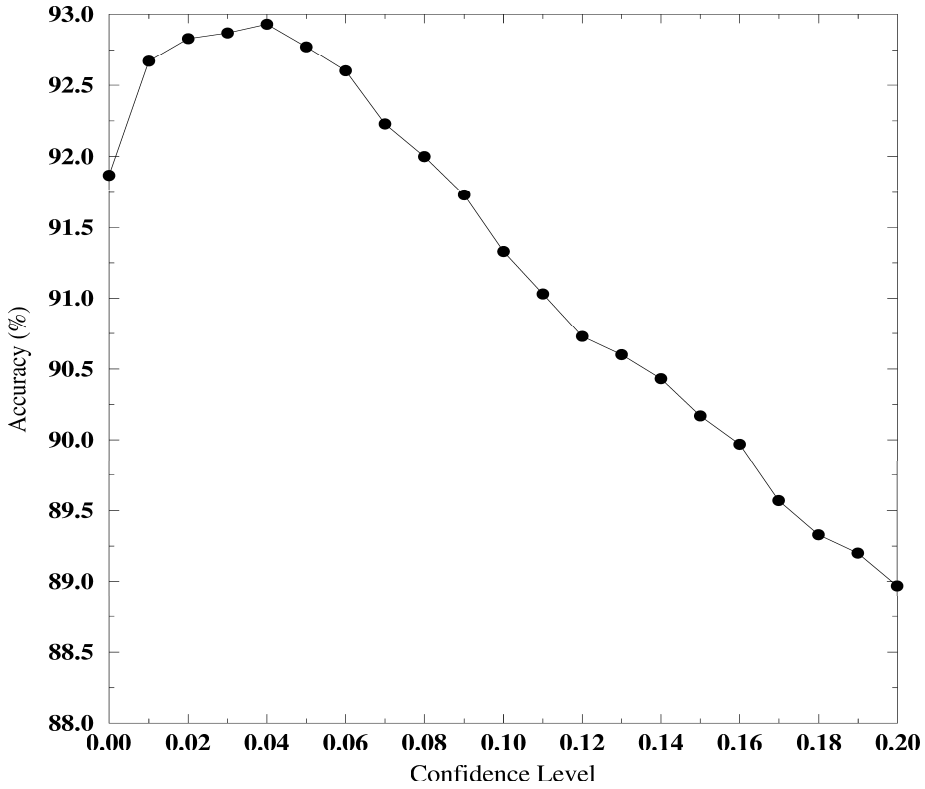

Figure 4.7. Effects of the confidence threshold to the accuracy of the CFP

noticed that the accuracy has a single maximum for different Δ values. This property makes genetic algorithms to be a good mechanism to find the best value for optimum accuracy. After some value of Δ, performance of the CFP does not change, because if some of the feature weights grow too much, these features will dominate the others in the voting process. After this point, increasing the value of Δ will not improve the accuracy (e.g., see Fig. 4.6 for $\Delta > 0.07$).

Fig. 4.7 shows the achieved accuracy of the CFP with different values of the confidence threshold (CT). The best accuracy is achieved for $CT = 0.04$. For higher values of the confidence threshold performance of the CFP degrades, due to the removal of the informative partitions along with noisy ones. In general a confidence threshold between 0.02 and 0.05 gives good results depending on the noise level of the application domain.

Fig. 4.8 shows the memory requirements of the CFP in terms of the average number of partitions, for different values of the confidence threshold. For higher values of the confidence threshold memory requirements of the CFP decreases. Because CFP removes unrepresentative partitions (whose representativeness value is very small compare to the observed frequency of their class), only the

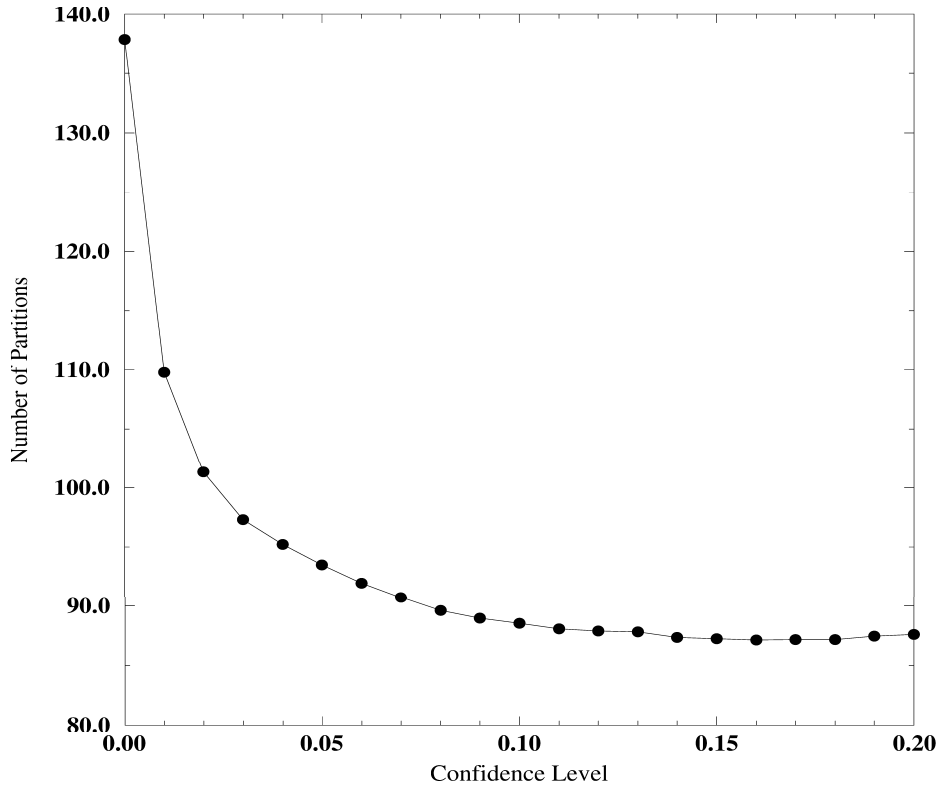

Figure 4.8. Effects of the confidence threshold to the memory requirement of the CFP

representative partitions are left in the concept description.

Fig. 4.9 shows the achieved accuracy of the CFP with different values of the confidence threshold on a noisy data. CFP removes partitions more aggressively for higher values of the confidence level. In Fig. 4.9, for noise level greater than 20 %, confidence threshold of 0.05 causes accuracy degradation. For $CT = 0$ (no partition removing), $CT = 0.01$, and $CT = 0.02$, CFP achieved nearly equal accuracy. An important observation is that, memory requirement (Fig. 4.10) of the CFP is significantly (about 45 %) reduced for $CT = 0.01$ and $CT = 0.02$. This is an expected result since increasing the noise level causes construction of many unrepresentative partitions. These partitions are removed from concept descriptions. Consequently, percentage of the noise in the concept description decreases.

Fig. 4.11 shows the achieved accuracy of the CFP with different values of confidence threshold on a domain which has many irrelevant attributes. This data set contains 10 % classification noise, and 10 % of the attribute values are unknown. The number of irrelevant attributes changes from 0 to 20. The fluctuations on the accuracy are due to the randomness of the values of the irrelevant attributes.

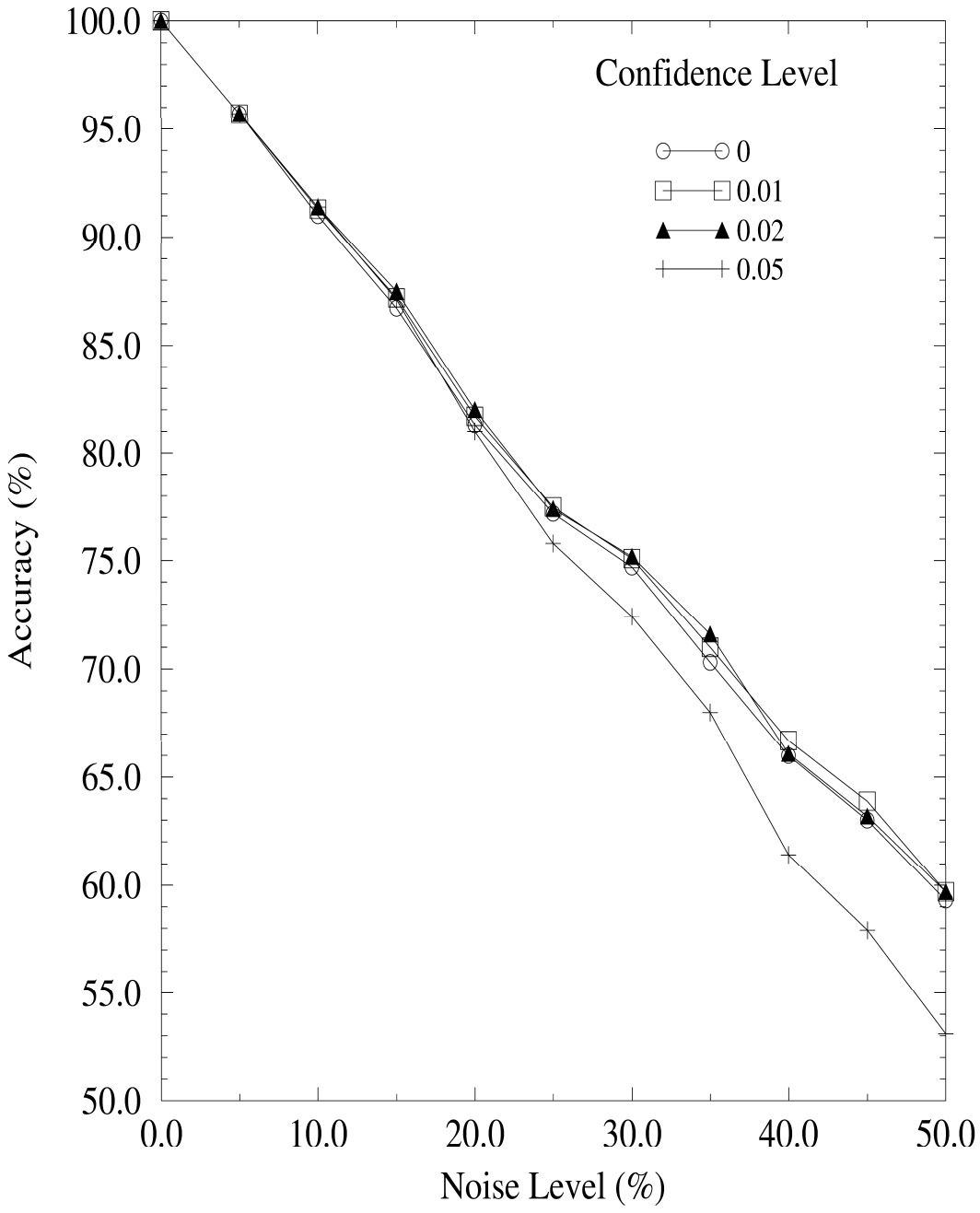

Figure 4.9. Effects of the confidence threshold to the accuracy of the CFP, on noisy domains

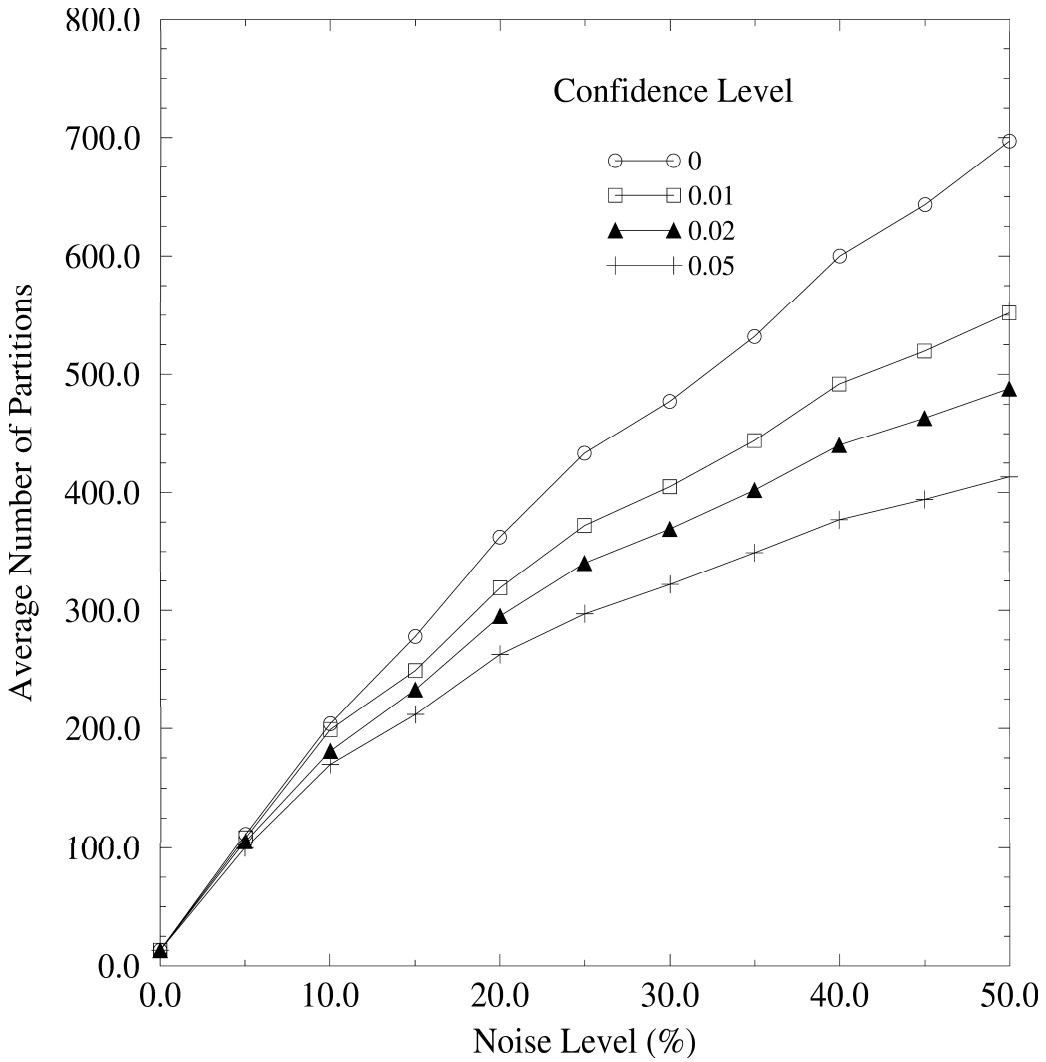

Figure 4.10. Effects of the confidence threshold to the memory requirement of the CFP, on noisy domains

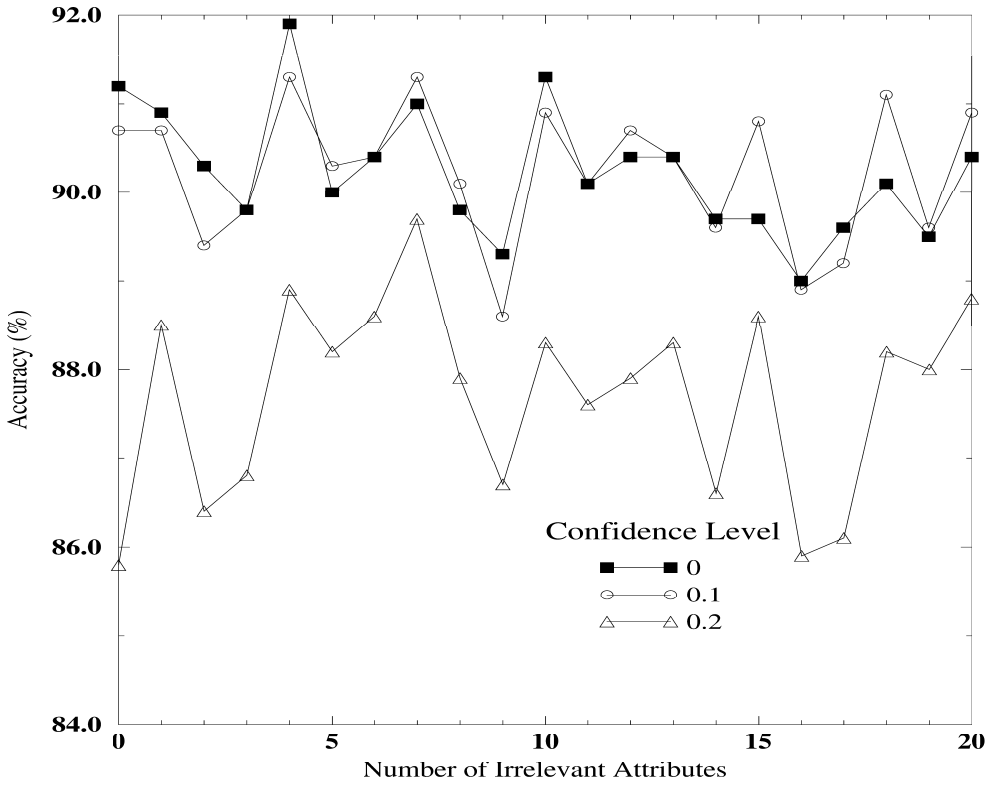

Figure 4.11. Effects of the confidence threshold to the accuracy of the CFP, on a domain with many irrelevant attributes

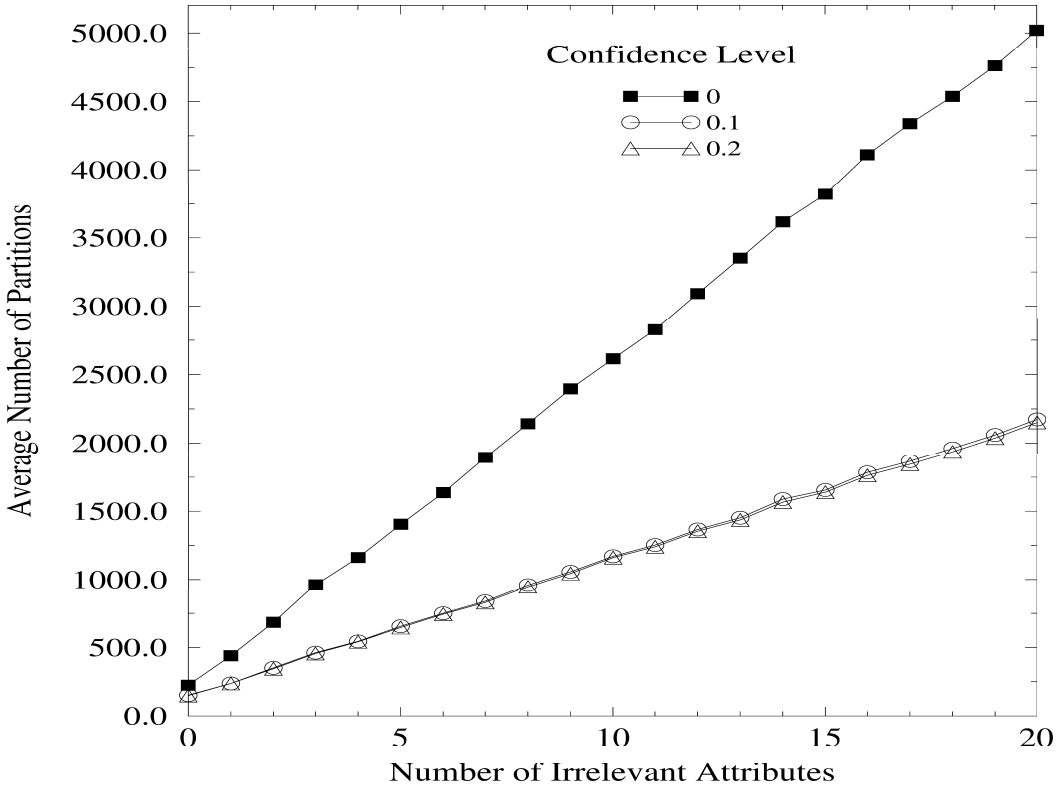

Figure 4.12. Effects of the confidence threshold to the memory requirement of the CFP, on a domain with many irrelevant attributes

CFP achieved nearly equal accuracy for confidence thresholds 0 and 0.1. However, for $CT = 0.2$ the CFP achieved about 4 % worse accuracy. On the other hand, CFP achieved significant memory reduction with confidence threshold 0.1 (Fig. 4.12). The memory requirement of the CFP with $CT = 0.1$ and $CT = 0.2$ are almost the same. In both cases, about 50 % reduction in the memory requirement is achieved with respect to zero confidence threshold.

4.2.3 Experiments with Real-world Data Sets

The CFP algorithm has been tested using real[4] and artificial data from various problem domains. The use of real data in these tests provide a measure for the system's accuracy on noisy and incomplete data sets, and most importantly, it allowed comparisons between CFP and other similar systems[5]. Below, each data set is described briefly and experimental results are presented.

In order to see the effect of a genetic algorithm in learning domain dependent parameters of the CFP, we compared the GA-CFP with the regular CFP algorithm. The GA-CFP algorithm was compared with the CFP algorithm, where the feature weights are identical and the generalization limits are set to two extremes. In the first extreme, for all feature D_f's are set to zero disabling any generalization. In the second extreme, for all feature D_f's are set to maximum value $(max_f - min_f)$, resulting in the maximum possible generalization.

Iris Flowers: Iris flowers data set from Fisher [11] consists of four integer-valued features and a particular species of iris. There are three different species (classes): *virginica*, *setosa*, and *versicolor*. The four variables measured were sepal length, sepal width, petal length, and petal width. The data set contains 150 instances. The accuracy of the CFP in Table 4.1 was obtained for $\Delta = 0.015$, $CT = 0$, and D_f's were 0.6, 0.1, 0.4, 0.1, respectively.

The reported results of the EACH [34] and measured results of the C4.5, CFP, and GA-CFP algorithms are shown in Table 4.1. In testing with C4.5 we used the decision trees that are generated after pruning, since these trees performed better than the trees before pruning in general. To allow for proper comparisons, the experimental design used was the same as that used by Salzberg

[4]All real data sets are taken from UCI repository of machine learning databases (ics.uci.edu: pub/machine-learning-databases).

[5]An empirical comparison of the CFP with similar systems will appear in [39].

Table 4.1. Success rates for iris flowers (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	88.0
CFP: $\Delta = 0, D_f = max$	92.7
CFP: $\Delta = 0.015$	96.7
GA-CFP	98.0
EACH	95.3
C4.5	95.3

[34]. For Iris data set the leave-one-out cross-validation accuracy estimation technique has been used.

Fig. 4.13 shows the two-dimensional view of the instances of the iris flowers data set, where X-axis represents *sepal length* and Y-axis represents *sepal width* feature of the iris flowers data set. The figure also shows the constructed partitions respectively. As seen from Fig. 4.13, projection of the concepts to the feature dimensions overlap. Therefore, CFP algorithm generates many partitions next to each other. Consequently, weight of these features are small.

Fig. 4.14 shows another two-dimensional view of the instances of the iris flowers data set, where X-axis represents *petal length* and Y-axis represents *petal width* feature of the iris flowers data set. As seen from the figure projection of the concepts to the feature dimensions do not overlap. Consequently, weight of these feature are greater than other features. Hence, petal length and petal width are the determining features for iris flowers data set according to CFP algorithm. Constructed partitions for iris data set is given in Appendix.

In the Fig. 4.13 and Fig. 4.14 class virginica, setosa, and versicolor are represented by $+$, $-$, and o respectively.

Breast Cancer[6]: Breast Cancer data set contains 273 patient records. All the patients underwent a surgery to remove tumors, all of them were followed up five years later. The objective here is to predict whether or not breast cancer would recur during that five year period. The recurrence rate is about 30 %, and hence such prognosis is important for determining post-operational treatment. The data set contains nine variables that were measured, including both numeric and binary values. The prediction is binary : either the patient did suffer a recurrence of cancer or not.

[6]Stuart Crawford of Advanced Decision Systems provided this data

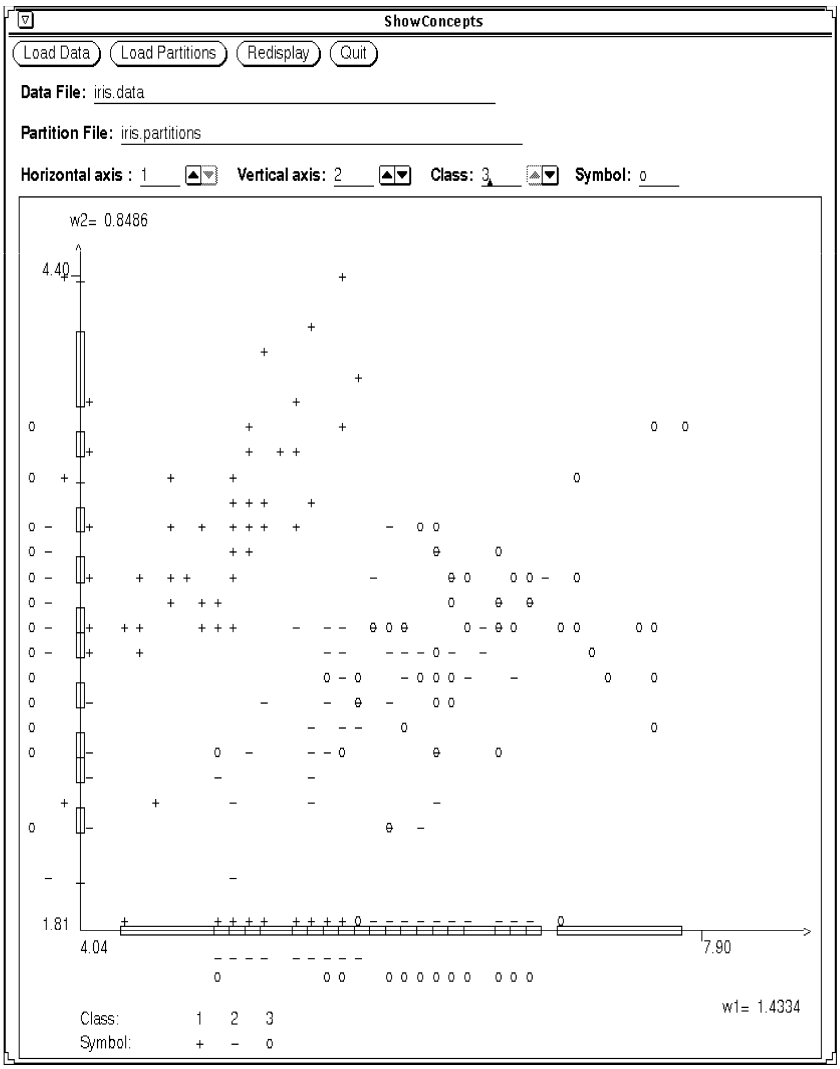

Figure 4.13. 2-D view of the iris data: Sepal length vs. Sepal width

Table 4.2. Success rates for breast cancer data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	73.2
CFP: $\Delta = 0, D_f = max$	74.1
CFP: $\Delta = 0.07$	77.5
GA-CFP	78.7
EACH	77.6
CN2	71.0
C4.5	70.1

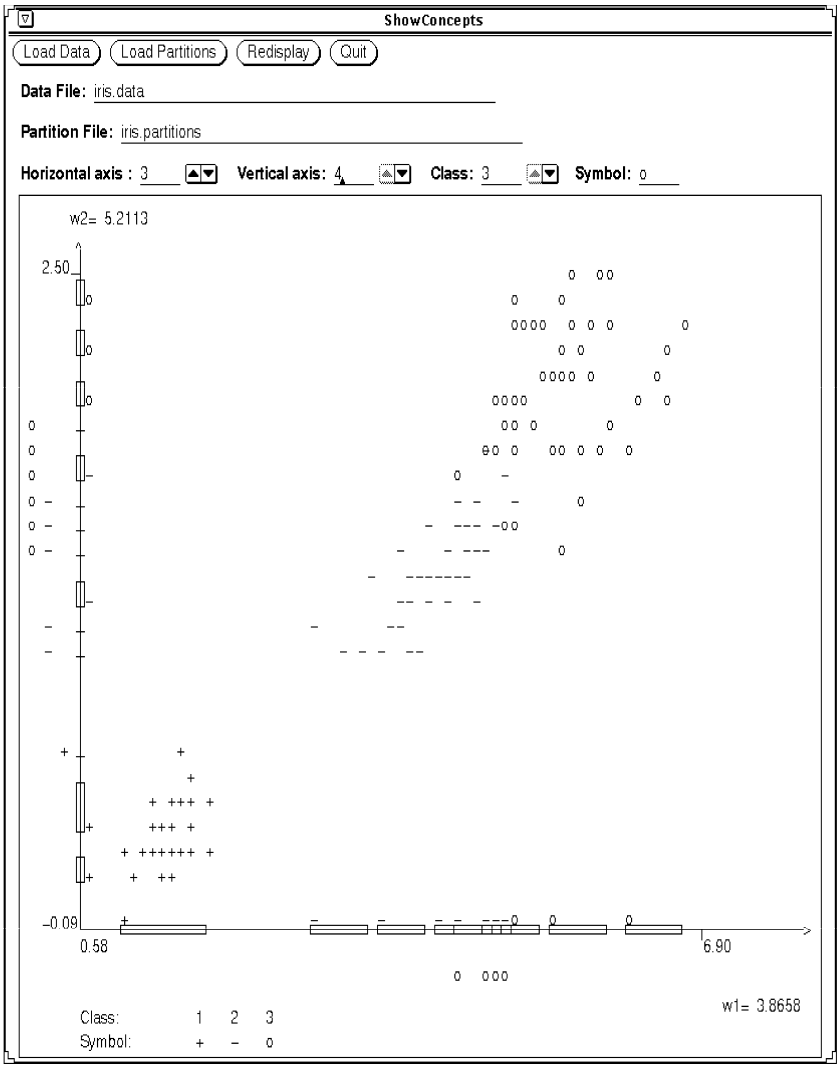

Figure 4.14. 2-D view of the iris data: Petal length vs. Petal width

Table 4.3. Success rates for Hungarian heart disease data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	68.7
CFP: $\Delta = 0, D_f = max$	75.5
CFP: $\Delta = 0.02$	82.3
GA-CFP	91.4
IB1	58.7
IB2	55.9
IB3	80.5
C4	78.2

This data set has been also used to test the CN2 [8] algorithm, which achieved 70-71 % accuracy. The accuracy of the CFP in Table 4.2 was obtained for $\Delta = 0.07$, $CT = 0$ and D_f's were 4, 0, 6, 0.5, 0.1, 0.5, 0.5, 0.5, 4, respectively. The reported results of the EACH [34] and measured results of the C4.5, CFP, and GA-CFP algorithms are shown in Table 4.2. In testing with C4.5 we used the decision trees that are generated after pruning, since these trees performed better than the trees before pruning in general. To allow for proper comparisons, the experimental design used was the same as that used by Salzberg [34]. For Breast Cancer data set, for each trial, 70 % of the examples were randomly chosen for training the rest used in testing. Four different trials were run, and the final results are an average of those trials.

Predicting Heart Disease: The CFP was tested on two widely used medical databases, namely the Cleveland and Hungarian databases. The Cleveland and Hungarian data sets contain heart disease diagnoses collected from the Cleveland Clinic Foundation and Hungarian Institute of Cardiology, respectively. A diagnosis is described by 13 numeric-valued features (e.g. age, fasting blood sugar level etc.). The objective here is to determine whether a patient has a heart disease. The Cleveland data set consists of 303 instances and the Hungarian data set consists of 294 instances. The performance of the CFP algorithm is also compared with the reported accuracy of the instance-based learning algorithms [1]. All results reported in Table 4.3 and Table 4.4 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets.

The accuracy of the CFP for the Hungarian database in Table 4.3 was obtained for $\Delta = 0.02$, $CT = 0$, and D_f's were 1.3, 0, 0.2, 19.1, 36.1, 0, 0.6, 22.9, 0, 0.6, 0.03, 0.1, 0.9, respectively. The accuracy for the Cleveland

Table 4.4. Success rates for Cleveland heart disease data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	77.6
CFP: $\Delta = 0, D_f = max$	75.5
CFP: $\Delta = 0.025$	84.0
GA-CFP	94.3
IB1	75.2
IB2	69.6
IB3	73.8
C4	70.7

Table 4.5. Success rates for waveform data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	53.4
CFP: $\Delta = 0, D_f = max$	62.0
CFP: $\Delta = 0.02$	76.0
GA-CFP	86.5
IB1	75.2
IB2	69.6
IB3	73.8
C4	70.7

database in Table 4.4 was obtained for $\Delta = 0.025$, $CT = 0$, and D_f's were 7, 0.3, 0.4, 9, 11, 0.3, 0.4, 7, 0.3, 0.4, 0.4, 0.4, 0.4, respectively.

Classifying Waveforms: The waveform data set is artificial and consists of 21 numeric-valued features, which have values between 0 and 6. There are three different types of waveforms and they are equally distributed. The objective here is to determine the type of a given waveform. Each feature includes noise (with mean 0 and variance 1). Out of 800 instances in the data set, 300 were used in the training.

The accuracy of the CFP for the waveform database in Table 4.5 was obtained for $\Delta = 0.02$ and $CT = 0$. Generalization limits for the waveform database were 0.2, 0.3, 0.6, 0.5, 0.8, 0.9, 1.0, 0.9, 0.6, 0.6, 0.7, 0.6, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.5, 0.5, 0.4, respectively. All results reported in Table 4.5 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets.

Table 4.6. Success rates for congressional voting database (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	88.9
CFP: $\Delta = 0, D_f = 2$	65.6
CFP: $\Delta = 0.04 D_f = 0$	95.5
GA-CFP	96.2
IB1	91.8
IB2	90.9
IB3	91.6
C4	95.5

Congressional Voting Database: The Congressional Voting database is linearly separable. It contains the (known) voting records of the members of the United States House of Representatives during the second session of 1984. It is described by 16 boolean attributes and has 288 missing values among the 435 instances. 350 of the voting records are used in training and remaining 85 are used in testing. The value to predict is the political party of a member (Democrat or Republican) given their voting record.

All results reported in Table 4.6 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets. The accuracy for the voting database in Table 4.6 was obtained for $\Delta = 0.04$, $CT = 0$, and all the D_f's were zero.

Classifying Glasses: This data set consists of attributes of glass samples taken from the scan of an accident.[7] Each of the 214 examples is a member of one of six classes. There are nine features. In comparing GA-CFP with GA-WKNN, we used cross-validation accuracy estimation technique. Each data set was divided into five disjoint partitions. The only constraint on otherwise random partitioning was that classes be represented equally in each partition. We generated five training/test sets for each data set. Four-fifth of the data were used for training and the remaining fifth was used for testing. The results are shown in Table 4.7. The accuracy for the glass database in Table 4.7 was obtained for $\Delta = 0.03$ and $CT = 0$. Generalization limits for the glass database were 1, 10, .4, 0.5, 2, 0.4, 8, 2, and 1 respectively.

Pima Indians Diabetes Database: This data set contains diabetes diseases

[7]Collected by B. German of the Home Office Forensic Service, Aldermaston, Reading, Berkshire, UK.

Table 4.7. Success rates for glass data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	44.3
CFP: $\Delta = 0, D_f = max$	58.1
CFP: $\Delta = 0.03$	55.7
GA-CFP	71.7
GA-WKNN	62.2

Table 4.8. Success rates for Pima Indians diabetes database (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	67.0
CFP: $\Delta = 0, D_f = max$	66.5
CFP: $\Delta = 0.025$	70.7
GA-CFP	74.6

collected from National Institute of Diabetes and Digestive and Kidney Diseases. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The population lives near Phoenix, Arizona, USA. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The data set contains records of 768 patients with 8 features. 576 instances are used in training the remaining 192 instances are used in testing. The objective of this data set is to determine whether result of the test is positive or negative for diabetes. All results reported in Table 4.8 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets.

Ionosphere database: David Aha briefly investigated this database. He found that nearest neighbor attains an accuracy of 92.1 %, that Ross Quinlan's C4 algorithm attains 94.0 % (no windowing).

The radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. *Good* radar returns are those showing evidence of some

Table 4.9. Success rates for ionosphere database (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	83.2
CFP: $\Delta = 0, D_f = max$	80.2
CFP: $\Delta = 0.015$	87.6
GA-CFP	92.1
C4	94.0
KNN	92.1

type of structure in the ionosphere. *Bad* returns are those that do not; their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

This data set contains 351 instances. 200 of the instances are used in training the rest are used in testing. Each instance consist of 34 continuous-valued features. It is a binary (good or bad) classification problem. All results reported in Table 4.9 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets. The accuracy for the ionosphere database in Table 4.9 was obtained for $\Delta = 0.015$ and $CT = 0$, and $D_f = 0.05$.

Liver disorders data set: This data set contains 345 instances and collected by BUPA Medical Research Ltd. Each instance constitutes the record of a single male individual. There are 6 attributes and the first 5 variables are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption. The last attribute presents drinks number of half-pint equivalents of alcoholic beverages drunk per day. The objection of this data set is to determine whether patient has liver disorders or not. 276 of the instances are used in training the remaining 69 are used in testing. All results reported in Table 4.10 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets. The accuracy for the liver disorders database in Table 4.10 was obtained for $\Delta = 0.01$ and $CT = 0$, and D_f's were 7, 7, 7, 7, 7, and 0.5 respectively.

Table 4.10. Success rates for liver disorders database (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	56.2
CFP: $\Delta = 0, D_f = max$	57.3
CFP: $\Delta = 0.025$	63.0
GA-CFP	68.2

Table 4.11. Success rates for wine classification data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	77.0
CFP: $\Delta = 0, D_f = max$	87.1
CFP: $\Delta = 0.01$	91.6
GA-CFP	95.0

Wine recognition data set: This data is provided by Institute of Pharmaceutical and Food Analysis and Technologies. The classes are separable. The leave-one-out technique is used. In a classification context, this is a well posed problem with "well behaved" class structures. A good data set for first testing of a new classifier, but not very challenging. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultures. The analysis determined the quantities of 13 constituents found in each of the three types of wines. Data set contains 178 instances and all attributes are continuous-valued. The accuracy for the wine database in Table 4.11 was obtained for $\Delta = 0.01$ and $CT = 0$. Generalization limits for the wine database were 0.9, 8, 3.4, 0.6, 11, 6, 6, 1.5, 1, 2.5, 1.7, 1.6, and 16 respectively.

Predicting patient thyroid type: This data set is provided by James Cook University. Five laboratory test results (continuous-valued) are used to predict whether a patient's thyroid to the class euthyroidism, hypothyroidism or hyperthyroidism. The diagnosis (the class label) was based on a complete medical record, including anamnesis, scan etc. There is 215 patient records in the data set each of which has one of the three classes (normal, hyperthyroid, or hypothyroid) and 80 % of the records are used in training. All attributes are continuous.

All results reported in Table 4.12 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from

Table 4.12. Success rates for thyroid data (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	80.4
CFP: $\Delta = 0, D_f = max$	89.2
CFP: $\Delta = 0.025$	90.8
GA-CFP	95.4

Table 4.13. Success rates for mushroom database (%)

Algorithm	Success Rate (%)
CFP: $\Delta = 0, D_f = 0$	87.6
CFP: $\Delta = 0, D_f = max$	68.52
CFP: $\Delta = 0.01 D_f = 0$	98.5
GA-CFP	99.2
STAGGER	95.0

the data sets. The accuracy for the thyroid database in Table 4.12 was obtained for $\Delta = 0.025$ and $CT = 0$. Generalization limits for the thyroid database were 4, 4, 0.1, 0.1, and 3.5 respectively.

Mushroom Database: Mushroom records were drawn from The Audubon Society Field Guide to North American Mushrooms. STAGGER [35] asymptoted to 95 % classification accuracy after reviewing 1000 instances. This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. There is no simple rule for determining the edibility of a mushroom. The database contains 8124 instances with 22 nominally-valued features. 1000 instances are used in training the remaining 7124 are used in testing. All results reported in Table 4.13 were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets. The accuracy for the mushroom database in Table 4.13 was obtained for $\Delta = 0.01$, $CT = 0$, and $D_f = 0$.

In these experiments we noticed that the performance was not sensitive to the small changes in the D_f settings. For binary-valued attributes distance parameter was set to zero for no generalization. The feature weight adjustment rate and the generalization limits are domain dependent. In these experiments

their values (for CFP) are determined by trial and error, separately for each application domain. It is clear from these experiments that the genetic algorithm can determine a very good set of domain dependent parameters of CFP. Even the result of the CFP with settings that are found by trial and error are good enough compare to the other similar algorithms.

Chapter 5

Conclusion

In this thesis I have presented a new methodology of learning based on feature partitioning, called CFP. It is an inductive, incremental and supervised learning method. The CFP learns a partitioning of values for each feature of the application domain. The CFP algorithm is applicable to domains, where each feature, independent of other features, can be used to classify the instances.

This approach is a variant of algorithms that learn by projecting into one feature dimension at a time. For example, ID3 learns in that greedy manner while building a conjunction. The novelty of CFP is that it retains a feature-by-feature representation and uses a voting scheme in categorization. Algorithms that learn by projecting into one dimension at a time are limited in their ability to find complex concepts.

The CFP makes significant modifications to the exemplar-based learning algorithms. The analysis of the CFP shows that, compared to many other similar algorithms, it is applicable to a large class of concepts, and requires small number of examples and a small amount of memory to learn a given concept. It is also proved that the CFP algorithm has a low training complexity.

Another important improvement is the natural handling of unknown attribute values. Most of the systems use *ad hoc* methods for handling unknown attribute values [15, 28, 29]. Since the value of each attribute is handled separately, attributes with unknown values are simply ignored by the CFP.

The CFP will clearly fail in some cases. For example, if the projection of concepts on an axis are overlapping each other, the CFP constructs many

partitions of different classes next to each other. In that case, the accuracy of classification depends on the observed frequency of the concepts.

The CFP uses feature weights to cope with irrelevant attributes. Introducing feature weights protects the algorithm's performance, when an application domain has irrelevant attributes. The idea of feature weights have also been used successfully in other similar systems [22, 34]. In the CFP the feature weights are dynamically adjusted according to the global weight adjustment rate (Δ), which is an important parameter for the predictive accuracy of the algorithm. Another important component of the CFP is the generalization limit for each attribute, which controls the generalization process. The confidence threshold is introduced into the CFP to reduce the percentage of the noise in the concept descriptions. The confidence threshold controls the removal of partitions from the concept description. The confidence threshold is used to improve the accuracy of the CFP and also to reduce the memory requirements. Empirical results justify this claim.

The weight adjustment rate, generalization limits, and confidence threshold are domain dependent parameters of the CFP, and their selection affects the performance of the algorithm. Determining the best values for these parameters is an optimization problem for a given domain. In GA-CFP a genetic algorithm is used to find a good setting of these parameters. The GA-CFP is a hybrid system, which combines optimization capability of genetic algorithm with classification capability of the CFP algorithm. The genetic algorithm is used to determine the domain dependent feature weights, generalization limits, and confidence threshold. Although the results of the GA-CFP algorithm are better than other classification systems, the use of genetic algorithm is costly. This is because the computation of the fitness function requires the execution of the CFP algorithm several times due to the used methods. However, an important characteristic of the parameters of the CFP algorithm is that these parameters are domain dependent. Therefore, the genetic algorithm can be used with only a portion of all the data available. Then, the CFP algorithm can be used with settings that are learned by the genetic algorithm. This claim was empirically justified on the iris flowers data set.

Partition is the basic unit of representation in the CFP algorithm. Each partition represents two (one, if lower and upper values of a partition is equal) parallel surfaces (hyperplanes) in feature space, which are orthogonal to the

axis of the partition and parallel to all other axes. Consequently, the regions constructed by the CFP are disjoint hyperrectangles. Since CFP retains feature-by-feature representation, projection of concepts will determine the applicability of the CFP to a domain. The CFP is not applicable to domains where all of the concept projections overlap, or domains in which concept descriptions are nested. In other words, it is applicable to domains where each feature can contribute the classification of an instance independent of others. In fact, this is the nature of the most real-world data sets. For example, in stock market applications, there are dozens of factors influencing the market. However, domain experts predict the future trend by just looking at some key variables [40].

Bibliography

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6:37–66, 1991.

[2] H. Almuallim and T. G. Dietterich. Learning with Many Irrelevant Features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547–552, 1991.

[3] D. Angluin and P. Laird. Learning from Noisy Examples. *Machine Learning*, 2:343–370, 1988.

[4] D. Anthony, J. Barham, and D. Taylor. The Use of Genetic Algorithms to Learn the Most Appropriate Inputs to a Neural Network. In *Proceedings of AINN*, pages 223–226, 1990.

[5] A. Blumer, A. EhrenFeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[6] J. G. Carbonell, editor. *Machine Learning: Paradigms and Methods.* The MIT Press, 1990.

[7] J. Catlett. On Changing Continuous Attributes Into Ordered Discrete Attributes. In *Proceedings of European Workshop on Machine Learning*, pages 164–178, 1991.

[8] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3:261–283, 1989.

[9] G. Dejong and R. Mooney. Explanation-Based Learning: An Alternative View. *Machine Learning*, 1:145–176, 1986.

[10] J. L. Elman. Distributed Representations, Simple Recurrent Networks, and Grammatical Structure. *Machine Learning*, 7:195–225, 1991.

[11] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7:179–188, 1936.

[12] J. M. Fitzpatrick and J. J. Grefenstette. Genetic Algorithms in Noisy Environments. *Machine Learning*, 3:101–120, 1988.

[13] S.I. Gallant. Connectionist Expert Systems. *Communications of the ACM*, 31(2):152–169, 1988.

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Maryland, 1989.

[15] J. W. Grzymala-Busse. On the Unknown Attribute Values in Learning from Examples. In *Proceedings of Sixth International Symposium Methodologies for Intelligent Systems*, pages 368–377, October 1991.

[16] H. A. Güvenir and İ. Şirin. The Complexity of The CFP, a Method for Classification Based on Feature Partitioning. In *Lecture Notes in Artificial Intelligence*, 1993. (to appear).

[17] H. A. Güvenir and İ. Şirin. A Genetic Algorithm for Classification by Feature Partitioning. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 543–548, 1993.

[18] D. Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177–221, 1988.

[19] K. De Jong. Learning with Genetic Algorithms: An Overview. *Machine Learning*, 3:121–138, 1988.

[20] M. J. Kearns. *The Computational Complexity of Machine Learning.* The MIT Press, 1989.

[21] T. M. Mitchell R. Keller and S. Kedar-Cabelli. Explanation-Based Generalization: A Unifying View. *Machine Learning*, 1:47–80, 1986.

[22] J. D. Kelly and L. Davis. A Hybrid Genetic Algorithm for Classification. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 645–650, 1991.

[23] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. *Machine Learning*, 2:285–318, 1988.

[24] H. Lounis and G. Bisson. Evaluation of Learning Systems: An Artificial Data-Based Approach. In *Proceedings of European Working Session on Learning*, pages 463–481, 1991.

[25] D. L. Medin and M. M. Schaffer. Context Theory of Classification Learning. *Psychological Review*, 85:207–238, 1978.

[26] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, California, 1991.

[27] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[28] J. R. Quinlan. Decision Trees as Probabilistic Classifiers. In *Proceedings of Fourth International Workshop on Machine Learning*, pages 31–37, June 1987.

[29] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, California, 1993.

[30] L. Rendell. A New Basis for State-Space Learning Systems and Successful Implementation. *Artificial Intelligence*, 20:369–392, 1983.

[31] L. Rendell. A General Framework for Induction and a Study of Selective Induction. *Machine Learning*, 1:177–226, 1986.

[32] L. Rendell and H. Cho. Empirical Learning as a Function of Concept Character. *Machine Learning*, 5:267–298, 1990.

[33] S. Salzberg. Distance Metrics for Instance-Based Learning. In *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems*, pages 399–408, 1991.

[34] S. Salzberg. A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6:251–276, 1991.

[35] J. C. Schlimmer and R. H. Granger. Incremental Learning from Noisy Data. *Machine Learning*, 1:317–354, 1986.

[36] N. N. Schraudolph and J. J. Grefenstette. *A User's Guide to* GAUCSD *1.4*, July 1992.

[37] J. W. Shavlik. A Framework for Combining Symbolic and Neural Learning. Technical Report 1123, Computer Science Department, University of Wisconsin-Madison, 1992.

[38] İ. Şirin and H. A. Güvenir. A Classification Algorithm Based on Feature Partitioning. In *Proceedings of the Second TAINN Symposium*, pages 283–288, 1993.

[39] İ. Şirin and H. A. Güvenir. Empirical Evaluation of the CFP Algorithm. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, 1993. (to appear).

[40] İ. Şirin and H. A. Güvenir. Prediction of Stock Market Index Changes. In *Adaptive Intelligent Systems*, pages 149–159, 1993.

[41] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Chapter 6

Appendix

Constructed partitions for iris flowers data set are given below. The first two numbers represent the lower and upper values of the partition, the third number represents the class value of the partition, and the last number is the representativeness value of that partition. Classes are enumerated so that classes 1, 2, and 3 represent *virginica*, *setosa*, and *versicolor*, respectively.

The CFP constructed 42 partitions for *sepal length* and weight of this feature was 1.433420.

```
4.30  4.90   1  19.60
4.90  4.90   3   1.00
4.90  4.90   2   1.00
4.90  5.00   1   3.27
5.00  5.00   2   2.00
5.00  5.10   1   3.27
5.10  5.10   2   1.00
5.10  5.20   1   3.27
5.20  5.20   2   1.00
5.20  5.40   1   6.53
5.40  5.40   2   1.00
5.40  5.50   1   3.27
5.50  5.50   2   5.00
5.50  5.60   1   3.27
5.60  5.60   3   1.00
5.60  5.60   2   5.00
```

```
5.60  5.70   1   3.27
5.70  5.70   3   1.00
5.70  5.70   2   5.00
5.70  5.80   1   3.27
5.80  5.80   2   3.00
5.80  5.90   3   4.00
5.90  6.00   2   2.36
6.00  6.00   3   2.00
6.00  6.10   2   2.36
6.10  6.10   3   2.00
6.10  6.20   2   2.36
6.20  6.20   3   2.00
6.20  6.30   2   2.36
6.30  6.30   3   6.00
6.30  6.40   2   2.36
6.40  6.40   3   5.00
6.40  6.50   2   2.36
6.50  6.50   3   4.00
6.50  6.70   2   4.73
6.70  6.70   3   5.00
6.70  6.80   2   2.36
6.80  6.80   3   2.00
6.80  6.90   2   2.36
6.90  6.90   3   3.00
6.90  7.00   2   2.36
7.10  7.90   3  12.00
```

The CFP constructed 33 partitions for *sepal width* and weight of this feature was 0.848582.

```
2.00  2.00   2   1.00
2.20  2.20   3   1.00
2.20  2.30   2   5.00
2.30  2.30   1   1.00
2.40  2.50   2   5.00
2.50  2.50   3   4.00
2.50  2.60   2   5.00
2.60  2.60   3   2.00
```

```
2.70  2.70   3   4.00
2.70  2.80   2  11.00
2.80  2.80   3   8.00
2.90  2.90   3   2.00
2.90  2.90   2   7.00
2.90  3.00   1   6.00
3.00  3.00   3  12.00
3.00  3.00   2   8.00
3.00  3.10   1   6.00
3.10  3.10   3   4.00
3.10  3.10   2   3.00
3.20  3.20   3   5.00
3.20  3.20   2   3.00
3.20  3.30   1   7.00
3.30  3.30   3   3.00
3.30  3.30   2   1.00
3.40  3.40   3   2.00
3.40  3.40   2   1.00
3.40  3.50   1  14.00
3.60  3.60   3   1.00
3.60  3.60   1   2.00
3.70  3.80   1   7.00
3.80  3.80   3   2.00
3.90  4.20   1   5.00
4.40  4.40   1   1.00
```

The CFP constructed 15 partitions for *petal length* and weight of this feature was 3.865773.

```
1.00  1.90   1  49.00
3.00  3.60   2   6.00
3.70  4.20   2  17.00
4.30  4.50   2   6.75
4.50  4.50   3   1.00
4.50  4.80   2  10.13
4.80  4.80   3   2.00
4.80  4.90   2   3.37
4.90  4.90   3   3.00
```

4.90	5.00	2	3.37
5.00	5.00	3	3.00
5.00	5.10	2	3.37
5.10	5.40	3	13.00
5.50	6.10	3	22.00
6.30	6.90	3	6.00

The CFP constructed 19 partitions for *petal width* and weight of this feature was 5.211325.

0.10	0.20	1	33.00
0.30	0.50	1	15.00
0.60	0.60	1	1.00
1.00	1.00	2	7.00
1.10	1.10	2	3.00
1.20	1.30	2	18.00
1.40	1.40	3	1.00
1.40	1.40	2	7.00
1.50	1.50	3	2.00
1.50	1.50	2	10.00
1.60	1.60	3	1.00
1.60	1.60	2	3.00
1.70	1.70	3	1.00
1.70	1.80	2	2.00
1.80	1.80	3	11.00
1.90	1.90	3	5.00
2.00	2.10	3	12.00
2.20	2.30	3	11.00
2.40	2.50	3	6.00

Chapter 7

Glossary

This glossary contains machine learning terms that are used in this thesis. The angle brackets "<>" indicate that the term used in a definition is an itself entry in the glossary.

Adaptive Systems: Control systems or pattern recognition systems that achieve desired performance by adjusting their internal parameters.

Attribute: A variable or one-argument descriptor used to characterize an object or a process. For example, the *color* (of an object) or the *duration* (of a process) are attributes.

Classification: A process of assigning to an instance its appropriate class label. Many diagnostic problems are basically problems of classification.

Concept Acquisition: See <Learning from Examples>.

Concept Description: A symbolic data structure defining a concept describing the class of all known instances of the concept.

Crossover: This is the crossing procedure in <Genetic algorithms> where by portions of existing rules are cut up and spliced together to form a new rule.

Decision Tree: A tree encoding a set of tests to classify a collection of objects into fixed categories according to predetermined features of the object.

Exemplar-Based Learning: A kind of <Learning from Examples> in which the <Concept Description> is constructed from the examples themselves.

Feature: See <Attribute>.

Feature Space: In a <Learning from Examples> problem, space defined by <Attributes>.

Generalization: Extending the scope of a concept description to include more instances (the opposite of <Specialization>).

Genetic Algorithms: A model of the process of natural selection, in which better adapted parents are more likely to survive and pass on their characteristic to their children.

Incremental Learning: Multistage learning in which knowledge learned at one stage is modified to accommodate new facts provided in subsequent stages.

Inductive Inference: A mode of reasoning that starts with some assertions, e.g., specific observations, and concludes with more general and plausible assertions.

Inductive Learning: Learning by drawing <Inductive Inferences> from facts and observations obtained from a teacher or environment.

Learning from Examples: Inferring a general <Concept Description> from examples and (optionally) counterexamples of that concept. This is a form of <Inductive Learning>.

Machine Learning: A subdomain of artificial intelligence concern with developing computational theories of learning and constructing machines with learning capability.

Mutation: One of the standard genetic operator of <Genetic Algorithms>. Arbitrarily alters one more components of a selected structure.

Negative Example: In <Learning from Examples> a counterexample of a concept that may bound the scope of <Generalization>.

Neural Network: A network of neuron-like elements that performs some simple logical functions.

Parameter Adjustment: Changing the relative weight of different terms in a mathematical expressions, as a function of credit (blame) for past successes (failures); a kind of incremental curve fitting.

Positive Example: In <Learning from Examples> an example or instance of a concept to be learned.

Probably approximately correct (PAC) model: A computational learning model. An algorithm A is PAC learnable if it can construct description of of unknown target concept with high confidence, that is good approximation of the concept, independent of probability distribution of examples.

Specialization: Narrowing the scope of a <Concept Description> thus reducing sets of instances it describes (opposite of <Generalization>).

Supervised Learning: Examples are pre-classified in <Learning from Examples>.

Training Set: A database of examples, which are pre-classified, which is given to a learning system to enable it to construct the <concept description> By contrast, a test set contains data in a similar form which were not used during training phase.