An Experimental Study of Concept Formation

John H. Gennari

Department of Information and Computer Science
University of California, Irvine, CA 92717

Technical Report 90-26

Sept. 14, 1990

Dissertation
submitted in partial satisfaction of the requirements for the degree of
Doctor of Philosophy in Information and Computer Science

Contents

List of Tables

List of Figures

Acknowledgements

The work in this thesis grew out of the ICARUS project and was supported by members of the ICARUS research group. Of these, I have worked most closely with Pat Langley, Wayne Iba, Kevin Thompson, and John Allen.

Pat also deserves additional recognition for two roles in this research. first, he has always been a font of research ideas. Although not all of these have proved feasible, he has taught me the ability to be creative about research directions and problem solving. Second, he has a strong ability to edit and teach clarity in writing. I believe this is an invaluable skill for any technical discipline.

Finally, I must give credit to the research environment at the Department of Information and Computer Science at UCI. In part, all of my fellow graduate students deserve some credit for this effort. Without late-night improptu research meetings, and a free exchange of ideas (and criticisms), this dissertation could never have been completed.

This research was supported in part by Contract MDA 903-85-C-0324 from the Army Research Institute.

Abstract of the Dissertation

An Experimental Study of Concept Formation

by

John Hewson Gennari

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 1990

Pat Langley, Chair

This thesis presents a careful experimental examination of the task of *concept formation*. This learning task involves acquiring concepts or classes that group observed instances. These classes can then be used for the recognition or classification of unseen instances and for predicting values of unknown attributes. To carry out this task, I present CLASSIT, a robust, general-purpose system for incremental, unsupervised concept learning. In order to test the system, the thesis reports on a variety of experimental results. There are two broad categories for the experiments in this thesis: experiments that test the ability of the system with different input data, and experiments that hold the input data constant and test various components of the system by changing the system itself. The latter include parametric studies and comparative studies with cluster analysis methods. The largest modification I suggest for the basic concept formation algorithm is the addition of an *attention* mechanism. This lets the system focus attention on only the most important features (or attributes) of an instance. With attention, instances can be more quickly recognized, since the less important attributes are simply ignored and treated as missing. Finally, although the accomplishments of the CLASSIT system are modest, I conclude by discussing the potential for using the system as part of a larger cognitive architecture.

Chapter 1
An Approach to Concept Formation

Suppose an explorer has just landed in an alien jungle on an unknown planet. To survive, he must organize the objects and information he perceives into useful concepts or categories. This process, *concept formation*, is one of the most basic of human learning capabilities. People can learn concepts without external supervision and in the face of a bewildering number of inputs.

This human ability can be abstracted into a very simple clustering task: given a set of instance descriptions, find a set of classes that group those instances. A wide variety of researchers have approached this general task, each with their particular biases and research goals. Before defining more carefully the task for concept formation, I begin this thesis by explicitly presenting my research goals and biases, contrasting these with other approaches to the general clustering problem.

1. Research goals

There are a number of reasons for pursuing research in artificial intelligence, ranging from improving engineering abilities to discovering elegant mathematical theories to finding psychological models of the human mind. Because these reasons are so varied, it is important that researchers make clear the goals and biases of their work. Only then can the research be understood and evaluated in a consistent manner. This research is most closely tied to the field of machine learning; it shares the goals and the assumptions of this paradigm. However, it has also been strongly influenced by other fields, borrowing goals from engineering, cognitive psychology, and cluster analysis.

Machine learning is a subfield of artificial intelligence that focuses on learning systems and algorithms. For this field, learning is the most important aspect of intelligence: the ability of a system to improve is that which distinguishes an adaptive system from a simple database of knowledge. Hence, in order to understand intelligence, we must understand and be able to model the learning process.

In order to model learning, one needs at least a working definition of this term. As with 'intelligence', a precise definition of 'learning' is somewhat controversial (e.g., see Carbonell, 1989, or Dietterich, 1990). For this work, I define *learning* as the ability of a system to acquire knowledge and improve its performance over time. More specifically, I define a *learning task* that specifies the inputs to the learner and the knowledge that is to be acquired, and a *performance task* that describes the ability that is improved over time as a result of learning. This is both a reasonably general definition, and one that allows for a clear definition of success.

As with most work in artificial intelligence, I use a computer system to implement and test my ideas. Hence, I describe an implemented system, CLASSIT, that instantiates my approach to concept formation. As with research in engineering, it is important that the system be rigorously tested with a variety of domains; the system should be robust, rather than special purpose. Therefore, after defining the learning and performance tasks, I include a wide variety of experiments with CLASSIT.

These experiments demonstrate both that learning occurs with a variety of input domains, and that the learning system compares favorably against other methods and variations.

Although CLASSIT is not a model of human learning, its development has been constrained and inspired by research in cognitive psychology. In this paradigm, a computer system is an analogy for an extremely robust learning system: the human learner. Thus, CLASSIT could be used to evaluate and explore models of human intelligence. Although I do not currently have any results to support this analogy, I am interested in investigating human learning. I hope that CLASSIT is psychologically plausible, and that it may easily be modified to be part of a more general architecture for human intelligence.

Finally, I have been strongly influenced by research in cluster analysis. This older field has studied the clustering problem from a more specific, application-oriented perspective. However, although machine learning approaches to concept formation derive from work in cluster analysis (Michalski & Stepp, 1983a), rarely have the two fields been directly compared. This thesis includes a description of the relationship between cluster analysis and machine learning, along with a direct comparison of my system and cluster analysis algorithms.

2. Problem definition

The specific instantiation of the concept formation task as addressed in this thesis is:

- Given: a sequential presentation of instances described by attribute-value pairs,
- Find: a hierarchy of concepts that groups these instances.

In order to make this task more concrete, consider a robot exploring an unknown environment. Assuming that the robot includes a perceptual system that reduces objects to attribute-value pair descriptions, then the robot could use a concept formation system to acquire knowledge about the environment and to organize that knowledge in a hierarchy. As the robot builds a concept hierarchy, it can use this learned information to recognize or classify objects that are similar to previously observed instances. This example demonstrates the learning and performance tasks of the system: it learns a concept hierarchy from the environment, and it recognizes a new instance as a member of some concept in that hierarchy.

There are a number of aspects of this task that differ from other approaches to clustering. First, the agent learns a *hierarchy* of concepts, rather than a simple set or list of classes. This data structure is similar to an *is-a* hierarchy and is inspired by the more general semantic network of Quillian (1967). The concept hierarchy organizes knowledge from very general concepts (near the root of the tree) to very specific concepts (near the leaves of the tree). One important difference between this structure and decision trees (Quinlan, 1986) is that every node in the hierarchy is a concept, rather then a simple decision point.

Second, learning occurs *incrementally*, meaning that one must learn from each instance without re-processing all previous instances. In contrast to cluster analysis and other methods that process a finite set of instances as a batch, one must accept input as a sequence of instance descriptions. This emphasis on incremental learning leads to the integration of learning with performance. Each instance can be used both for learning, as the hierarchy is modified or the concept definitions updated, and for performance, as the system recognizes the instance as a member of some concept.

Third, concept formation occurs in an *unsupervised* manner: instances are clustered into classes without advice from a teacher. In contrast, a supervised learner includes as input a specified class name for each observed instance. Unsupervised learning has an additional level of complexity to its task description: not only must a concept formation system decide which instances belong to which class, but also the number and structure of the classes. This is the most important feature separating this work from research on learning concepts from examples (Winston, 1975; Quinlan, 1986).

Finally, this definition of incremental concept formation leads naturally to a learning algorithm that carries out *incremental hill climbing* (Langley, Gennari & Iba, 1987). Hill climbing is a standard AI search method in which one tries to apply all possible operators, compares the resulting states using an evaluation function, selects the best state, and iterates until no more progress can be made. As we will see in Chapter 3, this description fits the CLASSIT algorithm quite closely.

For this research, concept formation implies that the learning agent is searching through a space of possible concept hierarchies. Since the agent uses a hill-climbing search strategy, only a single knowledge structure may be retained in memory at a time. With each new instance, the system modifies the hierarchy, and hence chooses a single direction to move through the space. The principal disadvantage of this search strategy is that a hill-climbing learner can get stuck at local optima. Unlike a depth-first search, it cannot explicitly backtrack through states, nor does it retain alternative structures such as beam or breadth-first search. However, hill-climbing approaches are simpler and have less memory requirements than other strategies.

3. Outline of the thesis

Before describing CLASSIT and its approach to concept formation in more detail, I first present a survey of other clustering methods in Chapter 2. There, I use a general definition of clustering, and consider a variety of alternative approaches to this problem. Because of a close relationship to my research, I focus on work in cluster analysis. The principal goal of Chapter 2 is to outline a variety of methods and organize them in a single framework. In this organization, a clustering method includes an *evaluation function* and an *algorithm*.

Chapter 3 contains a detailed description of the CLASSIT system. In addition to carefully presenting the underlying algorithm and its evaluation function, this chapter includes a detailed example to demonstrate the system. The CLASSIT system is carefully evaluated in Chapters 4 and 5. These chapters define performance measures and experimental methodology and then describe results as the system is tested over a set of both real and artificial domains. Chapter 4 focuses on the abilities of CLASSIT over different domains, while Chapter 5 presents comparative studies in which the learning system is modified.

Chapter 6 introduces a significant addition to CLASSIT: an attention mechanism. This mechanism is inspired by the human ability to focus attention on only a fraction of the perceptual input. For concept formation, attention can be used to make classification decisions without inspecting every attribute value in the instance description. This implies that the system must be able to find the more important features in the input, and to ignore other attributes. This ability is related to clustering in the face of missing information in the domain. Information can be missing either because the domain is incomplete in nature, or because the attention mechanism chooses to ignore some attributes. Chapter 6 includes an experimental evaluation of CLASSIT with missing information in both these cases.

Finally, Chapter 7 discusses some of the long-range goals for this concept formation system. Here, I introduce ICARUS, a cognitive architecture for learning (Langley, Thompson, Iba, Gennari, & Allen, in press). This framework includes a number of ideas for extending CLASSIT, especially as might be used by an autonomous robot in a reactive environment. Although the majority of these ideas are left as future work, some of the extensions inspired by ICARUS are implemented and undergoing testing.

4. Contributions of the thesis

The primary contribution of this thesis rests with the experimental evidence in Chapters 4, 5 and 6. I believe these chapters present a unified, robust approach to concept formation. The CLASSIT system includes a number of novel abilities: it works with a wide range of domains, and is able to learn despite noise or missing information in the input. To support this claim, I demonstrate the system's ability over a variety of real and artificial domains. Additionally, the attention mechanism described in Chapter 6 has the potential to streamline concept formation to focus on only a few important features in the input.

As a secondary accomplishment, I hope that this thesis brings together work from several very different paradigms. I have tried to extract the most useful abilities of cluster analysis, and apply them to problems faced by cognitive psychology, while using the tools of machine learning. The attention mechanism is one example of this theme, and I hope that work with the ICARUS framework will lead to others. This thesis emphasizes the importance of understanding and comparing research across different paradigms.

Finally, I hope that my experimental methodology does a comprehensive job of clarifying and evaluating my hypotheses. I believe that this type of experimental methodology was lacking in most early machine learning research. Without careful experimentation, it is far too easy for the researcher to make unwarranted claims, and to extrapolate to optimistic solutions. This thesis endeavors to make very precise claims supported by clear experimental evidence. To the extent that it succeeds, I hope this work stands as encouragement to other researchers in machine learning.

Chapter 2
Concept Formation and Cluster Analysis

In this chapter, I consider a very general form of the concept learning task: given a set of instances, group those instances into a set of classes. A fundamental difficulty for this task is that it requires some means of evaluating a set of potential classes. In particular, one needs an *evaluation function* that measures the quality of a set of classes with respect to the data. Creating an evaluation function is closely related to defining some measure of *similarity* among instances. In turn, the attributes (and attribute types) that describe instances affect the similarity measure. In addition, there are a number of different *algorithms* that create classes from instances. Although some algorithms require particular evaluation functions, often the researcher can try a set of different functions with a single algorithm, and vice versa.

In this chapter, I organize a large number of clustering techniques under one framework as a space of possible methods. This framework allows for a principled comparison of existing methods, and suggests places in the space for new, untried clustering methods. My hope is that this organization offers more insight than simply listing different methods from different fields, or trying to define a 'best' or 'optimal' technique. Although the framework I develop in this survey does not cover every clustering method, it does describe a wide variety of possible methods, and I believe it raises some interesting ideas about clustering.

To some degree, researchers in artificial intelligence have been unaware of related work outside of their own field. There is a large body of research in statistics and biology, usually known as *cluster analysis*, that is applicable to work in machine learning, if one allows for the different biases of these disciplines. Although a few AI researchers have acknowledged this area of work (Michalski & Stepp, 1983a; Stepp, 1987; Fisher & Langley, 1986), there has been no comprehensive survey of cluster analysis from an AI perspective, especially as it might be applied to concept formation. One goal of this chapter is to emphasize the similarity between these fields and to show how researchers in machine learning can benefit from a knowledge of cluster analysis.

I begin by presenting overviews of the clustering problem as seen from several different perspectives, including the machine learning view. In the second section, I describe the difficulty and importance of choosing a similarity measure or an evaluation function; this section also includes some of the most common and useful measures. I follow this with a description of algorithms that use these measures, and I conclude with some discussion of the difficulty of validating or evaluating a clustering technique.

1. Paradigms and biases

Rather than using a precise definition of the clustering task, I instead describe the problem from four different paradigms: machine learning, biology, statistics, and decision theory. This will make explicit the goals and biases of researchers from different fields.

My own biases for clustering come from the machine learning paradigm and the terminology used in this survey is from that body of literature. Both to give an idea of the diversity of the terminology, and for readers from other paradigms, the following is a brief 'translation' list of terms (the terms used in this thesis are presented last and italicized):

- an object, organizational taxonomic unit (OTU), event, or case is an *instance*,
- the characters, features, or variables that describe an instance are *attributes*,
- the distance metrics, association measures, or similarity coefficients that compare instances are *similarity measures*, and
- the closely related optimization criterion is an *evaluation function*.

Because researchers have approached the clustering problem from such different directions, not only is the problem described with different terminology, but the task itself varies slightly. This is hardly surprising – biology has very different goals than machine learning.

1.1 Clustering in machine learning

From the perspective of machine learning, clustering is viewed as a problem of concept formation. As with most of artificial intelligence, this field is biased by a computational analogy to human processes. Therefore, the process of clustering and the concepts produced by that process have implications for (human) learning, and for (human) knowledge organization and representation. These biases are from the closely related field of cognitive psychology, where the explicit goal is to study human cognitive processes. Concept formation has been studied by cognitive psychologists such as Smith and Medin (1981), Mervis and Rosch (1981), Barsalou (1987), Corter, Gluck and Bower (1988), and Anderson (1988). Although machine learning does not always make explicit this bias, there is usually at least a weak analogy between the learning system and the human learner.

As an example of an application for concept formation, consider an exploring robot that perceives a sequence of different balls. Even if the robot is equipped with a perceptual system that reduces each instance to a set of attribute-value pairs, it still must create and organize a useful set of concepts about these balls. For example, after observing a few baseballs, it should create a concept for these and be able to recognize a new baseball as a member of that class, and not as an instance of some other class (volleyballs, tennis balls, etc.). Using this example, I can describe the distinguishing features of clustering in machine learning. Since the work in this thesis is from the machine learning paradigm, this list of features reinforces the discussion in Chapter 1 of my own biases and goals.

One distinguishing feature of concept formation is that the classes learned should be *intensional*, rather than extensional. For example, the baseball class should include a "conceptual description" of the baseballs seen, rather than simply a list of all member instances (Michalski & Stepp, 1983b). This emphasis on intensional concept definitions means that evaluation functions that compare classes are more appropriate for concept formation than similarity measures that compare instances.

A second aspect of concept formation is that the classes learned are usually arranged in a concept hierarchy. That is, the learned concepts are organized into a hierarchy with more general, inclusive concepts toward the top, and more specific, exclusive concepts toward the bottom. This reflects the hierarchical nature of knowledge in typical machine learning domains. For example, soccer balls and volleyballs are more similar to each other than to baseballs or to lacrosse balls. A natural hierarchy for these four types of balls would be to put soccer balls and volleyballs together into a more general "soft, large" class, and lacrosse balls and baseballs into a "hard, small" class.

A third characterizing feature of concept formation is that learning occurs *incrementally*. As the robot observes each successive ball, it should add to its knowledge immediately; the concepts learned are updated by each new experience without reprocessing previous instances. In contrast, a nonincremental system must receive the entire set of instances before producing a set of classes. Such a system is incompatible with the goals of concept formation because one may not know the complete 'set' of instances, and one may need to use the learned concepts at any point in time. For example, the robot should be able to use its knowledge at any point during learning, and it should continue learning, no matter how many balls it encounters. These problems are perhaps more obvious for human learners, who observe a never-ending sequence of instances.

Finally, one should be able to measure the improvement of a learning system on some *performance task*. This is a task used to test (and quantify) the ability of the system before and after learning. With this type of numeric measure, the success of a concept formation system can be evaluated over a number of different domains, or a set of different systems can be compared on given data set.

As I have described the problem, clustering is *unsupervised*. There is also a large amount of work in machine learning on supervised concept formation, usually known as "learning from examples". Although this is a related task, the differences between these two problems are very important. Supervised concept formation learns to determine which of a known set of classes an instance belongs to, whereas unsupervised learning imposes a structure of concepts (ones that are not known a *priori*) on the set of instances.

1.2 Clustering in biology

Historically, the first computational approaches to clustering arose in the field of biology. In particular, 'numerical taxonomy' grew out of the following problem: given a set of species or organisms, find a taxonomic hierarchy that organizes them into species, genera, phyla, and classes. The principal purpose of this hierarchy is to suggest evolutionary relationships among individuals.

As an example, suppose a biologist discovers a new set of worms. After describing each worm by some set of attributes (or "characters"), the biologist inputs this set of instances to a clustering system. The resulting hierarchy should define classes of similar worms, as well as showing how these classes are related to each other. This information can lead to predictions about how related worms may behave, and also to theories about how worm attributes have evolved over time.

The emphasis in this discipline is to find a practical method, instead of worrying about the theoretical implications of a particular technique. For instance, biologists are not very concerned with evaluating the performance of a given technique. Instead, their measure of success is subjective: if a method produces a useful taxonomy (one that leads to new insights, or has interesting evolutionary implications), then it is a good one. This bias has led other researchers to state that "the [biologist's] view of clustering is considered a radically empirical approach" (Aldenderfer & Blashfield, 1984, p. 21).

Since these methods are explicitly looking for a taxonomy, the classes found should be disjoint and organized into a hierarchy. A flat list of classes or a set of overlapping classes is not as useful. However, in contrast to machine learning methods, the classes created by numerical taxonomy techniques are usually extensional, and similarity measures, rather than evaluation functions, are used as the basis for clustering.

Although a robust, general-purpose algorithm is appreciated in any field, these are not crucial features for numerical taxonomy. In this paradigm, it is reasonable to use different algorithms for different data sets. Also, for any given clustering problem, the researcher is only interested in a finite (usually small) set of instances. Therefore, the ability of a clustering method to incrementally process new instances is not very important.

This chapter emphasizes biological clustering methods; however, many of these same biases can be seen when clustering techniques are used in other sciences. In particular, the same emphasis on practical methods, similarity measures, and subjective evaluation appears in clustering for ecology and psychology.

1.3 Clustering in statistics

The statistician has a much more formal view of the clustering problem. In this approach, researchers are interested in a careful definition of clustering and in exploring theoretical implications of clustering methods. Although this paradigm has had some success, the heuristic nature of clustering can be an obstacle to the type of rigorous analysis preferred by statisticians. Likewise, objectively evaluating the result of a clustering technique has proven difficult.

For the statistician, one place to begin is a comparison of cluster analysis with other, well-established statistical methods such as factor analysis, analysis of variance, and discriminant analysis. For example, statisticians point out that choosing a set of attributes that describe instances is the general problem addressed by factor analysis. However, it appears that performing factor analysis as a pre-processing step has a detrimental effect on clustering.[1] Similarly, the standard multivariant practice of normalizing variables can cause problems: normalization can obscure differences that may be crucial for clustering (Everitt, 1980).

Statisticians have also analyzed and compared the algorithms and evaluation functions of clustering methods themselves. Although this effort has shown that some methods and similarity measures are equivalent (Anderberg, 1973), it has not been able to establish any single clustering method as best. The difficulty is that, unlike most statistical methods, clustering is *heuristic*. Since the algorithms use 'rules of thumb' that are not guaranteed to produce correct solutions, they are difficult to analyze and compare.

Although one cannot measure the subjective goal of finding an 'interesting' set of classes, statisticians are interested in quantitatively evaluating aspects of a solution. Unlike the biologist's

1. There is considerable debate on this issue. See Everitt (1979) or Aldenderfer and Blashfield (1984) for more discussion.

perspective, the statistician's 'solution' need not be a hierarchy of classes: for some domains, a flat list of classes is more appropriate; for others, overlapping or probabilistic classes may be preferable.

1.4 Clustering as decision theory

An abstraction of the clustering problem has been studied by a few researchers in the field of decision theory (Jaynes, 1986; Cheeseman, Kelly, Self, Stutz, Taylor, & Freeman, 1988). In this view, the goal is to correctly predict the probabilities that a new instance x is a member of a class ω_i: $P(\omega_i|x)$. This expression can be re-written using Bayes' theorem:

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)} \quad .$$

The probability of each class, $P(\omega_i)$, is usually known – it can be computed as the number of members of ω_i divided by the total number of instances. Additionally, since $P(x)$ is the probability of x independent of class, it can be ignored – when comparing two different classes, ω_1 and ω_2, the denominator is the same and can be dropped. In fact, the only unknown term on the righthand side is the class conditional probability density function, $P(x|\omega_i)$. This is the probability of x given ω_i, or the value of x that would be predicted by class ω_i. These functions must be estimated; for example, one can assume a normal distribution, and search for a good estimate of the parameters μ and σ that characterize this distribution.

Using this foundation for a clustering method guarantees that the method will maximize the probability of correctly characterizing the data. Hence, if a system is faithful to the theory, it is "provably" optimal, and there is little need for empirical evaluation. For this reason, researchers from this paradigm place less emphasis on the algorithms used to implement a Bayesian classifier. However, it is usually difficult to create a system that is faithful to the theory, and a number of assumptions must be made along the way. Most solutions based on decision theory encounter the usual problems of heuristic search, but at this lower, algorithmic level. The principal advantage of the Bayesian framework is that by providing an underlying theory of decisions, the researcher can easily make explicit the theoretical implications and necessary assumptions for any solution to the clustering task.

2. Similarity measures and evaluation functions

Although it is not always made explicit, any clustering technique produces a set of classes in which the members of a given class are similar to each other in some way. Using the 'search' terminology, one is looking for classes in which the similarity between instances within a given class is greater than that between instances from different classes. From this perspective, a clustering technique can be characterized by how it defines 'similarity'.

For some techniques this basis is made explicit: there is a *similarity measure* or a *distance metric* that quantitatively measures the distance between two instances. For others, the goal is to maximize some *evaluation function* or criterion. Such a function measures the 'goodness' of a set of classes; usually this is based on the similarity among classes, rather than between two instances.

Clustering depends upon a similarity measure; in turn, the scores of a similarity measure depend upon the attributes used to describe an instance. I begin by describing some issues and problems pertaining to attributes. Next, I present a set of similarity measures and evaluation functions. These are described independently from the clustering algorithms that use them, so that they can be compared directly. This also makes clear that there is usually more than one choice of measure or evaluation function available to the researcher.

2.1 Attributes: choices and representations

For the biologist or social scientist who is approaching clustering techniques as a tool, there are a potentially infinite number of attributes available to describe an instance. Rather than blindly using as much information as can be found, the scientist can choose only those attributes that are 'relevant' to the task. As mentioned earlier, factor analysis is a well-defined statistical method that is sometimes used to create a smaller number of more 'appropriate' attributes from the pool of available attributes.

Unfortunately, this is somewhat circular logic. In spite of its widespread use, Everitt (1979) argues against using factor analysis or any method that eliminates attributes before clustering. The purpose of clustering is to discover an unknown set of classes. As it searches for these, it will establish which attributes are 'relevant', but to decide this beforehand would slant the clustering process. Factor analysis can have the detrimental effect of hiding those attributes that may be crucial to finding a hierarchy of classes. This method assumes a single, known class; hence, Everitt suggests that it may be used after clustering, but never beforehand. Researchers also sometimes place weights on the attributes prior to clustering. This has the same effect as using factor analysis, and is vulnerable to the same criticism.

Any method for measuring similarity depends to some degree on the representation used for the attributes that describe an instance. Anderberg (1973) points out that there are two ways of characterizing attributes: the measurement scale used for the attribute, and the number of possible values an attribute may take on. For this thesis, I will describe four main types of attributes: continuous, ordinal, symbolic, and binary.[2]

Continuous attributes have an infinite range and are measured along a continuous scale. Examples of this type of attribute are real-valued measurements of height, weight, and temperature. An *ordinal* attribute has a finite range with an ordering on the possible attribute values. Examples might be number-of-fins, or any continuous attribute that has been rounded, such as age to the nearest year.[3] A *symbolic* attribute also has a finite range, but there is no order to its values. Examples of this may be shape, place of birth, or type of sailboat. Finally, a *binary* attribute

2. Note that my terminology is different from Anderberg's, reflecting my machine learning bias.
3. These examples are actually both *interval* attributes, where the difference $x - y$ between two attribute values is well defined. Ordinal attributes can also have values such as **many** and **few**; although many is greater than few, one cannot define the distance between them.

has only two possible (symbolic) values. Often, these are presence-or-absence attributes such as has-backbone or is-freckled.

The similarity measure or evaluation function employed will be dependent upon the attribute types used to describe instances. In fact, instances may be described by a combination of different types of attributes. Unfortunately, one of the many unsolved problems in cluster analysis (from a statistician's point of view) is that there is no 'good' way to combine different attribute types. That is, despite some attempts, there are no theoretically sound similarity measures that can be applied to different attribute types, especially if binary and continuous attributes are combined. For this reason, the measures described below are organized according to whether they are appropriate for continuous, ordinal, symbolic, or binary attributes.

2.2 Measures for continuous or ordinal attributes

I begin by considering similarity measures for continuous attributes: measures comparing two instances that are described by a set of continuous or ordinal attributes. Let i and j be the instances, each described by K attributes, e.g., $i = \{x_1, x_2, .., x_K\}$. One of the most obvious similarity measures between these two points is to use a *distance metric.*[4] Each attribute defines a dimension, and each instance can be plotted in this K-dimensional space. The Euclidean distance between two points is

$$D_{ij} \quad = \quad \left[\sum_{k=1}^{K}(x_{ik} - x_{jk})^2\right]^{1/2} \quad .$$

A simpler, and related metric, is the city-block distance:

$$D_{ij} \quad = \quad \sum_{k=1}^{K} |x_{ik} - x_{jk}| \quad .$$

Euclidean distance is probably the most well-used measure for similarity; it is also used as the basis for some evaluation functions. Although a city-block distance may seem less intuitive, it is computationally much cheaper, and may be appropriate when ordinal attributes are used.

Both of these measures are sensitive to linear transformations of the input data. For example, if some attributes are transformed from, say, inches to miles, then these re-scaled data will have completely different similarity measures.[5] As Duda and Hart (1973) point out, this may or may not be a problem, depending on whether such transformations are natural to one's domain.

The use of Pearson's correlation measure is one way to achieve invariance with respect to linear transformations. The original intent of this measure is to correlate pairs of attributes for factor analysis. In order to correlate instances, researchers simply reversed the equation syntactically.

4. Distance metrics are more properly called *dissimilarity measures*, since the greater the distance, the less similar two instances are.
5. Note that this 'attribute scaling' is related to the attribute weighting described earlier. Attribute weighting is a controversial practice only if the similarity measure used is sensitive to linear transformations of the data.

That is, imagine the data set as a $K \times N$ matrix of values (if there are K attributes and N instances); the usual use of correlation is to measure similarity among the columns (the attributes), so the reversed equation should measure similarity among the rows (the instances). This reversed correlation is referred to as *Q-mode factor analysis* (Anderberg, 1973, p. 113). It states that the correlation (distance) between two instances is

$$D_{ij} = \frac{\sum_k^K (x_{ik} - \bar{\bar{x}}_i)(x_{jk} - \bar{\bar{x}}_j)}{\left[\sum_k^K (x_{ik} - \bar{\bar{x}}_i)^2 \sum_k^K (x_{jk} - \bar{\bar{x}}_j)^2\right]^{1/2}} ,$$

where $\bar{\bar{x}}_i = 1/K \sum_k^K x_{ik}$, the average attribute value for a given instance, i.

This approach has been used with some success among researchers in psychology (see Aldenderfer & Blashfield, 1973, pp. 22–23). However, this measure is largely discredited (especially in other fields) because there is no justification for the syntactic reversal. The meaning of the equation is lost: for example, since $\bar{\bar{x}}$ is an average across different attributes, it may be averaging 'apples and oranges', and may not have the semantics expected for that term.

2.3 Measures for binary or symbolic attributes

Neither correlation nor Euclidean distance can be applied to an attribute with binary or symbolic values. One characteristic of such an attribute is that, given two values, the expression $x_i - x_j$ does not have any meaning. A similarity measure for symbolic attributes is faced with a simple comparison: either two values are the same, or they are different. Over a set of attributes, the simplest way of comparing two instances is to find the percentage of matching attributes:

$$S_{ij} = \frac{\textit{number of matching attributes}}{\textit{total number of attributes}} .$$

In artificial intelligence, this is a 'partial match': a score of one indicates that all attributes match, while a zero says that no attributes match.

Since binary attributes are very common, researchers have usually treated this case separately. If one looks at two instances, i and j, there are four possible relationships for each binary attribute; Table 2 presents these in a 2×2 association table. If these are totaled over all attributes, a and d represent the number of matched attributes, while c and b are mismatches. Therefore, the simple matching measure described above can be expressed as

$$S_{ij} = \frac{a+d}{K} ,$$

where $K = a + b + c + d$, or the number of attributes. A distance measure can be defined as $b + c$: the more mismatches, the greater the distance between the instances. This is known as the *Hamming distance* (Hamming, 1980). The distinction between a, the number of positive matches, and d, the number of negative matches, is made because binary attributes can express the presence

Table 1. A 2 × 2 association table

$j \backslash i$	1	0
1	a	b
0	c	d

or absence of some observable feature. This is often the case in biology; in such a domain, it may be more appropriate to use a measure that does not count 'missing' matches:

$$S_{ij} = \frac{a}{a+b+c} \quad .$$

This similarity measure is known as *Jaccard's matching coefficient* (Romesburg, 1984, p. 143).

Jaccard's coefficient and the simple matching measure are the most commonly used similarity measures for binary attributes. However, there are a large number of related similarity measures that can be defined in terms of Table 1. Most of these other measures are closely related to Jaccard's coefficient or the simple matching measure; see Romesburg (1984) for a description of 12 different measures for binary attributes. As one example, similarity can be described as the probability that instances i and j match on a variable minus the probability that they mismatch:

$$S_{ij} = \frac{(a+d)-(c+b)}{K} \quad .$$

Although these measures are defined expressly for binary attributes, they can be easily adapted for symbolic attributes. This can be accomplished either by converting a symbolic attribute into a set of binary attributes, or by converting the measure itself. Converting the attribute can be accomplished by using the n possible values to create n presence-or-absence binary attributes. (Of course, the use of this type of attribute suggests Jaccard's coefficient as an appropriate measure.) Alternatively, a measure defined for binary attributes can be used with symbolic attributes, if one lets $a + d$ be the number of matches, $b + c$ be the number of mismatches, and d be the number of times the corresponding attribute is missing, or not applicable to the given pair of instances.[6]

Although it is not difficult to convert from symbolic to binary attributes, the only measure that can compare instances with both symbolic and continuous attributes is a simple combination of existing measures. *Gower's coefficient* is a sum over all attributes of one of three measures: if the attribute is binary, Jaccard's coefficient is used; if the attribute is symbolic, the simple matching measure is used; and if the attribute is continuous (or ordinal) a normalized city-block distance metric is used. Despite its apparent generality, Romesburg (1984) notes that the measure has rarely

6. Anderberg (1973) presents this conversion, as well as many other ways to convert measures and attributes from one type to another.

been used in practice. This may be because the mathematical properties of Gower's coefficient are unknown; of course, this criticism can be made against other, well-used measures.

2.4 Evaluation functions

Evaluation functions are distinct from similarity measures in that they compare sets of classes, rather than a pair of instances. This difference can be trivial; some evaluation functions are simple extensions of similarity measures. However, the emphasis on classes rather than instances is important to machine learning. From this perspective, as the system learns, the evaluation function controls the search for useful classes by evaluating the quality of a set of concepts with respect to the data. In contrast to similarity measures, evaluation functions often imply a particular type of concept definition. As I present different evaluation functions, I will point out their relations to various similarity measures, as well as their implications for concept representations.

Average distance

The most straightforward way to evaluate classes is to evaluate all of the members by using a similarity function. For example, one common method is to sum the distance from each object to the class mean; the lower this average distance, the better the class. Although this could be defined with any metric, the most common function is based on Euclidean distance; over all classes, the evaluation function is:

$$trace(\mathbf{W}) \quad = \quad \sum_j^J \sum_k^K \frac{1}{N_j} \sum_i^{N_j} (x_i - \bar{x}_{jk})^2 \quad ,$$

where N_j is the number of members in class j, and $\bar{x}_{jk}$ is the average over all members of class j for attribute k. This expression is known as $trace(\mathbf{W})$, because it is the sum along the diagonal of the within-group covariance matrix. (Also known as a *scatter matrix*, this is the matrix of all possible covariances among K attributes.) Note that for a single class and attribute, this expression corresponds to the variance for that attribute. In fact, this function suggests a concept representation consisting of a list of means and variances for each attribute, since this information is needed to compute $trace(\mathbf{W})$.

This evaluation function is one of a set based on the matrix identity, $\mathbf{T} = \mathbf{W}+\mathbf{B}$, where $\mathbf{T}$, $\mathbf{B}$ and $\mathbf{W}$ are the total, between-group, and within-group scatter matrices, respectively.[7] In general, these functions attempt to either minimize $\mathbf{W}$ (a measure of within-group differences) or maximize $\mathbf{B}$ (between-group differences). In order to compare matrixes, they must be converted to a scalar: one can use either the determinant of the matrix, or (more cheaply) the 'trace' of a matrix. The three most common functions are minimizing $trace(\mathbf{W})$ (defined above), maximizing $trace(\mathbf{W}^{-1}\mathbf{B})$, and minimizing the determinant of $\mathbf{W}$.

It is important to note that $trace(\mathbf{W})$ has the same problem as Euclidean distance; it is sensitive

7. See Hand (1981) for a more detailed discussion of this identity, as well as further references to the use of these functions.

to normalization of the data, and to linear transformations of attributes. It also prefers clusters that form hyper-spheres in the attribute space. However, the other proposed functions are computationally much more expensive, especially since they need to compute the inverse of the matrix **W**. Anderberg (1973, p. 175) cites evidence that there are no good reasons for selecting one function over another, and therefore he suggests considering only $trace(\mathbf{W})$, the simplest function.

These average distances functions are defined to work with instances described by continuous attributes. Although one can define a 'distance' between two instances with symbolic attributes, there certainly cannot be a mean for each symbolic attribute. If the data includes symbolic attributes, the researcher must use some other evaluation function.

Attribute correlation

Hanson and Bauer (1989) describe an evaluation function for symbolic attributes based on correlation between attributes. Each class is defined by all possible pairwise contingency tables (as in Table 1). To evaluate the class, the distribution over each contingency table, D_{ij}, is averaged for all pairs of attributes; the greater this average correlation, the better the class. The 'cooccurance distribution' (to use the authors' term) is defined as:

$$D_{ij} = \frac{\sum_m \sum_n (x_{mn} log\ x_{mn})}{(\sum_m \sum_n x_{mn})(log\ \sum_m \sum_n x_{mn})} \quad ,$$

where x_{mn} is a count in the contingency table for attributes i and j. Recall that each position in a contingency table counts the number of times attribute value m for attribute i co-occurs with value n for attribute j. The average value of D_{ij} is a measure of the within-class cohesion. The authors label this as W_c, and state that the complete evaluation function is to maximize W_c/O_c, where O_c is defined in a similar way, except that the cohesion is measured across classes. Once again, the general goal is to maximize within-group similarities and between-group differences.

This 'correlation' measure, D_{ij}, is more closely related to the chi-square test of independence than to Pearson's correlation coefficient for continuous attributes. For this reason at least, their method is distinct from Q-mode factor analysis. However, Anderberg (1973) shows that there are dangers in interpreting results with any measure based on the correlation of attributes. In particular, the distribution measure may not be consistent across different attributes; if $D_{ij} = .52$, and $D_{kl} = .41$, one *cannot* conclude that i and j are more closely related than k and l.

Another problem with this particular method is that it is only well suited to domains with relatively few binary attributes. Although there are no theoretical problems with other types of data, the storage costs and, to a lesser degree, the computational costs become very severe. With K attributes each with M values, each concept needs $(K^2 - K)/2$ tables, each of which has M^2 entries.

The advantage of a method based on correlation is that it can easily find concepts that depend on some relationship between two attributes.[8] Because this relationship is explicitly represented in

8. Of course, as used here, 'correlation' only refers to pairs of attributes. Three-way or N-way correlations are not explored.

the correlation tables, the concept can be easily discovered and represented by Hanson and Bauer's system. Systems that do not use some form of correlation often have difficulty with this type of class.

FUNCTIONS BASED ON INFORMATION THEORY

Gluck and Corter (1985) present an 'information theoretic' evaluation function, *category utility*, for symbolic attributes. This function is based on the probability of an attribute value, $P(x_{kv})$. This probability can be expressed as the number of times attribute k has had value v, divided by the total number of instances. (Note that this probability is closely related to the simple matching measure.) Category utility measures the information that is gained by partitioning instances into classes. For a given attribute k,

$$Category\ Utility(k) \quad = \quad \frac{\sum_{j=1}^{J}\left[P(C_j)\sum_{v=1}^{V}P(x_{kv}|C_j)^2\right] \ - \ \sum_{v=1}^{V}P(x_{kv})^2}{J} \quad ,$$

where V is the number of attribute values for attribute k and J is the number of classes. $P(x_{kv}|C_j)$ is the probability of the attribute value conditioned by the class C_j, meaning that only those instances in class C_j are considered. In contrast, $P(x_{kv})$ is that probability without any class information; it is the information at the parent class.[9] Although category utility is based on the simple matching measure, the subtraction of the final term allows the function to measure information gain from parent to children. This gain is then divided by the number of children, so that different size partitions can be compared.

Both category utility and Hanson and Bauer's evaluation function work only for symbolic attributes; because they iterate through all possible attribute values, they cannot be applied to continuous attributes. In the next chapter, I will describe a synthesis of this evaluation function with the *trace*(**W**) function for use with continuous attributes.

BAYESIAN CLASS EVALUATION

Although a Bayesian clustering system explicitly compares classes, it does not usually have the same type of evaluation function as those described so far. Instead of evaluating classes with respect to all instances, the basic decision theory equation (presented in Section 2.4) compares a single instance against a set of classes. The difficulty with this equation is that in order to compute the class conditional probabilities, $P(x|\omega_i)$, one needs an estimation of the class parameters that define each ω_i. Duda and Hart (1973) conclude that, in general, there is no analytically simple way to find this estimation, and that computational costs for an exact solution rise exponentially with the number of instances.

However, there are a number of estimation techniques that have been empirically successful. Fried and Holyoke (1984) use a simpler algorithm based on the Euclidean distance similarity measure

9. Gluck and Corter (1985) defined category utility for two classes; here, I have shown Fisher's (1987a) generalization to J classes. The information theoretic model also uses logs instead of squared terms ($P(x)log(P(x))$ instead of $P(x)^2$). However, Gluck and Corter claim that this difference will not affect the behavior of a clustering system.

to find initial class parameter estimates. Using these estimates, they can then determine $P(x|\omega_i)$. Anderson (in press) uses a *coupling probability* (a user-defined parameter) to define the prior probability of each class and the probability that an instance is a member of a new class. This allows him to get an initial partitioning of instances into classes. Then, for a new instance, he can compute $P(x|\omega_i)$ based on the membership of each ω_i.[10]

Unfortunately, it is difficult to compare these systems' "evaluation functions" to others. Part of the difficulty is that the task is described as estimating (or updating) class parameters, rather than evaluating the quality of class definitions. For example, Cheeseman et al. (1988) describe the update algorithm for their Bayesian system, but do not give an equation for this evaluation process, nor describe how it compares to other clustering evaluation functions.

3. Algorithms for clustering

Any clustering technique can be described as an algorithm that uses some kind of a similarity measure or evaluation function to search for a set of classes. Thus far, I have described these measures out of context; this section will show how they are used by algorithms. My goal in presenting these components separately is to show that any combination of algorithm and measure defines a clustering technique. To some degree, one may choose a part from bin 'a' and a part from bin 'b' to create a clustering method. Of course, very few researchers have actually tried this, and not every nut will fit onto every bolt. For each algorithm, I will point out the original similarity measure proposed, as well as others that could be used.

In this section, I divide clustering methods into three groups: *agglomerative* algorithms, *iterative optimization* algorithms and *incremental* algorithms. The agglomerative approach is the oldest, having been proposed by workers in biology and ecology. With the advent of the computer, iterative optimization methods became popular as a simpler heuristic approach to clustering. Finally, incremental algorithms were inspired by human concept formation, and were created by researchers in machine learning.

In addition to describing some of the most important algorithms, I will consider two characteristics of each method. First, although any algorithm must produce some set of classes as output, the form and organization of the classes is dictated by the researcher's goals. For example, the researcher may prefer a simple list of classes or he may need a specific-to-general hierarchy of classes. Likewise, the researcher may prefer each instance to be assigned to a single class, or to more than one class, or even to all classes probabilistically. Second, different algorithms have very different computational and memory costs. The computer cannot be treated as an infinitely powerful machine. Especially from a machine learning point of view, it is important that the algorithm and the similarity measure be as inexpensive as possible.

10. Because Anderson works with symbolic attributes, his evaluation function is related to the simple matching measure, except that the new instance is compared to the set of all member instances. Anderson's work is also interesting because his algorithm is *incremental* (see Section 4.3).

3.1 Agglomerative methods

Historically, the first algorithms for clustering were agglomerative methods. Since they were developed by biologists, they produced a hierarchy (a taxonomy) of classes, from the most general class (including all instances) to the most specific classes (covering only one instance). Although these are still the most widely used algorithms, they are expensive both in space and time requirements.

An agglomerative method begins with each instance as a separate class, and repeatedly combines these smaller, specific classes to form larger, more general classes. This process builds up a hierarchy of classes, finishing when all instances have been agglomerated into one top-level class.[11] In order to determine which instances to 'agglomerate', these algorithms require a *similarity matrix* that shows how close, according to some similarity measure, every instances is to every other instance. Given this matrix, a general agglomerative algorithm can be described as follows:

```
1.  Compute and store the similarity matrix.
2.  Find the best value in the matrix and its associated pair of instances.
3.  Merge these two instances (or classes) into a larger class.
4.  Create a new similarity matrix that includes the new class.  (This may
    require recomputing some values.)
5.  If there is only one class, return the hierarchy produced and stop; else,
    go to step 2.
```

This description does not make clear exactly how to accomplish the fourth step. In order to enter a new class into the similarity matrix, one must decide how to use the similarity measure to compare a class to an instance or to other classes. The simplest (or at least the cheapest) solution is to define the similarity between two classes, A and B, as the similarity of the closest two instances a and b, where $a \in A$ and $b \in B$. This is called the "nearest neighbor" or "single linkage" algorithm, because two classes are combined by the shortest single link between them. This algorithm tends to produce long chaining clusters and this can be a disadvantage in some domains.

Other agglomerative algorithms use different ways to measure the 'similarity' between two classes. For example, instead of a single link, one can find the "average link" distance between classes. This requires computing the class mean for each attribute every time a class is created or expanded, and then measuring the distance to other groups or instances from this mean. This method avoids creating long chains, and instead prefers clusters that form hyper-spheres in the instance space.

Unfortunately, this method is rather expensive: because it requires recomputing part of the similarity matrix, it has a computational cost of $O(n^3)$, where n is the number of instances. The nearest neighbor algorithm can be implemented so that no recalculation occurs, but the computational cost is still $O(n^2\ log(n^2))$,[12] in addition to the cost of sorting the similarity matrix. Depending on the similarity measure and the number of attributes per instance, computing and storing the similarity

11. The reverse of this approach is known as a *divisive* algorithm. This begins by assuming every instance is in the same highest-level class, then repeatedly divides this class into some number of children, until each (very specific) class has only one instance. Although a few such algorithms have been proposed (MacNaughton-Smith et al., 1964; Fisher, 1984), they have been rarely used.

12. For each of n^2 similarity values, one must check to see if the instances have already been merged into the same class (a cost of $log(n^2)$). See Aldenderfer (1973) for a description of this algorithm.

matrix alone may be prohibitively expensive. In fact, for domains where there are a large number of attributes and instances, agglomerative methods are largely unsuitable.

Another significant problem with agglomerative algorithms is that they produce only *binary* hierarchies. Instead of this structure, the researcher is often interested in finding some 'optimal' number of classes. Although there are some methods for cutting or flattening the binary hierarchy (see Aldenderfer & Blashfield, 1984), these are not guaranteed to find the best or most appropriate set of classes. In fact, Everitt (1979) points out that such techniques may be misleading; he concludes that finding the optimal number of classes from a binary hierarchy is an open problem.

3.2 Iterative Optimization

Iterative optimization algorithms were created in response to the expense of agglomerative methods and to the difficulty of finding the correct number of classes. Instead of trying to find this ideal number, one can simply give the number of classes, k, to the clustering system. Iterative optimization searches for these k classes by repeatedly reassigning instances to different classes in order to improve the score of some evaluation function. Although these algorithms do not produce hierarchies of classes, they are more efficient than agglomerative methods, and by transferring instances from class to class they can recover from an initial 'bad' decision.

In general, one can view the clustering problem as a search over the huge space of possible partitionings of the instances into classes. A simple method would examine every possible partition, and find the one with the best score according to an evaluation function. Unfortunately, this is computationally impossible even with a relatively small number of instances; for example, there are approximately 5.28×10^{28} ways to partition 50 instances into four classes.[13] Therefore, instead of a complete search through this space, iterative optimization methods use hill-climbing techniques to iteratively improve the evaluation score until an optimum is reached. As with any hill-climbing method, the starting point for the search may be critical, and the algorithm can converge on a local optimum instead of the global optimum.

Iterative optimization methods require a criterion to optimize at each iteration. Theoretically, one could choose any evaluation function for this role. However, in order to keep the overall clustering system efficient, the researcher should use a relatively simple evaluation function, one that that system can compute cheaply as it considers each re-assignment. For example, one of the simplest and most popular techniques is the *k-means algorithm*:

```
1.   Use the first k instances as seed points.
2.   Assign each of the remaining instances to the class represented by the
     nearest (Euclidean distance) seed point.
3.   Recompute new seed points as the centroids (the average attribute values)
     of each class.
4.   Iterate between steps 2 and 3 until no reassignments are made.
```

Although the number of iterations required before halting is unknown, Anderberg (1973) gives a

13. Duda and Hart (1973) give the exact expression for the number of ways to partition n instances into c classes; an approximation is $c^n/c!$.

proof that such algorithms will eventually converge, and in practice this is usually a reasonably small number (less than ten). When Euclidean distance is used, Hand (1981) shows that this algorithm is equivalent to optimizing the $trace(\mathbf{W})$ evaluation function.

One can make a number of modifications to this algorithm. First, since the starting point can be critical to a hill-climbing searcher, different methods can be used to choose it. For example, the k seed instances can be chosen randomly, or they can be chosen so that all seeds are at least some minimum distance apart. Duda and Hart (1973) point out that the entire algorithm can be repeated with different seed selections so that the researcher can compare possibilities. In fact, they even suggest using an agglomerative method to find the initial partition, although this seems expensive. Anderberg (1973) also describes a number of seed selection techniques.

A second modification can be made by computing new class centroids whenever an instance is reassigned to a class. In this case, the algorithm may converge much earlier; for example, MacQueen's (1967) k-means algorithm uses only two passes through the instances. In the first pass, centroids are modified as each reassignment occurs; during the second pass the centroids remain fixed.

Finally, one can expand the algorithm so that it addresses two higher-level problems. First, because one may not always be able to specify the number of classes, k, *a priori*, one can try the k-means algorithm with different values of k, allowing an estimate of the 'best' k. Second, the researcher may need a hierarchy of classes, rather than the flat list that iterative optimization algorithms usually produce. To this end, one can simply execute the algorithm recursively on each of the k groups identified on its first execution.

Although these extensions appear to be expensive, 'brute-force' solutions, Michalski and Stepp (1983b) have incorporated them into the k-means algorithm in their CLUSTER/2 program.[14] This system also includes a step that helps avoid local optima: when the k-means algorithm converges, CLUSTER/2 chooses new seed points at the edge of each class, instead of at the centroids. It then restarts the k-means algorithm with these new seeds, and if the resulting partition remains unchanged, the system returns those classes as the solution. Finally, CLUSTER/2 differs from most iterative optimization techniques in that it uses domains with symbolic, rather than continuous attributes. As the $trace(\mathbf{W})$ evaluation function only works with continuous domains, this system uses a special-purpose evaluation function defined by a set of (user-specified) parameters. Unfortunately, iterative optimization has not been tried with any more principled evaluation function for symbolic attributes.

Cheeseman et al. (1988) also use an iterative optimization approach, but their AUTOCLASS system does not strictly partition instances into classes. This system carries out a hill-climbing search for the best set of classes, where each class is defined by a mean and a standard deviation. However, class membership is completely probabilistic, specifying the probability that $x \in c$, for any object x and for every class c. These 'fuzzy' classes are especially suitable when an instance is described reasonably well by two (or more) competing classes.

Although iterative optimization algorithms have met with some clear successes, their domain

14. CLUSTER/2 is not usually identified as using a k-means iterative optimization clustering algorithm. This characterization became clear only after surveying these older methods.

of application is somewhat constrained. If an approximation of k is available, if the instances are described by continuous attributes, and if a flat list of classes is sufficient, then iterative optimization is a good approach to the clustering problem. Although the CLUSTER/2 system relaxes these constraints, it does so only at substantial computational cost.

3.3 Incremental methods

In contrast to other approaches, incremental algorithms view the set of instances as a potentially infinite sequence. As each instance is processed, the algorithm makes an incremental modification to its current set of concepts. At any point in time, the concepts reflect information gained from all the instances encountered up to that point. This approach was designed with human concept formation in mind; it seems unlikely that a human learner would need to first receive some number of instances, and then stop receiving and perform the computation of the clustering task.

Even without this bias, these incremental systems are useful for a number of more pragmatic reasons. They require less computational time than other algorithms, and can therefore process larger databases. These algorithms also offer a solution to the problem of determining the number of classes. Finally, this type of algorithm is almost essential for any application in which the classes change over time. Schlimmer and Granger (1986) refer to this as *concept drift*: if new instances reflect new or different concepts, an incremental algorithm can adjust its concept definitions accordingly.

A general incremental algorithm for adding each new instance x to a hierarchy of classes can be described as:

```
For a new instance I, and some concept hierarchy:
    1. Incorporate I into the root node.
    2. Either: a) incorporate I to an existing child concept, or
               b) create a new child concept based on I.
    3. Unless a new child is added, recurse on the selected child
       concept.
```

Usually these algorithms produce a hierarchy of classes, but step three can be omitted if one prefers a simple list of classes. Unlike agglomerative methods, incremental algorithms need not produce binary hierarchies: the branching factor is variable and determined by how often new classes are created (in step 2b). Determining when to make a new class is critical to these algorithms – this choice allows incremental algorithms to automatically find an appropriate number of classes from the data.

EPAM (Feigenbaum, 1963) was the first system in artificial intelligence to approach the clustering problem. This system applies *monothetic* decision making to the basic incremental algorithm. This means that all internal nodes in the hierarchy (or *discrimination network*) are associated with a single attribute. As the system sorts a new instance through the hierarchy, each internal node tests a single attribute, and clustering decisions are based on that attribute value. In contrast, leaf nodes (called *images*) include a set of attribute values. When the clustering process reaches a leaf node, one of two actions occur. If the new instance matches the values at the node, then the instance is added to that node, and the concept definition is expanded. Otherwise, EPAM re-sorts that instance through the hierarchy, making a new disjunct at some point where the attribute values differ.[15]

Lebowitz's (1985, 1986) UNIMEM system incorporates a number of advances over EPAM. First, every node in the hierarchy is a concept definition, and includes more than a single attribute. Thus, when classifying a new instance, this system uses a *polythetic* strategy, inspecting some number of attributes before making a decision. When an instance is incorporated into a concept, the system updates "confidence" values associated with each attributes at that concept by raising the confidence of the matching attributes, and lowering the confidence of any mis-matched attributes. This bookkeeping lets UNIMEM delete features with low confidence, potentially deleting entire classes, if they have no features with strong confidence. Finally, this system allows for *clumping*, or sorting instances to more than one class. However, UNIMEM does not allow the completely probabilistic classification suggested by Cheeseman et al. (1988).

As defined above, the incremental algorithm is a pure hill climber — it can get trapped in the same kind of local optima as iterative optimization methods. Fisher's COBWEB system (1987a, 1987b) added some operators to the algorithm that were designed to alleviate this problem. In addition to options a) and b) at step two, the system considers *merging* two existing classes and *splitting* a class into its children. These operators permit the system to move away from local optima, since they allow a form of backtracking through the space of possible concept hierarchies.

As described earlier, researchers in machine learning prefer algorithms that create intensional class definitions. Incremental algorithms modify these definitions with each new instance, rather than adding the instance to a membership list. This suggests that an incremental system need not store every instance. This ability leads to reduced memory costs, especially if the number of instances is very large. Developing ways to 'forget' instances, or avoid storing them at all, is a current research topic (Gennari et al., 1989).

4. Evaluating a clustering method

With this many clustering methods to choose from, one would hope for some principled way of comparing different methods. In general, we would like some quantitative measure for how well a given method has succeeded. Unfortunately, because the goals of clustering vary and are often very poorly defined, a single definitive measure is impossible. To some degree, any quantitative comparison of results is impossible because the goal of clustering is subjective. For example, some researchers prefer one clustering method to another simply because it produces a 'more intuitive' or 'more pleasing' set of classes. However, researchers with a more formal bias have found a number of ways to measure different aspects of a solution.

One measure used in discriminant analysis and in machine learning is to see how often a system classifies instances correctly. In some domains, the correct class is known beforehand, so the

15. This description of EPAM is a simplification that ignores aspects of the system that model human learning, and emphasizes the similarity to other incremental algorithms. See Gennari et al. (1989) for a more complete description.

hierarchy produced by some number of "training" instances can be evaluated by how correctly it classifies a number of "test" instances. For concept formation, the problem is that the 'correct' class is not known. At best, this measure can be used to compare an 'intuitive' set of classes with the new set produced from the data; it is usually *not* a good evaluator of the process that created the new classes.

One measure that can be used to compare hierarchies is "cophenetic correlation". This method compares the output of agglomerative algorithms, and is therefore used mostly by biologists. This measure uses correlation to compare the original similarity matrix with a new matrix that is derived from the final hierarchy. Instead of the similarity between instances i and j, an entry (i, j) in the new matrix has the value of the similarity measure when the class containing i was merged with the class containing j. These two matrices can then be compared by computing their correlation – a high correlation between entries suggests that the clustering method did a good job of capturing the information in the original similarity matrix. However, Aldenderfer and Blashfield (1984) point out that this measure is not statistically sound. The use of correlation assumes that the values in the matrices are normally distributed. Because the derived matrix is dependent on the original matrix and contains much less information, this is usually not the case.

As an alternative, Aldenderfer and Blashfield suggest that a better way to compare concept hierarchies is to use what they call a "Monte Carlo" method. There are three steps in this procedure. First, a set of random data is generated that is normally distributed and based on the averages of the original data. In effect, this represents the original instances, but grouped as a single class. Next, this random data set is clustered, resulting in a "base-line" hierarchy. Finally, the original hierarchy is compared to this base-line hierarchy (for example, by comparing an analysis of variance within each class). A large difference between hierarchies means that the algorithm has done a good job of finding classes from the data. Although the score resulting from this procedure has little absolute meaning, it can be used to compare a set of different methods – the method with the biggest difference is the 'best' for a given data set.

Rather than comparing hierarchies, there is a more general way of evaluating a clustering method. Instead of evaluating whether a classification is 'correct', the idea is to judge the 'usefulness' of that classification. Although this may seem difficult to do, one measurement of 'usefulness' is the degree to which the classes can predict attribute values of a new instance. Predictive ability is related to the 'recall' task in cognitive psychology, and has been used in machine learning (Fisher, 1987a; Gennari et al., 1989). This recall task occurs when an agent is given an incomplete set of cues (attributes) from a new instance, and must use its memory of past instances to recall values for the unspecified attributes of the instance. For machine learning, predictive ability can be used as a performance task to evaluate the learning system (as described in Section 1.1). In Chapter 4, I define this performance measure more precisely.

5. Chapter summary

The framework delineated in this chapter allows for a comparison of clustering methods across a wide spectrum of research fields. By describing a technique in relation to others proposed by different fields, one can focus on what is new and distinctive about the work. More importantly, an awareness of these related efforts allows the researcher to avoid duplication of work.

Although concept formation in machine learning offers some new insights to the clustering problem, there is certainly information to be gained from other clustering methods. Too often, the researcher in machine learning proceeds without any awareness of other potential solutions. This can mean that she may not apply an existing solution to a particular clustering problem, or worse yet, may present a 'new' approach that is identical or very close to an older solution to the same problem.

At the same time, machine learning certainly offers a new perspective for clustering methods. The biases it brings from cognitive psychology can be useful if applied to traditional cluster analysis methods. For example, the preference for low-cost, incremental methods may bring computational benefits without a loss of performance. Also, using predictive accuracy as a performance task seems to provide a useful method of evaluation in an otherwise unexplored area.

This survey of clustering methods is intended as the beginnings of a bridge between cluster analysis methods and efforts in machine learning. A recognition of these older efforts has been overdue in the machine learning literature, while an understanding of machine learning approaches to concept formation has been missing in other fields. In the chapters that follow, I describe a machine learning system, evaluate it with some clear performance tasks, and directly compare it to these earlier efforts.

CHAPTER 3

An Introduction to CLASSIT

My approach to concept formation is instantiated by the CLASSIT learning system. As stated in the first chapter, an implemented computer system is important for two reasons. First, it forces the researcher to fully detail the ideas and goals of her theory. Second, a running system is a tool for experimentation and careful modification, allowing for progressive refinements of the ideas that inspired the system.

This chapter describes CLASSIT at an introductory level: it does not consider performance abilities, nor a number of embellishments and details of the system. In order to appreciate the results of experimentation, the reader must have a thorough understanding of the system's operation. Many of the ideas described here are not new, and I try to provide pointers back to influential research and ideas. At the same time, I include forward pointers to the work presented in Chapters 4, 5 and 6 that explore the strengths and weaknesses of CLASSIT's abilities.

This chapter begins by describing CLASSIT's representation of input data and concepts. Next, I present the concept formation algorithm, defining both its learning and performance tasks. This section also presents the evaluation function used for learning, although I investigate alternative evaluation functions in Chapter 5. Finally, in order to demonstrate and clarify the operation of CLASSIT, I include a detailed example as the system learns from a sequence of playing ball descriptions.

1. Representation in CLASSIT

For CLASSIT, issues of representation can be divided into three areas. First, I present the representation used for instances – the input language for the system. Next, I describe the representation of the learned concepts – the knowledge for the system. Finally, I discuss the organization CLASSIT uses to arrange these concepts in memory.

1.1 Instance representation

As with most other work in concept formation (and cluster analysis) CLASSIT begins with the assumption that instances are described by a known set of attribute-value pairs. However, as described in Chapter 2, there are a number of different attribute types. Unlike most earlier work in cluster analysis and concept formation, CLASSIT uses instances that may be described by any combination of ordered (continuous or ordinal) and unordered (symbolic) attributes.

For example, if the system is learning from a set of playing balls, a golf ball might be described as {**weight: 46gr, diameter: 4.3cm, color: yellow, texture: dimpled**}. In this domain, two of the attributes are continuous and two are symbolic.[1] Allowing different types of attributes lets the system work with a wider variety of input domains.

Previous concept formation efforts (such as COBWEB or UNIMEM) focused on symbolic or, at best, ordinal attributes. However, from a psychological standpoint, such work ignores the ability of the human perception system to work with continuously-scaled measurements. Whenever a measurement is taken from an instrument, that value is most naturally represented as a continuous attribute. In contrast to machine learning research, work in cluster analysis often deals with continuous attributes, but rarely with data sets that are mixtures of symbolic and continuous information. There are a variety of domains that are most naturally represented using mixed continuous and symbolic data, such as diagnosis tasks. In Chapter 4, I present a number of real-world domains that are of this mixed format.

CLASSIT also allows instance descriptions to be *incomplete*. Simply stated, this means that not all attributes must be present for every instance. This lets the system work in domains with missing information. In real applications, data is often unavailable for a number of reasons. Additionally, this design allows for the ability to simply ignore some attributes (and treat them as missing) because they may be unimportant. This introduces the idea of *selective attention*, which is discussed in Chapter 6.

One limitation of this representation is that the attribute-value pairs are arranged in a flat list. In some domains, It may be desirable to represent instances with a *structured* representation. Although an earlier instantiation of CLASSIT (Gennari et al., 1989) included a limited approach to simple structured instances, the current system leaves this issue to future work. Thompson and Langley (in press) are exploring a more general solution to concept formation with structured instances (based in part on this work) with their LABYRINTH system.

1.2 Concept representation

Like Fisher's (1987a) COBWEB system, CLASSIT uses probabilistic representations for its concepts. That is, information about a concept is stored with probabilities: if x is a member of concept X, x will have value V_i for attribute A with conditional probability $P(A = V_i|X)$. Like other aspects of the system, this representation is inspired by work in cognitive psychology. Smith and Medin (1981) use the term *probabilistic concepts* to describe concept representations that include conditional probabilities.

There are two parts to such a concept representation. First, there are conditional probabilities stored for every attribute. Second, each concept includes the probability of the concept as a whole: the *base rate*, $P(X)$, for concept X. This is simply defined as the percentage of instances (relative to the parent) that are sorted to that concept.

Although conditional probabilities are easy to calculate with symbolic attributes, some modification is required when using continuous attributes. In particular, since the probability of a continuous attribute having any particular value is zero, one must assume some probability density

1. CLASSIT does not make any distinction between continuous and ordinal attributes, nor between symbolic attributes and binary, unordered attributes.

model. Therefore, a concept stores information about each continuous attribute in terms of a probability density function. For CLASSIT, I assume that concepts are based on normal distributions; thus, the parameters for each density function are the mean, μ and the standard deviation, σ, of that concept's population.

Although this assumption does restrict the range of concepts possible, Fried and Holyoke (1984) suggest two arguments for using normal distributions. First, such distributions seem to appear frequently in the natural (ecological) world. Second, the authors point to psychological evidence about basic-level categories that suggests human categories can be approximated by normal distributions.

The type of information stored for each attribute depends on the type of that attribute. If the attribute is symbolic, CLASSIT simply stores a set of counts for each attribute-value, from which it can compute conditional probabilities. If the attribute is continuous, the system keeps the mean and standard deviation for the population of member instances.[2] Example concepts can be seen in Figure 1 of the next subsection.

Although such a hybrid representation may appear cumbersome, it seems the most natural choice when one is working with both continuous and symbolic attributes. The only alternative is to convert attributes from one type to another. There are a large variety of conversion methods available. For example, Lebowitz's (1985) UNIMEM system converted all ordered attributes into 'symbolic' ranges, thereby allowing a symbolic clustering method to accept numeric attributes. However, any conversion method is necessarily imperfect: it must either lose information or add unwarranted information to the data. If it is possible to use both symbolic and continuous data directly, there is no reason to bother with imperfect conversion methods. Chapter 4 presents experimental evidence for this claim.

1.3 Concept organization

As with most machine learning approaches to concept formation, CLASSIT organizes its knowledge into a hierarchy.[3] Each node in the hierarchy is a concept, and nodes are partially ordered from general to specific. Hence, leaf nodes often correspond to single instances, whereas nodes higher in the hierarchy correspond to more general concepts, with the root node summarizing all instances that the system has observed.

Figure 1 presents the simple concept hierarchy that CLASSIT builds when presented four examples of playing balls. Each instance is described by the four attributes mentioned earlier: weight, diameter, color, and texture. Hence, each node in the hierarchy includes information about each of the attributes: the mean and standard deviation if the attribute is continuous, and the conditional probability for each observed value if the attribute is symbolic. In this case, there are three golf balls clustered under concept C_2, and one ping-pong ball as concept C_3. Note that the latter node

2. In order to compute σ incrementally, CLASSIT stores the count, the sum of values, and the sum of the squares. From these, σ and μ can be quickly computed when needed.
3. As described in Chapter 2, Feigenbaum's EPAM (1963), Michalski and Stepp's CLUSTER/2 (1983a), Lebowitz's UNIMEM (1985), and Fisher's COBWEB (1987a) all build concept hierarchies of some sort. Exceptions include Anderson and Matessa's work (1990) and Cheeseman et al.'s (1988) work with AUTOCLASS.

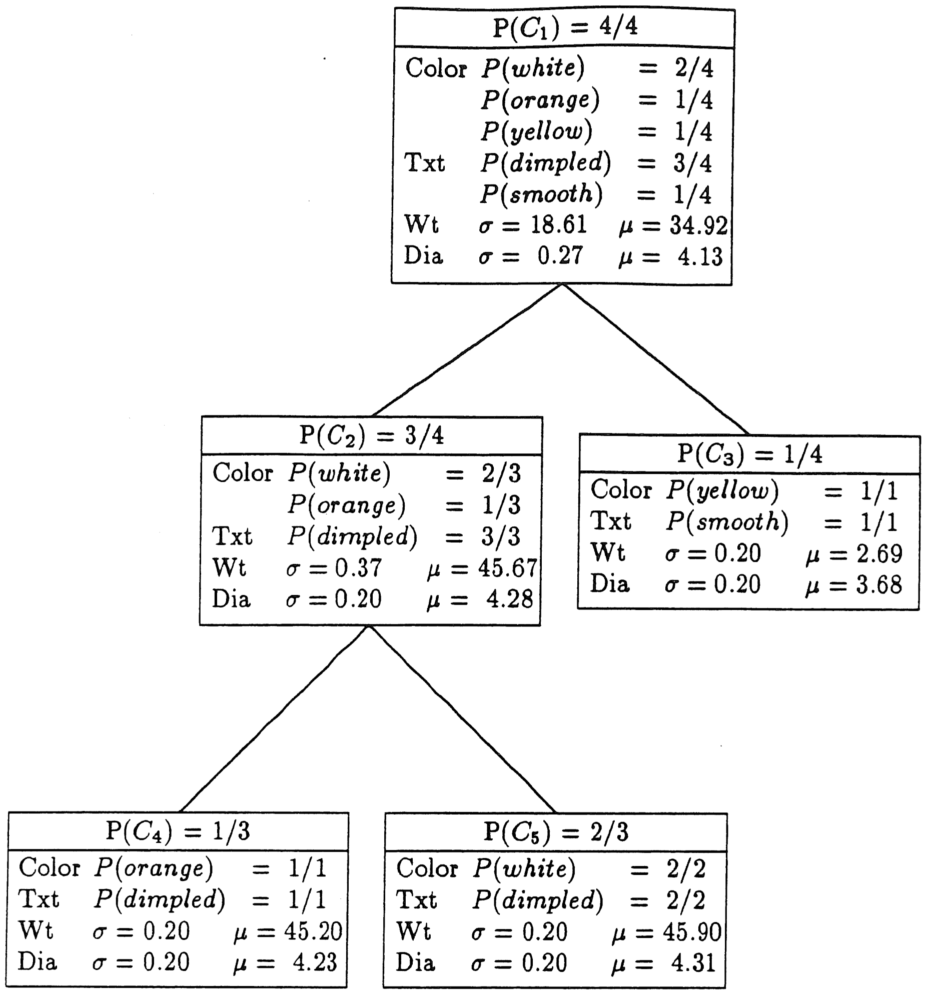


Figure 1. A simple CLASSIT concept hierarchy.

is a singleton class – a concept with only one member instance.[4] Syntactically, this concept is similar to a simple description of the instance. However, the semantics for this node are the same as any other concept in the hierarchy.

At the second level in this hierarchy, the golf balls are divided into two subclasses: orange ones and white ones. Note that the probabilities for each symbolic attribute are conditional given membership in that class, and that the probabilities for each class are relative to their parents. Hence, the singleton class C_4 has a probability of $\frac{1}{3}$, and not $\frac{1}{4}$. Also, note that the leaf node C_5 is not a singleton class; its two members are so similar that CLASSIT chose not to further subdivide these instances, and therefore forgets their particular attribute values. The next section describes how this can occur.

This type of concept hierarchy sets machine learning research in concept formation apart from

4. In Figure 1, the σ value for any numeric attribute of a singleton class is 0.2. Section 2.2 explains the origin of this number.

older work in cluster analysis. Such methods produce either a flat list of classes or a binary tree from which classes must be somehow extracted. A general to specific hierarchy (with an arbitrary branching factor) allows concepts to be created at more than one level of generality. This organization also allows for an efficient organization of learned concepts. That is, any concept may be quickly retrieved by descending down some path from the root node, rather than inspecting all known concepts.

One limitation of this type of memory structure is that there is only a single path to each concept. Rather than a tree of concepts, a more general structure would be a directed acyclic graph of concepts. This would allow different chains of reasoning (represented by different paths through memory) to lead to the same concept. Chapter 7 discusses such an extension to CLASSIT in more detail.

2. CLASSIT's algorithm and evaluation function

After describing the structure of CLASSIT's concepts, and the memory it uses to store and organize concepts, I am now ready to describe the mechanisms that the system uses to acquire the concept hierarchy. As with the other clustering methods presented in Chapter 2, I describe my approach to concept formation in two parts: the algorithm and the evaluation function.

2.1 The CLASSIT algorithm

CLASSIT uses an incremental algorithm to acquire its concept hierarchy. This means that that its algorithm can be roughly described as in Section 4.3 of Chapter 2:

```
For a new instance I, and some concept hierarchy:
    1. Incorporate I into the root node.
    2. Either: a) incorporate I to an existing child concept, or
               b) create a new child concept based on I.
    3. Unless a new child is added, recurse on the selected child
       concept.
```

From a machine learning perspective, one of the most important aspects of this algorithm is that the learning method (concept formation) and the performance method (classification of an instance) are completely intertwined. Whenever an instance is classified, learning also occurs. The new instance is incorporated into a number of concepts, thereby modifying their definitions, and it may cause entirely new concepts to be added to the hierarchy, as in Step 2b above. Conversely, learning only occurs when new instances are classified.

Table 2 presents a more detailed outline of the central learning and classification algorithm for CLASSIT. As with the simple description of incremental concept formation, the new instance I is recursively sorted through the hierarchy of concepts (nodes). The recursion terminates when the system decides to create a new concept based on the instance I.[5] This can occur in two ways.

5. The recursion can also halt when the score is 'high enough' – I will discuss this momentarily.

First, if the current node N has no children (a leaf node), then the hierarchy is extended down by adding the new instance below N (see the definition of **fork-node** in Table 3). Second, if I is sufficiently distinct from the existing children of N then the system places I as a new child of N.

Table 2. The top-level CLASSIT algorithm.

```
Input:    The current node N of the concept hierarchy.
          An unclassified (attribute-value) instance I.
Actions: Modifies the concept hierarchy to include the new instance.
Top level call: classit(Root, I).

Classit(N, I)

  If N is a leaf node,
  Then Fork-node(N, I)
       Incorporate(N, I).
  Else
    Incorporate(N, I).
    For each child C of node N,
         Using some evaluation function,
         compute the score for placing I in C.
    Let Best be the node with the highest score, Best-val.
    Let Second be the node with the second highest score.
    Let Disj be the score for adding I as a new singleton child of N.
    Let Merge be the score for merging Best and Second into one node.
    Let Split be the score for splitting Best into all its children.
    If Best-val is the best score,
         Then if Best-val is high enough
               Then Incorporate(Best,I),
               Else Classit(Best, I).
    Else if Disj is the best score,
         Then place I by itself in a new singleton child of N.
    Else if Merge is the best score,
         Then let Merged be Merge(Best, Second, N).
               Classit(Merged, I).
    Else if Split is the best score,
         Then Split(Best, N).
               Classit(N, I).
```

At each level, the instance I is incorporated into the chosen concept, N. This means that all the attributes defining N and observed in I are updated: for symbolic attributes, the probabilities for each value are modified; for numeric attributes, the means and the standard deviations are updated. In this way, the definition of N is modified to account for its new member, I. Note that N is defined intensionally: no list of instances is included in N, and I's particular attribute values are not stored in N.

Next, CLASSIT chooses among a set of four operations: selecting an existing child class, creating a new disjunct, *merging* two classes, or *splitting* a class. The system uses its evaluation function (defined in the next section) to compute scores for each of these alternatives, and then carries out the action with the highest score. These alternatives are learning operators: they are methods of moving through the search space of possible concept hierarchies. As discussed in Chapter 1, CLASSIT carries out an incremental hill-climbing search for the best concept hierarchy (Langley et al., 1987).

In order to compute a score for the first operator, the system considers adding the new instance I to each child node in turn. If the system does choose to incorporate I into the best existing child, then it recurses on that node, unless the score is so high that further categorization is deemed unnecessary. This occurs by means of a system parameter called the *recognition criterion*; the point at which the system 'recognizes' an instance I as being indistinguishable from a concept N.

The user provides this parameter to the system as a percentage of the maximum value that the evaluation function can return. The recognition criterion lets the system 'forget' those instances that are similar enough to an existing concept. It is applied both to this first operator and in the procedure `fork-node`. The latter use lets this parameter indirectly control the depth of the tree: new levels are not added if instances are recognized as the same as existing leaf nodes. In either case if the recognition criterion suggests that the instance has been 'recognized', then the recursion terminates, and the system is ready for the next instance.

As mentioned earlier, CLASSIT chooses the second operator (create disjunct) when I is sufficiently distinct from all existing children. Just as `fork-node` controls the depth, this operator affects the breadth of the tree. Also, this operator provides another stopping condition for the classification process; after making a disjunct, the system is done with I, and can begin classifying the next instance.

Table 3 includes definitions of the third and fourth operators for CLASSIT: merging and splitting nodes.[6] These two operators are an important addition to the basic incremental algorithm, since they let the system reorganize the structure of its concept hierarchy. In general, because CLASSIT carries out a hill-climbing search, it may become stuck at a local optimum of the evaluation function. Because of this problem, the system may learn different hierarchies for different orderings of the same data. These two operators can help the system recover from these *order effects*. Because they reorganize the tree, split and merge let the system recover from earlier decisions, and restart the search for the best concept hierarchy.

For example, if initial instances are all of the same class C_1, CLASSIT may build a hierarchy that divides instances into some number of closely related subclasses. If later, the system observes

6. Fisher (1987a) originally introduced these operators in his COBWEB system.

Table 3. Auxiliary CLASSIT procedures.

```
Merge(N1, N2, Parent)
   Add M, a new child of Parent.
   Initialize M's conditional probabilities to
      the weighted average of N1 and N2.
   Set M's children to (N1, N2).
   Delete N1 and N2 from the children of Parent.
   Return M.

Split(N, Parent)
   Add the children of N to the children of Parent.
   Remove N from the children of Parent.

Incorporate(N, I)
   Increment the member count of N.
   For each attribute in I,
     If the attribute is symbolic,
       update the count for the value in I
     Else (the attribute is numeric)
       update the mean and standard deviation
       to include the value in I.

Fork-node(N, I)
   If I is different enough from N
   Then
     Create two children of N:
     One child is a copy of N.
     The other is a singleton child based on I.
   Else do nothing.
```

instances from a very different class C_2, then it needs to merge together the C_1 subclasses (see Figure 2a). This lets C_1 and C_2 be defined at the same level, and pushes the C_1 subclasses to a lower, more specific level of the concept hierarchy.

Conversely, instances from different classes may be placed in the same concept node. This can happen if the system assumes some single instance from C_2 is noise, and is a member of C_1. If additional instances show that there really is a separate class C_2, then this class will be built up as a child of C_1. Eventually, the system should prefer to split node C_1 (see Figure 2b), raising the

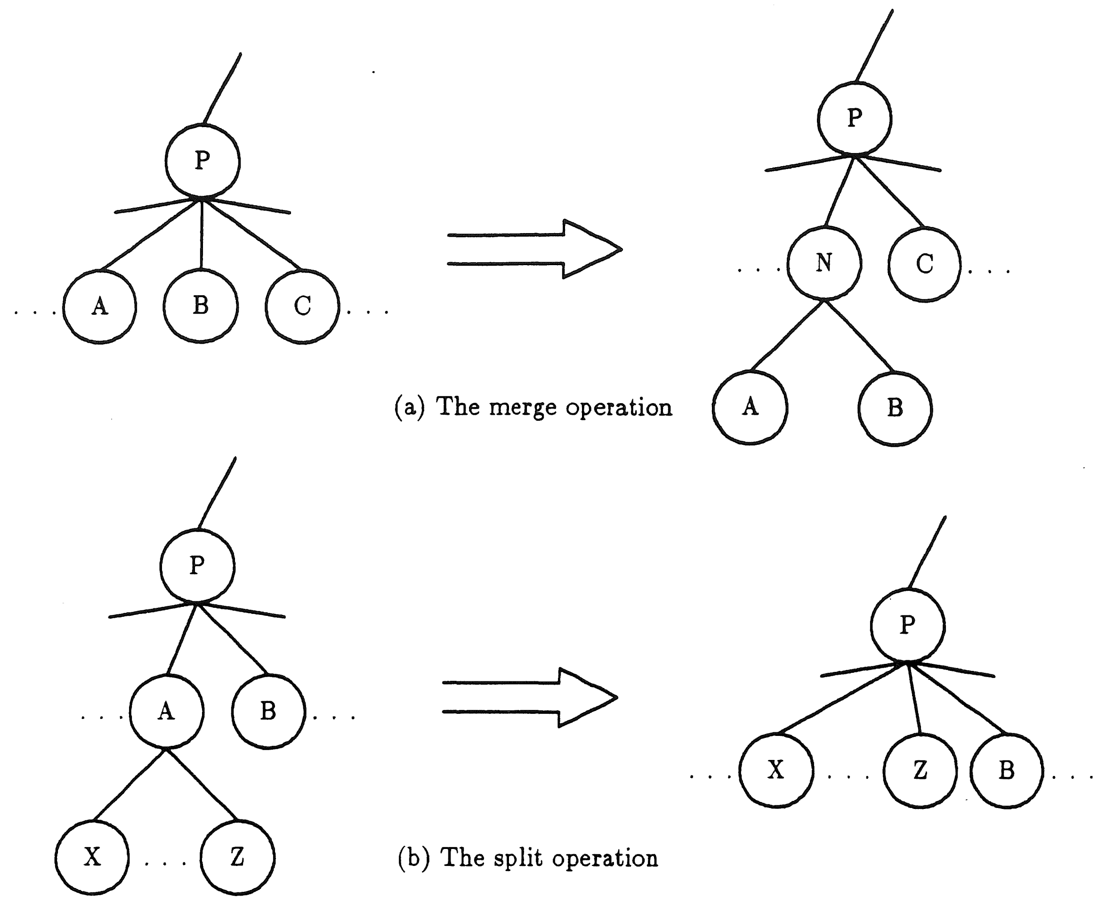


Figure 2. Merging and splitting nodes in a concept hierarchy.

subclasses to a higher level in the hierarchy. As one can see from the figure, merge and split are inverse operations.

In general, there are a variety of operations that could be used to reorganize a concept hierarchy. For example, Wallace (1989) suggests a general *grab*(c, w) operation that lets any node w become a new sibling of some other node c. Since this can be defined for every pair of nodes in the tree, the operation can be attempted many times whenever learning occurs. This is similar to Fisher's (1987a) *promote* operator, which raises a node to some other location in the tree. The principal difference between the promote and grab operators is that Fisher strongly limits the use of his operator, while Wallace allows the grab operation to be used on many pairs of nodes.

In contrast, CLASSIT only uses the merge and split operators, and in a strongly limited way. Table 2 shows that only a single merge (of the best and second-best children) and a single split (of the best child) is attempted at each level of the hierarchy. I chose this design to keep the system simple; until there is strong evidence suggesting that CLASSIT needs additional reorganization operators, I see no need to add additional mechanisms.

The best way to reorganize a concept hierarchy remains an open question. Chapter 4 presents some evidence that CLASSIT's method works well, but it is possible that other schemes will work even better. For example, a simple way to expand CLASSIT's reorganizing ability is to consider more than a single merge at each level, or consider merging more than two nodes. Finally, I should point out that these operators often make little or no change to the information in the hierarchy, but rather rearrange the information into a more efficient form. This has implications when testing the value of these operators.

2.2 The category utility evaluation function

As described in Chapter 2, a clustering method consists of an algorithm and an evaluation function. These parts should be at least somewhat interchangeable; one should be able to design a set of related clustering methods by using different evaluation functions in a single algorithm. This is true to some degree of CLASSIT: the algorithm does not specify how to compute the scores needed to make a decision. In theory, an arbitrary evaluation function may be used at this point in the algorithm. I will return to this idea in Chapter 5, but for now, I present the evaluation function used by the basic system.

A critical aspect of an evaluation function for CLASSIT is that it must be able to work with both symbolic and continuous attributes. The basic approach is analogous to that used by Gower's similarity metric. The system uses a symbolic measure for symbolic attributes, a numeric measure for numeric ones, and averages the values across all attributes. This is one of the simplests methods for combining attribute types, and therefore seems like a good starting point. Since this approach averages over two different evaluation functions, the functions should be as closely related as possible; in fact, CLASSIT uses a numeric evaluation function that can be derived from COBWEB's category utility function, as described in Chapter 2.

I have used this function as a starting point for two reasons. It has been used with some success by Fisher, and it has some psychologically appealing features. Not only does it allow for probabilistic concepts, but it also can be used to model some well-known psychological phenomena such as typicality and basic level effects (Fisher, 1987b; Fisher & Langley, in press). As presented in the Chapter 2, category utility for a single symbolic attribute, i, can be expressed as:

$$Category\ Utility(i) \ = \ \frac{\sum_{j=1}^{J}[P(C_j)\sum_{v=1}^{V}P(x_{iv}|C_j)^2] \ - \ \sum_{v=1}^{V}P(x_{iv})^2}{J} \quad ,$$

with summations over J classes and V values. It is the latter, innermost summations that must be generalized for continuous attributes. Since continuous attributes have an infinite number of possible values, the terms

$$\sum_{v}^{V} P(x_{iv}|C_j)^2 \quad \text{and} \quad \sum_{v}^{V} P(x_{iv})^2 \quad ,$$

must be re-written for continuous attributes. Both of these terms are a sum of squares of the probabilities of all values of an attribute. The former includes probabilities conditional on membership in a particular class, C_j, whereas the latter does not include class information. The second term is equivalent to the probability at the parent concept, since that node includes all instances in the partition regardless of class membership.

In order for these terms to be applied to continuous attributes, summation must be changed to integration, and some assumption must be made about the distribution of values. Without any prior knowledge about the distribution of an attribute, a reasonable assumption is that its distribution follows a normal curve. (This is the same assumption made in Section 1.2 for defining probabilistic concepts). Thus, the probability of a particular attribute value is the height of the curve at that value, and the summation of the square of all probabilities becomes the integral of the normal distribution squared. For the first term, above, the distribution is for a particular class, whereas the second term must use the distribution at the parent. In either case, this integral evaluates to a simple expression:

$$\sum_{v}^{V} P(x_{iv})^2 \quad \Leftrightarrow \quad \int \frac{1}{\sigma^2 2\pi} \; e^{-(\frac{x-\mu}{\sigma})^2} \, dx \quad = \quad \frac{1}{\sigma}\frac{1}{2\sqrt{\pi}} \quad ,$$

where μ is the mean and σ is the standard deviation. Finally, since this expression is used only for comparison, the constant term $1/2\sqrt{\pi}$ can be discarded.

In summary, one can replace the innermost summations from category utility with the term $1/\sigma$. Thus, the evaluation function used by CLASSIT for a continuous attribute i is

$$\frac{\sum_{j}^{J} P(C_j) 1/\sigma_{ij} \quad - \quad 1/\sigma_{pi}}{J} \quad ,$$

where J is the number of classes in the partition, σ_{ij} is the standard deviation for a given attribute in a given class, and σ_{ip} is the standard deviation for a given attribute in the parent node. Hence, the evaluation function over all attributes can be defined as

$$\frac{\sum_{i}^{I} \sum_{j}^{J} P(C_j) Info(C_{ij}) \quad - \quad Info(C_{ip})}{I \cdot J} \quad . \tag{1}$$

$Info(C)$ is a function that measures the value or quality of a class C. For a symbolic attribute i (with V values)

$$Info(C_i) = \sum_{v}^{V} P(x_{iv}|C)^2 \quad ,$$

and for a continuous attribute i

$$Info(C_i) = 1/\sigma_{iC} \quad .$$

This evaluation function is analogous to the function used by COBWEB; it is a transformation of category utility. For each attribute i this function sums over every child concept, C_j, and subtracts the information at the parent, C_p. Hence, this measures the *gain* in $Info(C)$ from parent to child levels of the hierarchy.

Unfortunately, this transformation introduces a problem for singleton classes. For these classes, the standard deviation of any numeric attribute is zero, and the value of $1/\sigma$ is therefore infinite.

CLASSIT resolves this problem by employing the notion of *acuity*, a system parameter that specifies a minimum standard deviation. This limit corresponds to the notion of a 'just noticeable difference' in psychophysics – the lower limit on our ability to make perceptual discriminations. Because acuity strongly affects the score of new disjuncts, it indirectly controls the breadth or branching factor of the concept hierarchy, just as the recognition criterion controls the depth of the hierarchy.

For both symbolic and continuous attributes, $Info(C)$ has the highest value when C is a singleton class. This seems to imply that category utility would always prefer to make disjuncts, creating a degenerate hierarchy where there is a singleton child for every instance observed. However, the division by the number of classes J biases the system toward hierarchies with smaller branching factors. Hence, as long as there is a finite maximum value for $Info(C)$, the evaluation function will at some point prefer not to add a new disjunct to the hierarchy. For symbolic attributes, this maximum value is 1.0, whereas for continuous attributes it is $1/acuity$.

3. A detailed example

A description of CLASSIT would be incomplete without a step-by-step example of the system's construction of a concept hierarchy. In fact, the examples used to instantiate a new theory or system are sometimes the most important (and frequently the most well-known) aspect of the description. In this section, I will step through an execution of the system as it learns about the domain introduced earlier in the chapter: playing balls described by weight (in grams), diameter (in centimeters), color, and texture.

CLASSIT usually begins with an empty hierarchy, and hence no information about the domain.[7] Therefore, the root node is instantiated as a singleton class after observing the first instance as shown in Figure 3a. Notice that the numeric attributes have $\sigma = 0.2$; this shows that for this execution, acuity is set at 0.2.

When the second instance is observed, CLASSIT incorporates it into the root and the two instances are used to extend the tree downward, making singleton classes at level one of the hierarchy (Figure 3b). The only other possible action at this point would occur if the instances were so similar that the recognition criterion parameter forced the instances together without creating a more specific level in the hierarchy. For this trial, the recognition criterion is set at 0.90; the instance must get a score of more that 90% of the maximum possible score in order to halt the classification process.

The third, fourth, and fifth instances are additional yellow and white ping-pong balls. In each case, the systems adds the new instance to concept C_1 (for yellow ping-pong balls) or C_2 (for white ping-pong balls). Also, each of these instances are similar enough that the recognition criterion does not let the tree extend beyond level one (where the root node is at level zero). After five instances, the classes C_1 and C_2 appear as in Figure 4. Upon encountering the sixth instance (an orange golf ball), CLASSIT makes a new singleton class, C_3, based on this instance. This hierarchy

7. Of course, CLASSIT may also be initialized with background information about a domain in the form of an initial concept hierarchy. Chapter 7 discusses this possibility further.

(a) First instance:				(b) Second instance:			
	color	=	white		color	=	white
	texture	=	smooth		texture	=	smooth
	weight	=	2.67		weight	=	2.64
	diameter	=	3.69		diameter	=	3.68

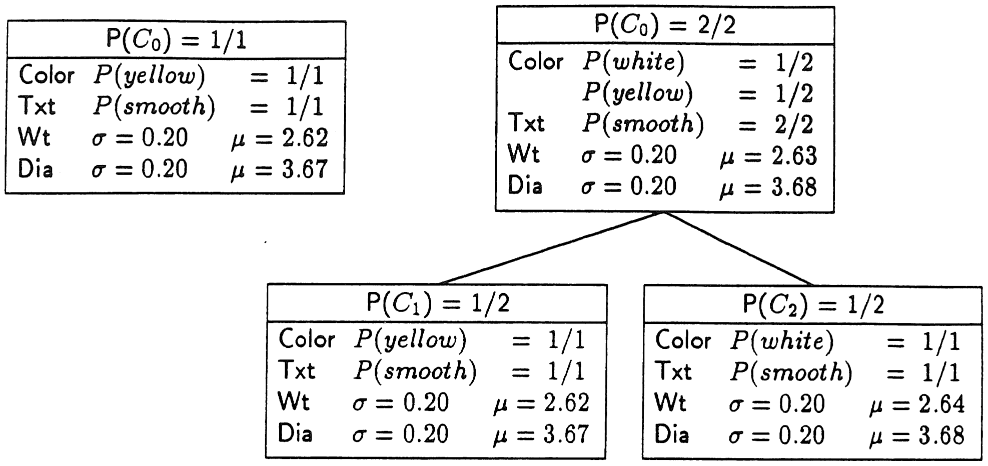

P(C_0) = 1/1
Color $P(yellow)$ = 1/1
Txt $P(smooth)$ = 1/1
Wt $\sigma = 0.20$ $\mu = 2.62$
Dia $\sigma = 0.20$ $\mu = 3.67$
P(C_0) = 2/2
Color $P(white)$ = 1/2
$P(yellow)$ = 1/2
Txt $P(smooth)$ = 2/2
Wt $\sigma = 0.20$ $\mu = 2.63$
Dia $\sigma = 0.20$ $\mu = 3.68$
P(C_1) = 1/2
Color $P(yellow)$ = 1/1
Txt $P(smooth)$ = 1/1
Wt $\sigma = 0.20$ $\mu = 2.62$
Dia $\sigma = 0.20$ $\mu = 3.67$
P(C_2) = 1/2
Color $P(white)$ = 1/1
Txt $P(smooth)$ = 1/1
Wt $\sigma = 0.20$ $\mu = 2.64$
Dia $\sigma = 0.20$ $\mu = 3.68$

Figure 3. The first two instances in CLASSIT's hierarchy.

Sixth instance:			
	color	=	orange
	texture	=	dimpled
	weight	=	45.20
	diameter	=	4.23

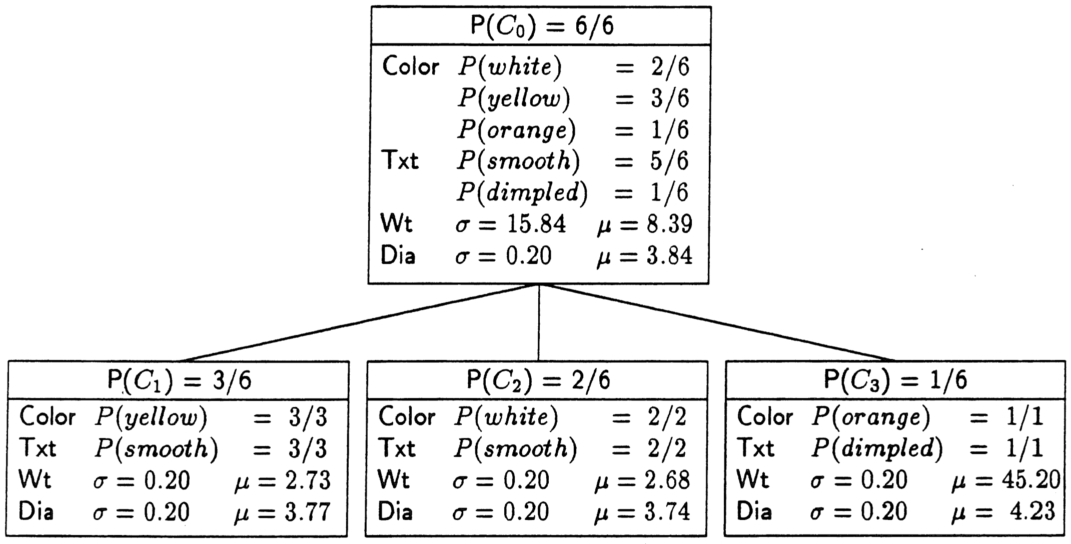


Figure 4. The effect of adding a new disjunct to CLASSIT's concept hierarchy.

Seventh instance:			
	color	=	white
	texture	=	dimpled
	weight	=	46.10
	diameter	=	4.36

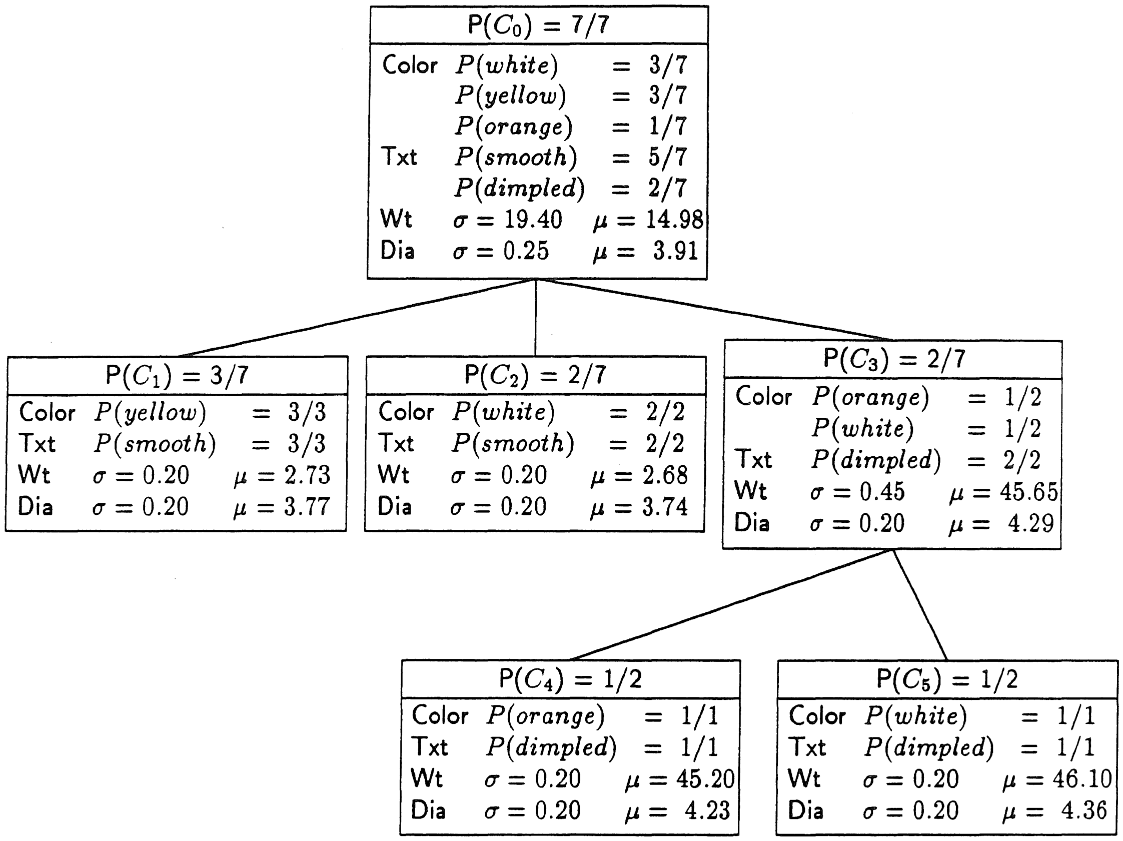


Figure 5. The result of extending CLASSIT's hierarchy down a level.

is shown in Figure 4. Although the diameter of the sixth instance is similar to other instances, all other attributes are different, which leads the system to store it as a new disjunct.

The seventh instance is a second golf ball. Since this instance is similar to the previous golf ball, CLASSIT chooses to add this instance into class C_3. However, the new instance is distinct enough (e.g., it is white rather than orange) to warrant extending the hierarchy downward to level two (it applies the procedure **fork-node** to C_3). Figure 5 shows the hierarchy after the system has processed the seventh ball. At this point, the evaluation function would prefer a hierarchy with two rather than three level one classes, corresponding to a class of ping-pong balls and a class of golf balls. However, this requires merging the two ping-pong classes, and this option is not considered until a new ping-pong ball is seen.

CLASSIT is able to apply the merge operator after observing the eighth instance. For this instance (a yellow ping-pong ball), the system finds that C_1 is the best match and C_2 the second best, but the score resulting from merging these classes is better than all other options. Therefore, a new

Eighth instance:

color	=	yellow
texture	=	smooth
weight	=	2.69
diameter	=	3.68

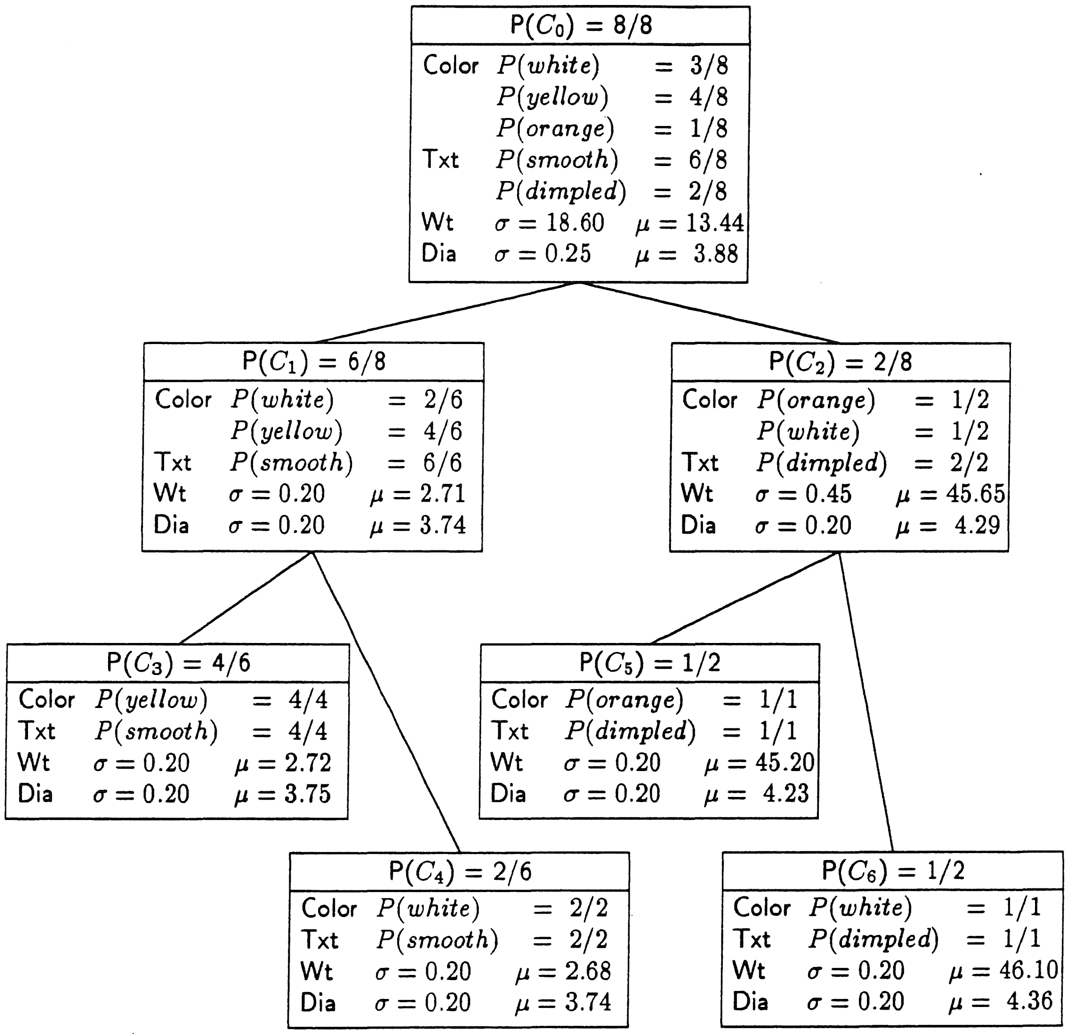


Figure 6. The result of merging two classes in the concept hierarchy.

node, C_3, is created by merging C_1 and C_2, as shown in Figure 6. Note that the old classes are pushed down to level two, and that the new instance is added into class C_3 before the recognition criterion stops further descent.

In order to demonstrate the split operator, we need to look at the hierarchy after CLASSIT has seen a number of very different objects, such as volleyballs. In this trial, the first volleyball is instance number nine. Despite the fact that this is a very different instance, the system decides to add it into the golf ball class, making a disjunct at level two rather than at level one. After all, it is possible this is the result of measurement error, or that it is simply a very large, very unusual golf ball. However, instances ten and twelve are additional volleyballs, and instance eleven is another golf ball. These instances strengthen the two subclasses below C_2, producing the hierarchy shown in Figure 7. (To reach this point, merge operators were used at level two with the eleventh and twelfth instance.)

Finally, with the thirteenth instance (another volleyball), the system applies the split operator. This causes all nodes under the old C_2 concept to be promoted, and C_2 is discarded as no longer useful. This tree is shown in Figure 8. Note the twelfth instance has been added to the new C_2, the promoted class of volleyballs.

This step-by-step trace should make clear the basic CLASSIT algorithm. As each each instance is seen, the system sorts it through the current concept hierarchy. At the same time, each instance modifies the hierarchy by changing the concept definitions and by modifying the structure of the tree. I have demonstrated the four operators the system uses to search for the 'best' concept hierarchy: incorporation, creating a new disjunct, merging, and splitting. Although this domain is quite small, it should be clear that the algorithm can be applied to hundreds of instances, each described by a large set of attributes. Chapters 4, 5, and 6 will use much larger domains to evaluate the system's ability.

Twelfth instance:	color	=	yellow
	texture	=	smooth
	weight	=	2.69
	diameter	=	3.68

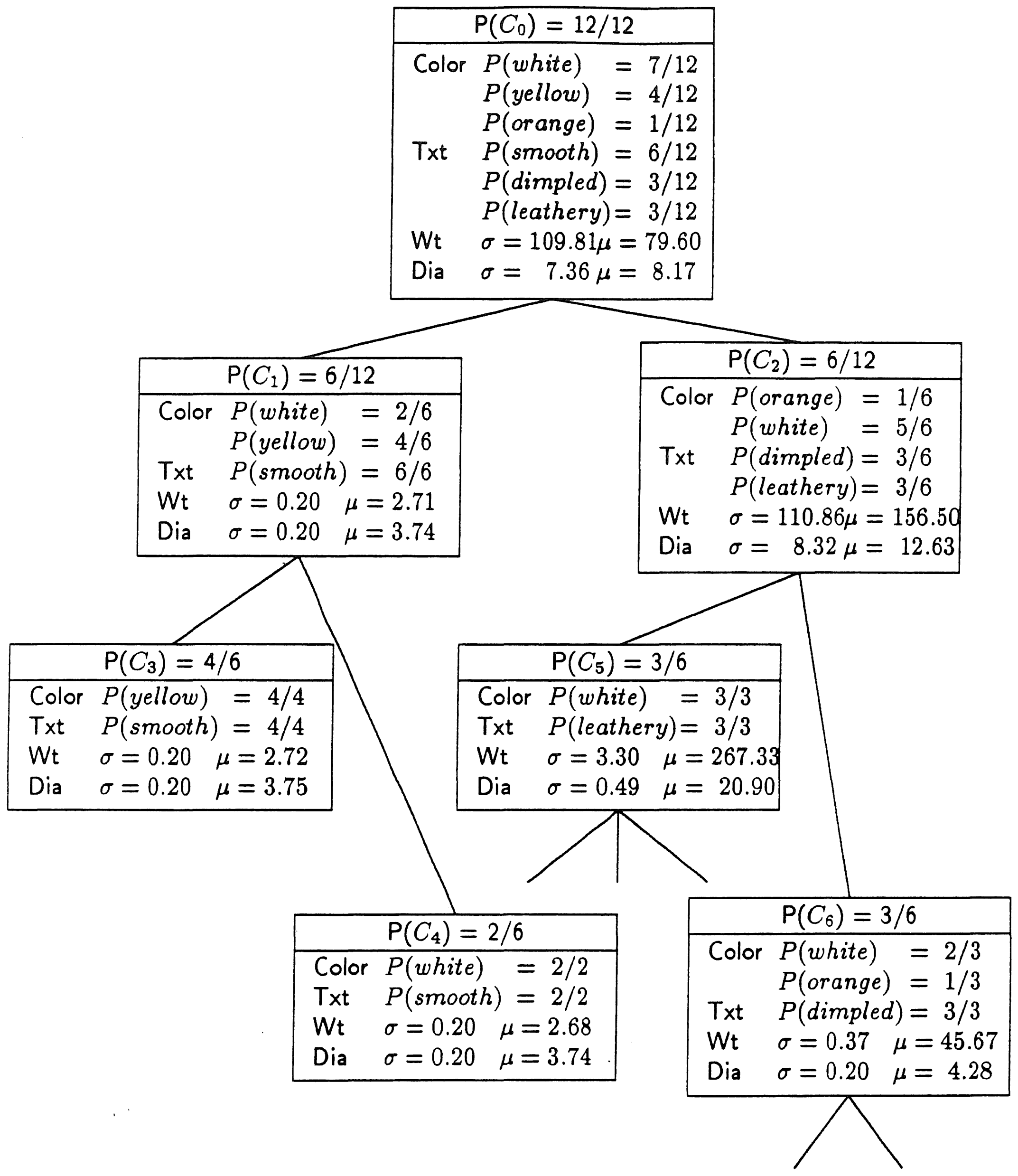


Figure 7. CLASSIT's concept hierarchy after 12 instances.

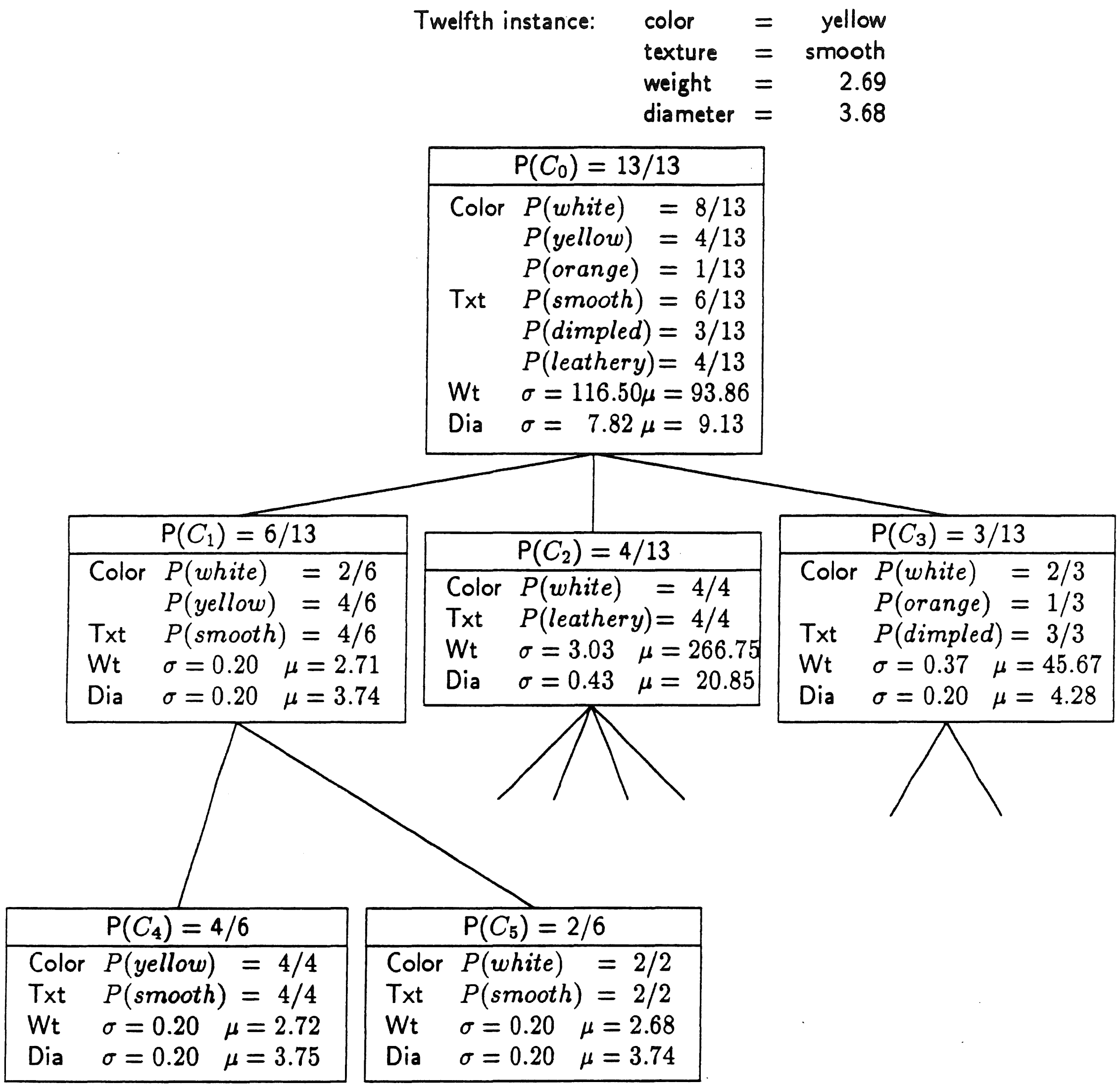


Figure 8. The effect of splitting a class in the concept hierarchy.

CHAPTER 4
Experimentation with Different Domains

In any scientific endeavor, after creating and presenting a new idea or theory, the value of the idea should be evaluated. Therefore, now that I have described a system for concept formation, the next step is to evaluate that system's learning ability. Purely mathematical ideas can be evaluated simply by analysis and proof. Unfortunately, despite advances in the theory of learning algorithms (Valiant, 1984; Haussler, 1987), most learning systems, including CLASSIT, are much too complex for complete analysis. Hence, the best available means for evaluating the system is with extensive, careful experimentation. Experimental studies are the focus of Chapters 4, 5, and 6.

Although this is clearly a good method for carrying out research, extensive experimentation is relatively new to machine learning. The experimental procedure I use roughly follows the outline presented in Langley (1988). I begin with a precise description of the performance tasks that I use to evaluate the system. The first section of this chapter presents the two measures used throughout the thesis: accuracy and efficiency. The remainder of the chapter looks at the effect on performance as the domain is varied. This contrasts with Chapter 5, where for a fixed set of domains, I look at a variety of modifications to the learning method.

There are a number of different ways to vary CLASSIT's domain. First, I look at a set of 'real-world' domains, to show that CLASSIT can actually learn useful concepts in a variety of real applications. Next, with the use of artificial domains, I show how some systematic changes in the domain (such as added noise or complexity) affect the system's learning ability. The following section is dedicated to domains that include both symbolic and continuous attributes. I discuss attribute conversion methods, and compare their use to CLASSIT's more integrated approach. The final section in the chapter discusses order effects: the effect of modifying the order of training instances on the system's learning and performance.

1. Performance measures

In order to carry out an experiment that evaluates the ability of a learning system, one must define a quantitative performance task. If learning is defined as improved performance over time, we must know exactly what is meant by 'performance'. In addition, if performance can be quantitatively measured, then one can determine how much learning has occurred. This measurement can then be used to evaluate a learning method across different domains and compare different learning systems on a given domain.

1.1 Predictive accuracy

To evaluate the CLASSIT system, the principal performance task I use is *predictive accuracy*. This task evaluates the usefulness of a concept by judging how accurately it can make predictions about future instances. More specifically, any concept can predict the attribute values of member instances; the system can infer that if x is a member of concept C, then x has certain attribute values (with some probability). One can then use the accuracy of these inferences to evaluate the learning system. This predictive accuracy task is an instantiation of a more general task for any inductive learning system: an application of the knowledge acquired during training to new, unseen test data. This approach for testing inductive systems was first popularized by Quinlan (1986).

Predictive accuracy was first defined and used in concept formation by Fisher (1987b). Fisher points out that the use of predictive accuracy is also motivated by the definition of the category utility evaluation function. This function was originally defined to maximize the the number of correct guesses that a class can make about an attribute value (Gluck & Corter, 1985). Hence, to evaluate the ability of a learning system that uses category utility, one should use this same prediction task.

Table 4 defines how predictive accuracy is measured to evaluate CLASSIT. For each test set instance I define predictive accuracy as the difference between the predicted and the actual values of the test attribute. Note that this difference is defined separately for symbolic and continuous attributes. For the former, either a one or a zero is returned, indicating a match or a mismatch; for the latter, the absolute value of the difference between `Predicted` and `Actual` is returned. This difference is then averaged over all test set instances, giving a performance measure for the system.[1]

The table shows that `BestClass` is found by making a call to CLASSIT$'$ with the test-set instance I. This call to CLASSIT$'$ is used to evaluate the performance of a static concept hierarchy. Therefore, in contrast to the usual CLASSIT system, learning is turned off as each test set instance is sorted through the hierarchy. Therefore, CLASSIT$'$ does not modify the concept hierarchy; it does not change any concept definitions, and it does not consider the merge and split operations.

This leaves only two choices as the system sorts I through the tree. Either it chooses an existing concept, in which case the algorithm continues recursively with that concept, or it prefers to make a new disjunct. In the latter case, rather than making the disjunct (and modifying the tree), CLASSIT$'$ returns the parent class as `BestClass`. As usual, the system also terminates when it reaches a leaf node or when the recognition criterion suggests that no further sorting is needed. This is slightly different from Fisher's (1987b) performance task: when COBWEB carries out this task, it does not consider disjuncts and does not have a recognition criterion parameter, so it always terminates at a leaf node.

As described in Chapter 2, this performance measure is related to the 'recall' task in cognitive psychology: given a set of cues (attributes) from a new instance, the system should be able to use its knowledge about past instances to recall the unspecified attributes of the instance. As I have defined predictive accuracy, there is only a single unspecified attribute. However, this task can be modified so that a number of attributes are set to unknown. I return to this issue in Chapter 6, which describes CLASSIT's ability with unknown attribute values in more detail.

1. Fisher (1987b) suggests going one step further and averaging this predictive accuracy over all attributes. Although this is certainly reasonable, it should not change relative performance, and therefore is not used in this dissertation.

Table 4. Measuring predictive accuracy.

```
Given:  Tree, a concept hierarchy.
        TestAtt, an attribute for prediction.
        TS, a set of test instances.

Predict (Tree, TestAtt, TS)
   For each instance, I, in TS do
       1. Save as Actual the value of TestAtt for I;
              set TestAtt to 'unknown'.
       2. With learning off, set BestClass to Classit'(Tree, I)
       3. Set Predicted to
           a) For continuous attributes,
                the mean of TestAtt in BestClass.
           b) For symbolic attributes,
                the most likely value of TestAtt in BestClass.
       4. Set Diff to the difference between Predicted and Actual.
   Return the average value of Diff over all instances in TS.
```

1.2 Retrieval efficiency

Although predictive accuracy is an important indicator of a learning system's ability, it overlooks the efficiency of the system. In general, there is often a tradeoff between efficiency and accuracy: it may be possible to arrive at an extremely accurate answer, but only with an impractical amount of processing time. Conversely, it is certainly possible to obtain an answer very efficiently that is not very accurate.

For these reasons I use *retrieval efficiency* as a secondary task for evaluating CLASSIT's ability. Rather than measuring the accuracy of predictions made by some class, this measures how much time is taken to classify the instance. As with predictive accuracy, this task uses an unseen set of test instances, and is averaged over those instances. This measure corresponds to a 'recognition time' test as used in cognitive psychology: given some knowledge about the environment (for this application, a concept hierarchy), measure the length of time needed to recognize or classify a new instance.

For CLASSIT, time can be defined as the average number of nodes inspected during classification. I also call this quantity the average work during testing, since it is related to the computational cost of classification. Table 5 defines `Work` for a concept hierarchy and a test set. As with predictive accuracy, work is an average over test-set instances, and includes a call to CLASSIT$'$ with learning turned off.

This performance task measures the quality of the concept hierarchy, rather than a measure of the predictive quality of the concepts in that hierarchy. In general, I use this measure whenever

Table 5. Measuring retrieval time.

```
Given: Tree, a concept hierarchy.
       TS, a set of test instances.

Work(Tree, TS)
  For each instance, I, in TS do Classit'(I, Tree)
    Count the number of nodes visited:
      For every recursive call to Classit'(I, N),
      Sum the number of children of N into Nodes-Seen.
  Return the average Nodes-Seen over all instances in TS.
```

looking at differences in structure and shape of the concept hierarchy. This arises in Chapter 5, when comparing different learning systems. Retrieval efficiency is also important for evaluating the attention mechanism I introduce in Chapter 6; in that chapter I will define recognition time in terms of both concepts and attributes observed during classification.

2. Performance with real domains

If one defines learning as improved performance over time, then the most basic experimental evaluation of a learning system is a *learning curve*, a graph that shows this improved performance. For CLASSIT, I will show learning curves with improved predictive accuracy over time. As the system observes instances, its concept hierarchy becomes a better and better representation of the class structure in the data, and therefore can be used to more accurately predict missing attributes. Additionally, the time needed to recognizing instances should not become too large as more and more instances are observed.

I begin this section with a description of the real-world domains used for testing CLASSIT. These domains are all in the public domain, and available from the UCI repository of machine learning databases.[2] After describing the databases, I present the experimental method and the resulting learning curve for each domain, as well as some discussion of results. Finally, I present efficiency results (recognition times) for these natural domains.

2. To obtain information about these databases, contact David Aha, Department of Information and Computer Science, University of California, Irvine, 92717, or send electronic mail to ⟨ml-repository@ics.uci.edu⟩

Table 6. The real domains used for testing CLASSIT.

Database	Symbolic attributes	Numeric attributes	No. of instances
Voting records	17	0	435
Glass	1	9	214
Heart-disease	8	6	263
Car insurance	8	16	205
Star LRS	1	93	531

2.1 The domains

I consider five real-world domains for evaluating CLASSIT. Together with experimentation on artificial domains, I believe this represents a good start toward demonstrating CLASSIT's ability with real problems. In comparison with earlier machine learning (and AI) research, this chapter uses a wide range of domains for evaluating the system. However, recent work suggests that five or six domains is near the minimum needed to evaluate the ability of a learning system (Buntine & Niblett, 1990). Table 6 summarizes the databases, showing the number and type of attributes for each. Appendix A presents sample instances from these domains, as well as additional details about each database.

VOTING DATABASE

The first database consists of voting records extracted from the 1984 U.S. congressional voting records by Jeff Schlimmer. There are a total of 435 instances (corresponding to the 435 representatives), and each instance is described by 17 binary attributes: 16 yea or nea votes, and one class attribute indicating 'democrat' or 'republican'. The voting attributes can also take a third attribute value: unknown (or an abstention). Since this is a form of missing information, I postpone the use of this attribute value until Chapter 6, where I describe CLASSIT's general approach to missing information. For this chapter, I only use instances from this domain that are complete, with no unknown attribute values.

This database includes a 'class attribute', which indicates the 'correct' class (political party, in this case) for each instance. I use this attribute for the performance task: CLASSIT will try to predict the political party of a congressman based on that person's voting record and on the learned concept hierarchy. This database has been used extensively in machine learning, especially by those in supervised learning (Aha & Kibler, 1989; Buntine & Niblett, 1990). For these researchers, the class attribute is treated specially, as feedback for the learning system. In contrast, CLASSIT treats all 17 attributes equally.

GLASS DATABASE

The second database contains 214 descriptions of glass fragments, as would be used by forensic science. Given a small fragment of glass, as is often found during criminal investigation, the problem for this domain is to identify the origin (window pane, car window, bottle, etc.) of that glass fragment. This database was originally collected by Brian German of the Home Office Forensic Science Service, U.K. There are nine numeric attributes such as refractive index and mineral content for various minerals, and one symbolic attribute. As with the voting database, the symbolic attribute indicates the 'class' of the instance; this will be the predicted attribute.

For this work, I use a binary class attribute: either the glass is a window fragment or it a non-window fragment. This follows the original use of this database by Evett and Spiehler (1987). In contrast, Buntine and Niblett (1990) allow the class attribute to take seven possible values. Both of these approaches seem to result in about the same accuracy, so I have chosen the binary representation for simplicity.

HEART-DISEASE DATABASE

The next database encodes heart disease information from 263 patients from the records of the Cleveland Clinic Foundation. There are 14 attributes per instance, of a variety of types. Attributes range from binary attributes such as male/female to continuous attributes such as blood pressure at rest. This database was collected by Dr. Robert Detrano, who converted all attributes to a numeric representation (Detrano et al., 1990). As much as possible, I have restored the attributes to their original mixed format: the database I use has eight numeric attributes and six symbolic ones.

The predicted attribute for this database is 'degree-of-disease'; indicating the degree of heart disease in the patient. This attribute is technically an ordinal attribute (see Chapter 2) with four values: $\{0, 1, 2, 3\}$. As with the previous database, I convert this attribute into a binary attribute: either sick $\{1, 2, 3\}$ or healthy $\{0\}$. Although this could be treated as a numeric attribute, there is more 'distance' between healthy and sick than there is among different degrees of sickness. This database has also been used by Aha and Kibler (1989).

CAR INSURANCE DATABASE

The fourth database describes data from a wide variety of cars from 1985, including attributes such as price, weight, wheel-base, and horsepower, as well as a 'risk factor' as assigned by automobile insurers. This database of 205 automobile descriptions was collected by Jeff Schlimmer. Unlike the other domains, there are no published results using this database.

This is probably due to a number of difficulties. First, there are a relatively large number of attributes of mixed types: there are 8 symbolic attributes and 16 numeric ones. Second, there is no definitive 'class attribute': although the risk factor is probably a good choice, the accuracy of this attribute is unknown. Finally, although it can be treated as symbolic, the risk factor is really an interval attribute, ranging from 3 to −2. For this chapter, I predict both risk factor and horsepower as numeric attributes.

STAR SPECTROMETER DATABASE

The final domain used to test CLASSIT is the largest: 531 instances of low resolution spectrometer (LRS) readings of stars. Each instance is described by 93 numeric attributes measuring spectral fluxes from different blue-band and red-band channel wavelengths. Also included is one class attribute, although the meaning (or validity) of this attribute is unclear. Although this particular set of instances has not been used elsewhere, Cheeseman et al. (1988) present qualitative results with a larger database (5425 instances) from the same domain.

As with the automobile database, this domain does not have a definitive class attribute. Therefore, in addition to predicting the class attribute, I would like to apply the performance task to other attributes in the domain. Unfortunately, because the semantics of this database are poorly understood, there does not seem to be a good way to choose among the remaining 93 attributes. For this chapter, I have selected two additional attributes at random, one each from the blue and red spectra.[3]

2.2 Learning results

I am now ready to describe the experimental method used to demonstrate that CLASSIT can learn from each of these domains. To produce learning curves, each domain must be broken into sets of training and test instances. In general, I have found that the system's concept hierarchy stabilizes after relatively few training instances. For most domains, performance is largely unchanged after learning from 30 to 50 instances. Hence, for these experiments, I use a training set much smaller than the size of the entire database. In order to be representative of the database, I use test sets of at least 30 instances.

Both the test set and the training set are selected at random from each domain, such that the two sets have no instances in common. This is an important requirement: if the test set were a subset of the training set, then a 'learning' system could perform well simply by saving all instances and parroting information back about each test set instance. This is the task for a database system, not an inductive learner. Because the test set instances are unseen, CLASSIT must carry out induction over the instances it has seen: it must generalize or extend its knowledge to account for the new instances.

There are some aspects of the CLASSIT system that affect learning ability but that I would like to ignore for these initial experiments. First, different orderings of the training set instances will affect the performance of the system. I will discuss this effect in detail in Section 5 of this chapter. However, for most experiments, all results are averaged over ten random orderings of the training

3. I have followed a suggestion from John Stutz (personal communication) that I use attributes from the higher and lower ends of the red and blue wavelengths.

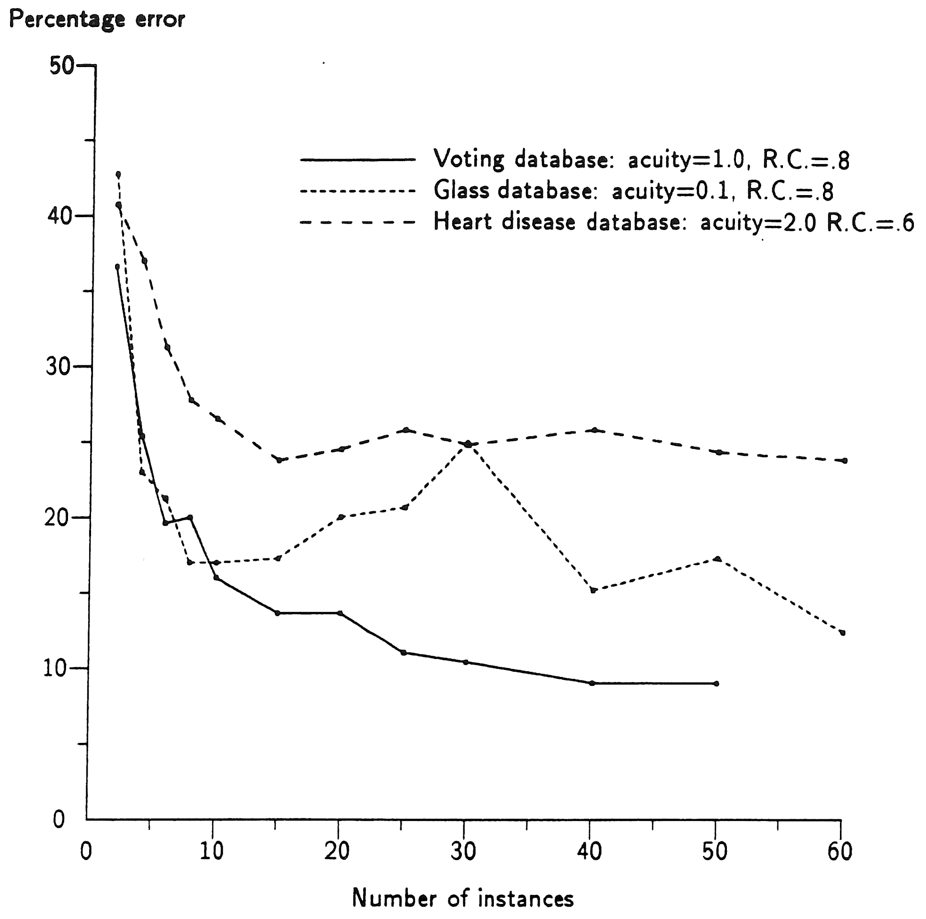


Figure 9. Learning curves for CLASSIT on three real domains.

set data. This makes it very unlikely that any result is due to a beneficial ordering that aids performance, or a misleading order that hinders performance.

Second, different settings of CLASSIT's two parameters, acuity and the recognition criterion, can affect performance. For these experiments, I will simply list the settings used for each domain. Obviously the degree to which CLASSIT must be 'tuned' to each new domain is an important concern; I will discuss this and explore the effects of parameter settings more completely in Chapter 5.

Figure 9 shows learning curves for the first three domains. As the graph shows, I used a training set of 60 instances for the glass and heart-disease domains, and 50 for the voting database. The figure also includes the parameter settings used for these experiments. I report performance as percentage error: at each test point, the percentage of incorrect predictions CLASSIT makes for the test-set instances. This is the measure I use whenever reporting predictive performance for a symbolic attribute.

All three curves shown in Figure 9 demonstrate some degree of learning. Unsurprisingly, asymptotic performance is best with the cleanest domain (the voting database), and worst with the noisiest domain (heart-disease). All three curves show that most learning has occurred by the

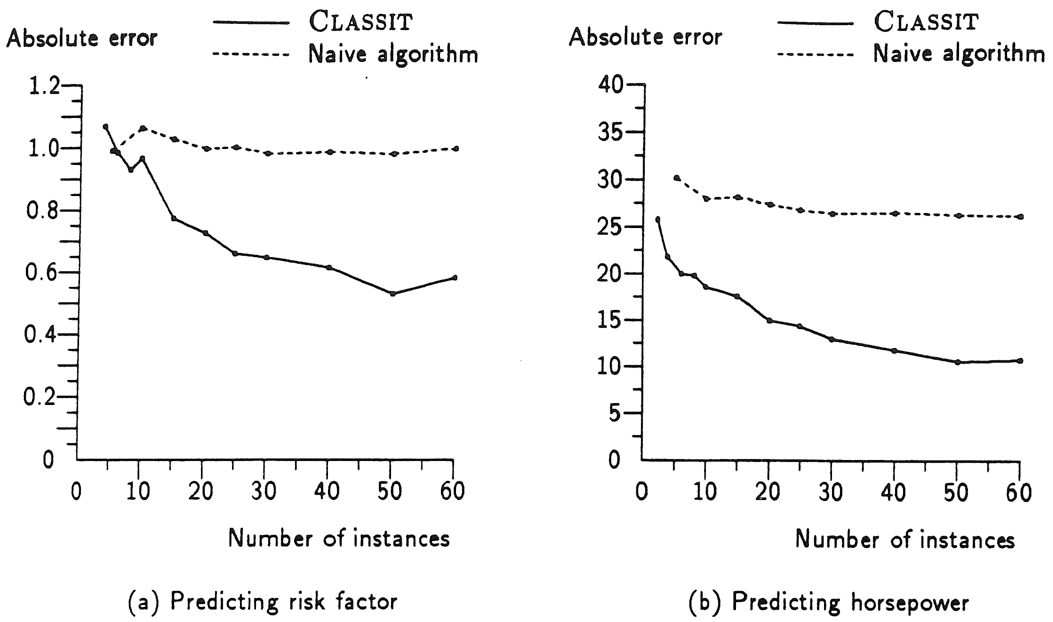


Figure 10. Learning curves for the automobile database.

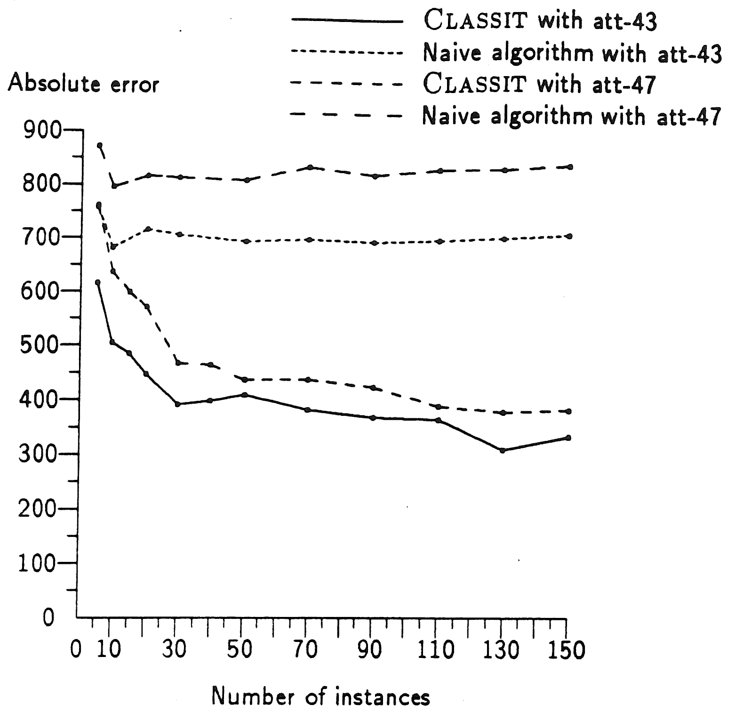
tenth instance; after this point, the concept hierarchy is fairly stable and little additional learning occurs. Also, each curve shows performance with the same type of attribute: a binary (symbolic) attribute. The simplicity of this attribute may contribute to the relatively rapid learning seen in these curves.

A different type of learning curve can be seen in Figure 10. These graphs show results from the automobile insurance database (with acuity set at 1.0 and recognition criterion at 0.9). For this experiment, the predicted attributes are numeric rather than symbolic. For these attributes I report average absolute error: the average distance from the attribute value that CLASSIT predicts to the actual attribute value in the test-set instance.

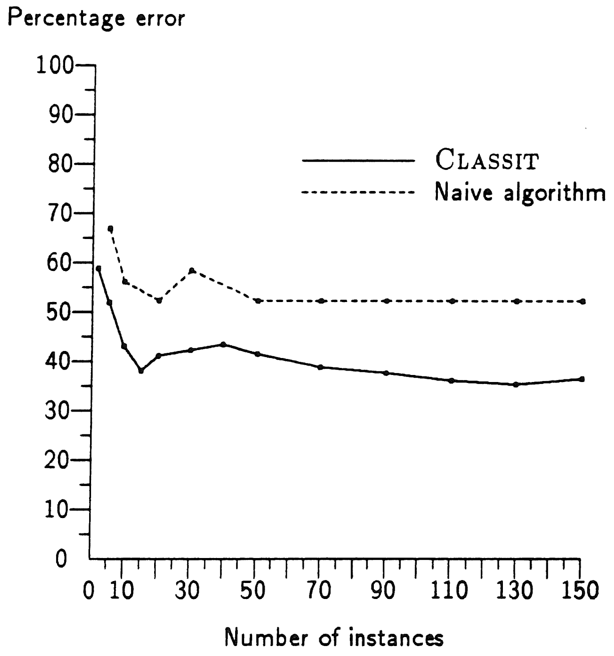
Figure 10(a) shows improved predictive ability for the 'risk factor' attribute, while Figure 10(b) shows results for the 'horsepower' attribute. The curves are displayed at different scales because the magnitude of the attributes are completely different: risk factor varies from −2 to 3 while horsepower ranges from 58 to 162. In addition to risk factor, I chose horsepower simply because there is probably some correlation between insurance risk and horsepower. There are certainly other attributes that would be equally interesting to predict (e.g., the price of the car, or the car manufacturer).

In order to evaluate these absolute errors, I compare CLASSIT's ability to that of a baseline error rate. This is shown in Figure 10 as a the learning curve for a *naive algorithm*. This algorithm simply always predicts the overall mean for the missing attribute. Although this is certainly not a lower bound for performance (it is possible to worse than average prediction), one would expect that any reasonable learning method would outperform this naive method.

Figure 11 presents learning curves for the final domain, the star LRS database. I present the learning curve for the 'class' attribute in Figure 11(b), while Figure 11(a) shows results from two randomly chosen attributes. For these experiments, acuity is set at 100.0, and the recognition



(a) Predicting attributes 43 and 47



(b) Predicting star class

Figure 11. Learning curves for the LRS Star data.

criterion at 0.7. As with the automobile database, I include the naive average prediction for comparison.

It should be noted that although the 'class attribute' is symbolic, it can take on ten possible values. This contrasts with the first three domains (see Figure 9) in which the predicted attributes were binary. One should not compare accuracy scores between these two types of attributes: it is much easier to be correct with two possible values than when choosing among ten values. This can be seen with the lower scores in Figure 11(b) for both CLASSIT and the naive algorithm.[4] Actually, the star instances are not distributed very evenly among the ten classes; three or four classes tend to dominate, with the most common class including almost 50% of the test set instances.

Although it is hard to intuitively evaluate CLASSIT's ability in this domain (especially without the aid of a domain expert), these results seem comparable to learning curves in other domains. I should point out that this domain is closest to the visual perception domain that I described in Chapter 1. Unlike diagnosis data, attribute values in the star database are simple readings from a sensor, forcing the system to learn concepts in a completely unsupervised environment. Although one might prefer something easier to visualize, CLASSIT was designed for this type of data.

2.3 Efficiency results

For all of these domains, CLASSIT shows some ability to improve predictive performance over time. I also examine the system's ability with the secondary performance task: its average retrieval efficiency. Figure 12 shows retrieval efficiency (work) versus instances for four of the real domains described in this chapter. The parameter settings, test sets, and any experimental variables are the same for this experiment as with the accuracy tests. Thus, the efficiency curve for the heart disease database was created from the same executions as used earlier for predictive accuracy.

This experiment shows that work rises approximately as the log of the number of instances. Since the depth of a tree grows at $log_b(N)$, for N nodes and branching factor b, this is exactly what should be expected when retrieving information from the concept hierarchy. It may also be possible to draw some conclusions about the different asymptotes for the different domains. According to Figure 12, CLASSIT is least efficient with the automobile domain, and most efficient with the voting database. This implies that the automobile domain requires more nodes than a domain such as the voting database, suggesting a difference in domain complexity.

Testing the efficiency and accuracy over a variety of real domains is an important step in the evaluation of a learning system. Although the domains differ along a number of dimensions, I hope to have shown that CLASSIT performs reasonably with all of them. This versatility suggests that the system should work well with a large number of potential domains.

4. A naive algorithm for symbolic attributes always predicts the most common attribute value, rather than the mean.

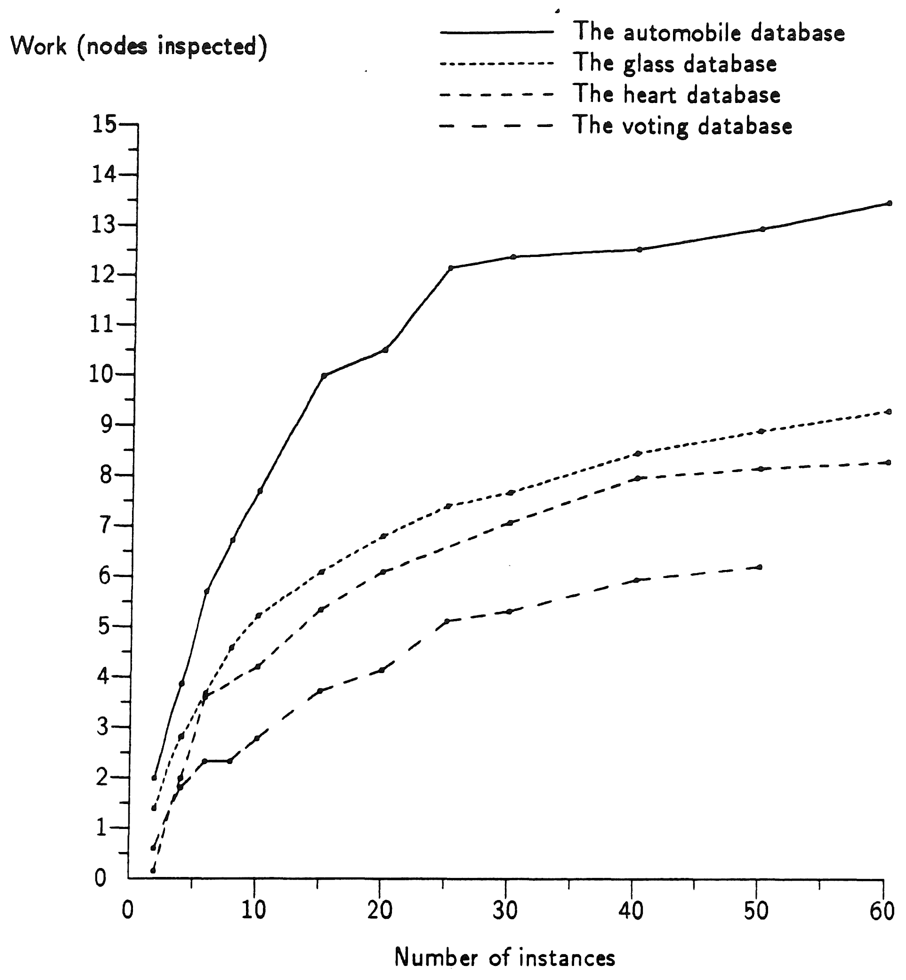


Figure 12. Retrieval efficiency for four real domains.

3. Varying domain characteristics

Although it is important to show that the system can learn with a variety of different real domains, this type of experimentation cannot guarantee that the system will not completely fail on the very next real domain. In order to understand the limitations of the system, one must characterize ways in which the input can vary, and then test the system at different points along these dimensions. To this end, I use *artificial domains* that can be systematically modified in order to demonstrate particular abilities (Langley, 1988).

The artificial domains that I use throughout this thesis are created by simple data-producing programs that incorporate a random number generator. In general, one can use these programs to precisely specify the class structure, along with the number and type of attributes for each domain. For symbolic attributes, one must specify a set of possible values, while for numeric attributes, the generator needs a mean and a standard deviation. These values can then be used with a polynomial approximation to produce values with the specified normal distribution. For completeness, all data generators (and some experimental details) are fully described in Appendix B.

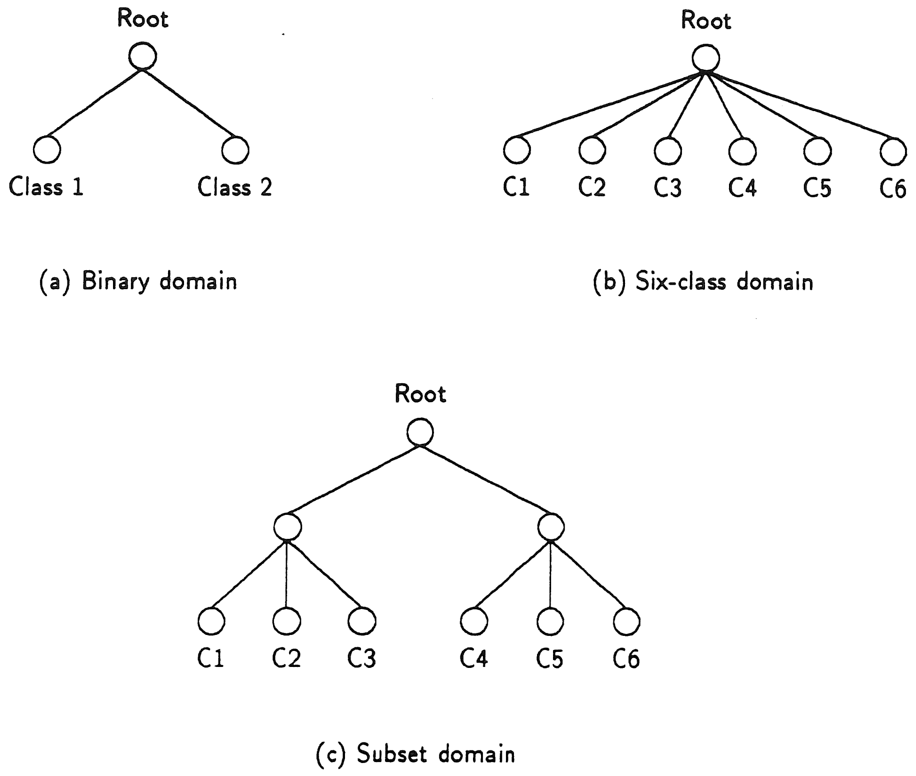


Figure 13. Three simple class structures.

In this section I use artificial domains to investigate three different domain characteristics: class complexity, noise, and irrelevant information. For each of these, I measure CLASSIT's predictive ability over a set of domains that differ along that dimension. As with experiments with real domains, all of the results in this section are averaged over ten different training-set orders. Although these experiments are preliminary, they do demonstrate that the system can be applied to a variety of different domains.

3.1 Varying the class complexity

One of the simplest tests possible with artificial domains is to verify that CLASSIT is not biased toward a particular type of class structure. One of the claimed advantages of the system is that it need not be told the number of classes *a priori*; instead, these are discovered automatically by the system. More precisely, I hypothesize that the system can reach a high level of predictive ability regardless of class structure. In order to support this claim, I show predictive performance as CLASSIT learns from three domains with different class structure.

Figure 13 shows three simple class structures: (a) shows a binary tree, (b) shows a tree with six top-level children and (c) shows a two-level tree with six subclasses. These three graphs specify the class structure of three artificial domains. The generators for these domains are described in

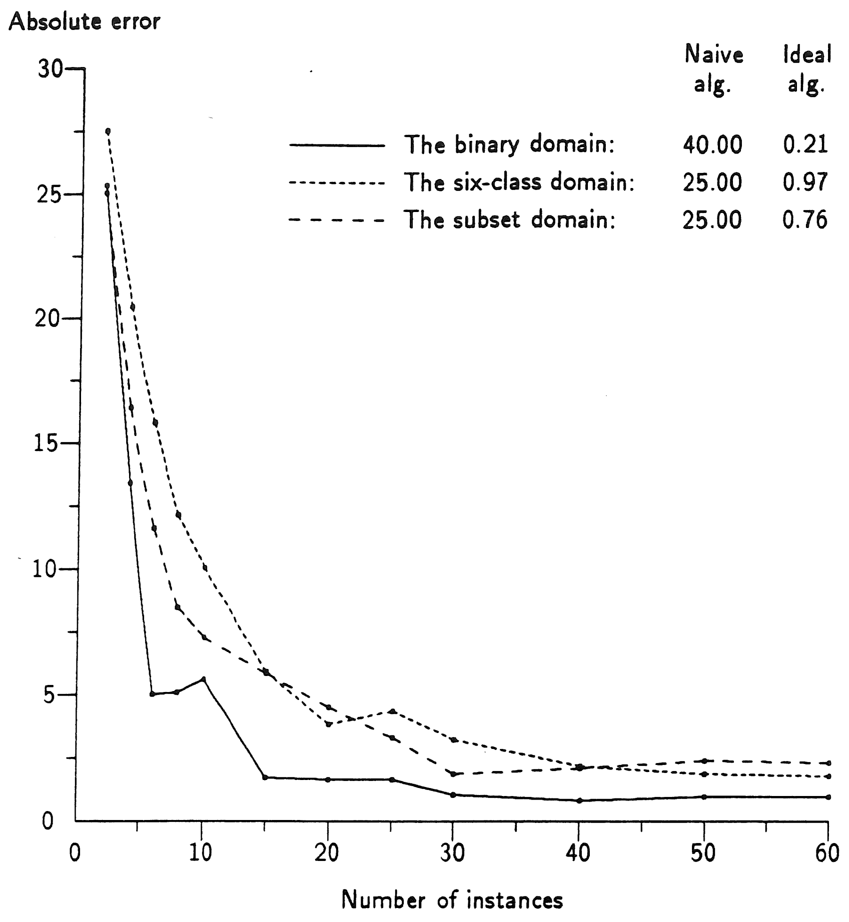


Figure 14. CLASSIT's predictive ability with varying class structure.

Appendix B. All of these domains have instances with nine numeric attributes. Predictive ability is measured with the ninth attribute, an attribute that is different for every class.

Learning curves for CLASSIT using these domains are shown in Figure 14. For all of these trials the system parameters were held constant: acuity is at 2.0 and recognition criterion at 80 percent. For each domain, I also include the predictive error for the naive algorithm. As described in the previous section, this method predicts the mean value over all instances. For this experiment, I only show asymptotic performance (after 60 instances) for the naive algorithm – for these domains, the 'learning' curves for this approach would be almost completely flat.

At the other extreme, the "ideal" error rate is also given for each of these domains. One can only determine the ideal error rate with artificial domains or when the 'best possible' hierarchy is known. This rate is determined by using the ideal hierarchy and the actual training sample of instances used during learning (in this case, 60 instances). With an infinite training set, the ideal error approaches zero; with finite sets this error indicates how far the particular sample used deviates from the norm.

In all of these experiments, I use a *noise-free* test set. This means that there is only one test set instance for each class defined by the data generator. This may seem contrary to the usual use of a test set, where the larger the test set, the better. This is the case in natural domains, because one does not know what is noise and what is 'real' class structure. By using a large test set the researcher hopes to ignore the noise in the domain by averaging. In contrast, with artificial domains, one knows the real class structure and can simply remove all noise from the test set. In this way, one can directly measure the ability of the learning system to acquire the desired 'real' class structure. Nordhausen (1989) also uses this type of noise-free test set in experiments with his IDS system.

As can be seen from Figure 14, predictive accuracy is largely unaffected by differences in class structure. In all cases, predictive error quickly approaches the ideal error. Learning is slower for the six-class domain than with a binary class structure, but this is inevitable; the system cannot make very accurate predictions until it has seen at least one instance from all six classes. I should note that predictive accuracy does not evaluate the shape of the concept hierarchy that CLASSIT produces. For example, the learning curve for the subset domain could have been duplicated by building a flat list of six classes. However, CLASSIT does build the structure with subclasses as shown in Figure 13(c). More importantly, Figure 14 does support the hypothsis that the system's performance does not depend on the class structure in a domain.

3.2 Varying the number of irrelevant attributes

In order to find the class structure for a domain, an unsupervised concept formation system must be able to discover which attributes are important and which should be ignored. This ability is important in truly unknown domains, where there may be many attributes and little knowledge about their semantics. If a system does not have this ability, then the user would be required to provide databases with only useful attributes that had something to do with the class structure. This is only one step away from supervised concept learning, where the user supplies information about concepts directly.

In contrast, I hypothesize that CLASSIT is able to determine which attributes in the domain indicate class structure. Attributes that are not correlated with any concept I will refer to as *irrelevant* attributes. More precisely, for the artificial domains that follow, an irrelevant attribute has the same mean and standard deviation for all instances.

In order to demonstrate that CLASSIT can find relevant attributes and ignore irrelevant ones, I use four artificial domains. The first domain has instances with four attributes, all of which are relevant: any of these are sufficient to predict class membership. These attributes do include some noise, but they are relatively clean. (The next section defines and discusses noise more completely.) The second domain simply adds four irrelevant attributes to the data generator. The next domain adds four more irrelevant attributes, and the final domain adds another eight such attributes, creating instances with four relevant attributes and 16 irrelevant ones. In all four domains, all attributes are numeric and the ideal class structure contains four top-level classes. See Appendix B for more description.

Figure 15 shows CLASSIT's ability with these artificial domains. For this experiment, acuity was at 1.0 and the recognition criterion at 0.8. Because the four relevant attributes have little

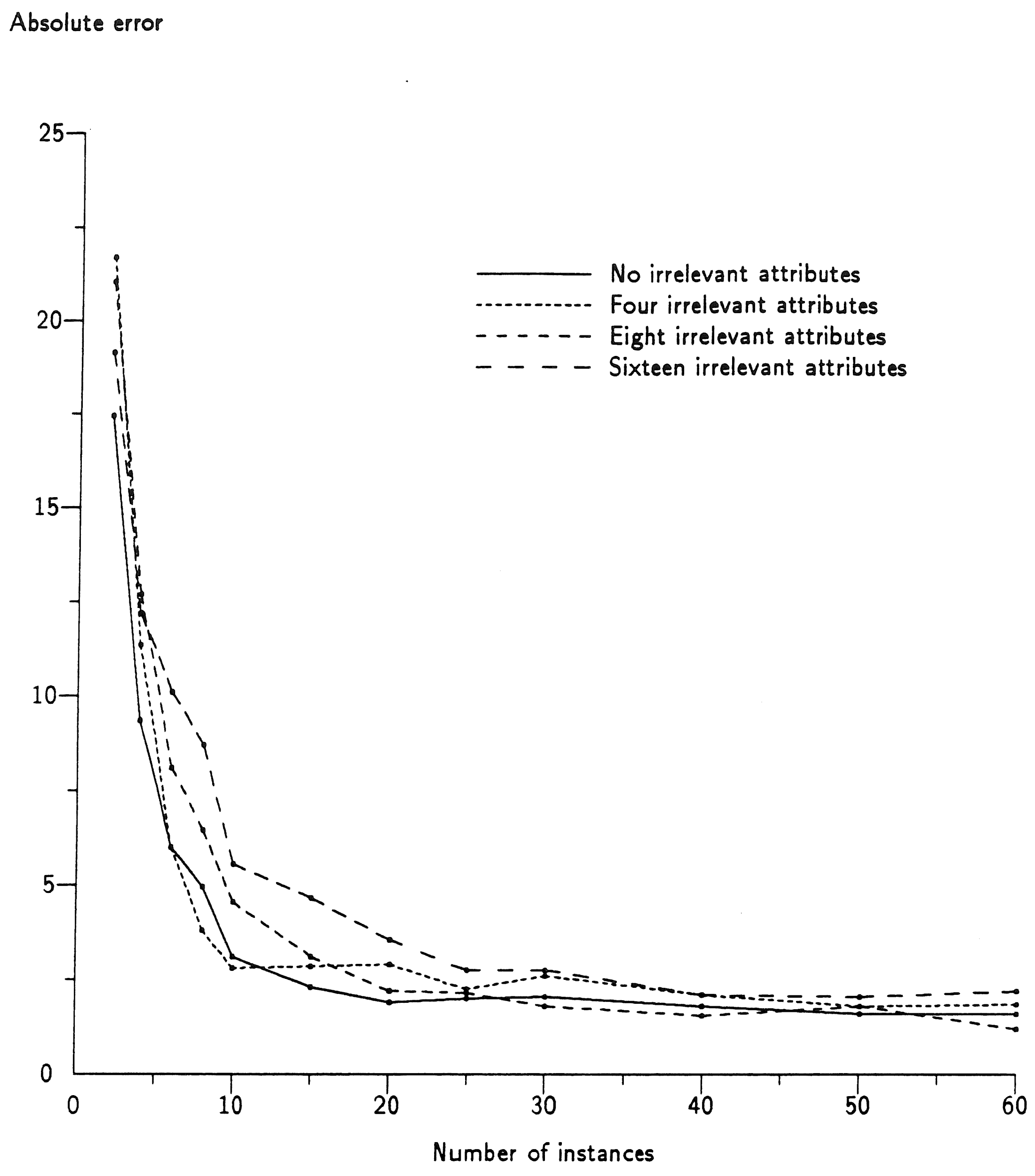


Figure 15. CLASSIT's predictive ability with irrelevant information.

noise, the learning curve quickly approaches the ideal error. For this training set of 60 instances, the ideal error is 0.47. In contrast, the naive algorithm would have an average predictive error of 20.0. As can be seen, the predictive performance of CLASSIT is largely unaffected by the number of irrelevant attributes. Although the asymptotic performance appears the same for all domains, it seems that learning is slightly slower with eight or sixteen irrelevant attributes. Presumably, this occurs because, with more irrelevant attributes, the system needs a little more time to 'find' the relevant attributes.

Although this is a very clean, simple demonstration, I expect that the ability to ignore irrelevant attributes is quite important in most real applications. For this reason, most of the artificial domains used in this thesis include irrelevant attributes. In fact, if a domain has an extremely large number of attributes, many of which are not informative, one may want to use the system to actually remove from consideration the irrelevant attributes. In Chapter 6, I expand on this idea to define a mechanism for selective attention.

3.3 Varying the noise

Because CLASSIT uses probabilistic concept descriptions, it has the capability to represent and work with noisy data. Rather than concepts that are described by logical conditions, such as "all widgets are red", CLASSIT can easily work with data where "most widgets are red (but not all)". This situation is known as a *noisy* domain. Supervised learning with noisy domains makes an additional distinction between noise in the class attribute versus noise in all other attributes (Quinlan, 1986). For unsupervised systems, there is no special class attribute, and hence noise occurs when any attribute is 'corrupted' from its expected value.

My hypothesis is that, since CLASSIT is designed for probabilistic concepts, it should perform well in the presence of noise. At the least, it should not show a sudden drop in performance with the addition of a small amount of noise. To test this, I use two noisy, artificial domains: one with symbolic attributes and one with numeric ones.

For symbolic attributes, I use a domain with four attributes and four concepts, where all the attributes are relevant and concepts are noise-free: each class can be defined as a conjunction of four attribute-value pairs. CLASSIT can trivially learn the concepts from this domain. Now, suppose one corrupts the training set instances so that some percentage of the attribute values take an alternative value.[5] Figure 16 shows results as the system learns from noise-free data and from data with 10 percent, 20 percent, and finally 30 percent noise. The recognition criterion was set at 70 percent for this experiment. (Since this is a symbolic domain, the acuity parameter is not used.)

For lower amounts of noise, there is no difference in asymptotic performance. At 30 percent noise, CLASSIT shows some difficulty, reaching only about 90 percent accuracy. Since, as usual, I use a noise-free test set, the system should be able to reach 100% accuracy even with noisy data. A possible explanation for this failure is that high levels of noise aggravate problems due to order effects. For some instance orders, the system is unable to recover from a poor decision based on some noisy instances seen early during training. Of course, this 10 percent difference in accuracy is small compared to the performance of a naive algorithm. There are four attribute values (one for each class) and since instances are drawn from the four classes with equal probability, an algorithm that simply predicts the most likely value would have an accuracy of 25 percent.

For symbolic attributes, noise simply means that the attribute takes some other random value from the set of possible values for that attribute. However, for continuous values, noise must be defined in terms of the standard deviation for that attribute. In particular, a set of numeric concepts is noisy if the concepts are defined by large standard deviations relative to each other. Noisy numeric concepts have overlapping distributions for some attributes.

For experiments with numeric noise, I use a simple domain with four classes and instances described by four attributes, one of which is irrelevant to class membership (see Appendix B for

5. Following Quinlan (1986), this noise value is chosen randomly from the range of possible attribute values such that each has equal probability.

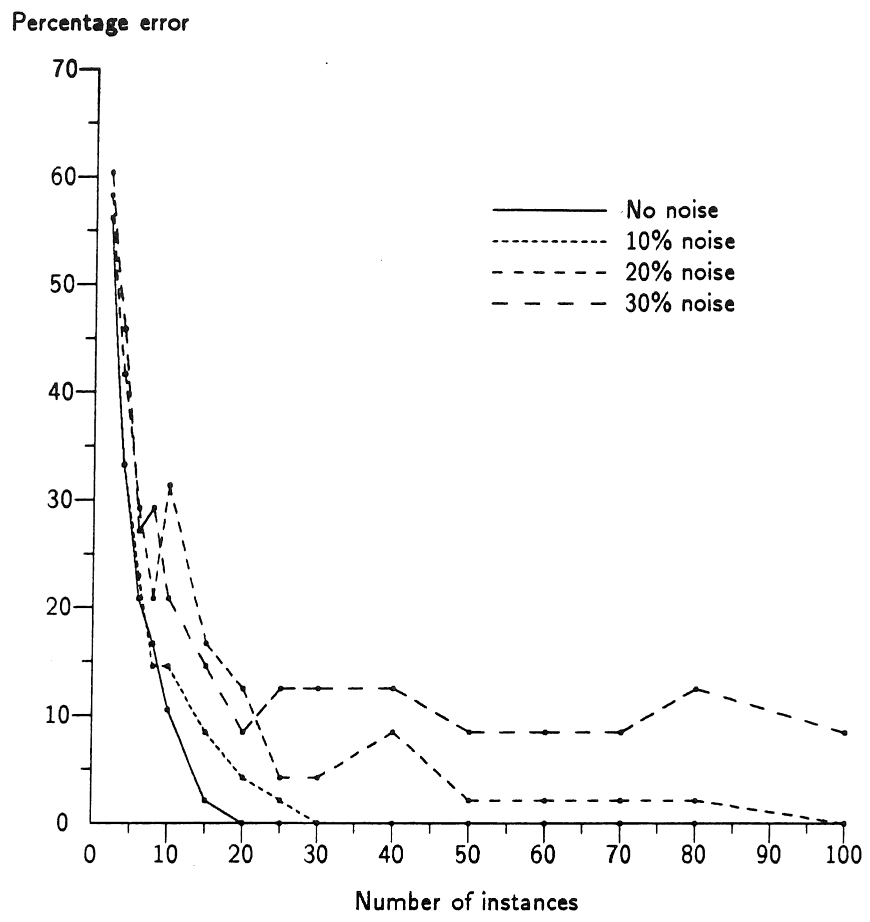


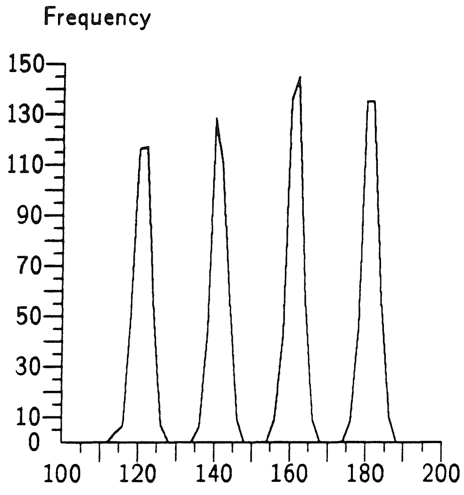
Figure 16. CLASSIT's predictive ability with noise in a symbolic domain.

details). In order to increase the noise, I increase the standard deviation for each class, while holding the mean values constant. This causes the four classes to overlap with each other, so that one cannot be certain of the class membership of an instance by inspecting its attribute values. Instead, class membership is fuzzy and best modeled by probabilities.

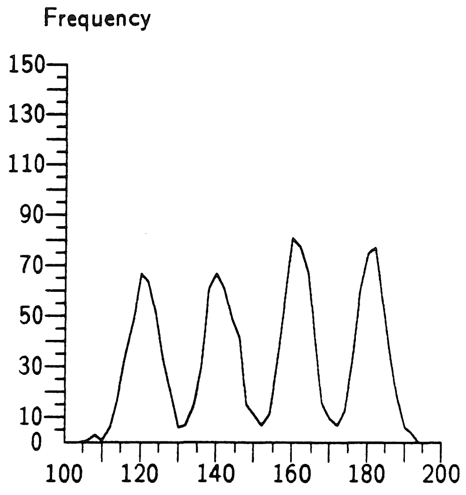
Figure 17 shows histograms for four different settings of standard deviation (σ). These graphs were created by inspecting all 1500 instances used for each noise level (10 runs of 150 instances each) and counting the frequency of the values for a particular attribute. At $\sigma = 2.0$, there is very little noise and the four classes are defined very clearly. However, these classes overlap and merge as σ is raised, and when $\sigma = 7.0$, it appears that distinguishing among the four classes is almost hopeless.

Figure 18 shows learning curves for CLASSIT at these four different noise levels. For all curves, acuity is set at 3.0 and the recognition criterion at 0.7. The figure shows predictive ability out to 70 instances; however, the curves remain much the same from the 30th instance through the 150th.[6]

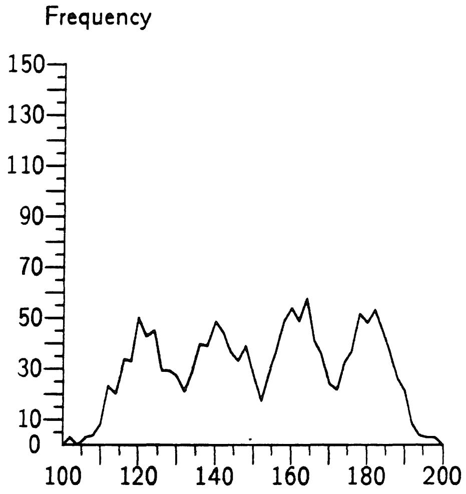
6. For the higher two noise levels, accuracy actually decreases slightly from instance 40 to instance 70. However, this trend reverses itself after the 70th instance, although the system never improves on the accuracy demonstrated after 30 instances.



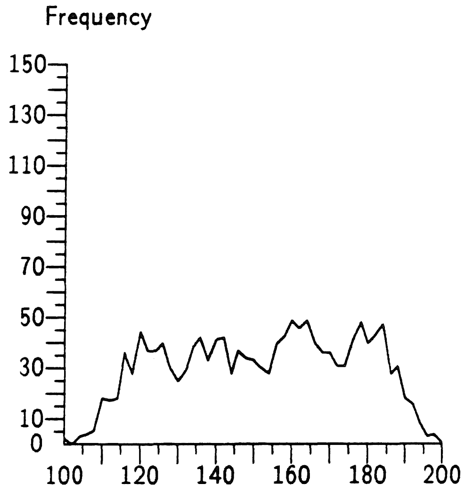
(a) Noise level $\sigma = 2.0$



(b) Noise level $\sigma = 4.0$



(c) Noise level $\sigma = 6.0$



(d) Noise level $\sigma = 7.0$

Figure 17. Histograms of numeric data with varying degrees of noise.

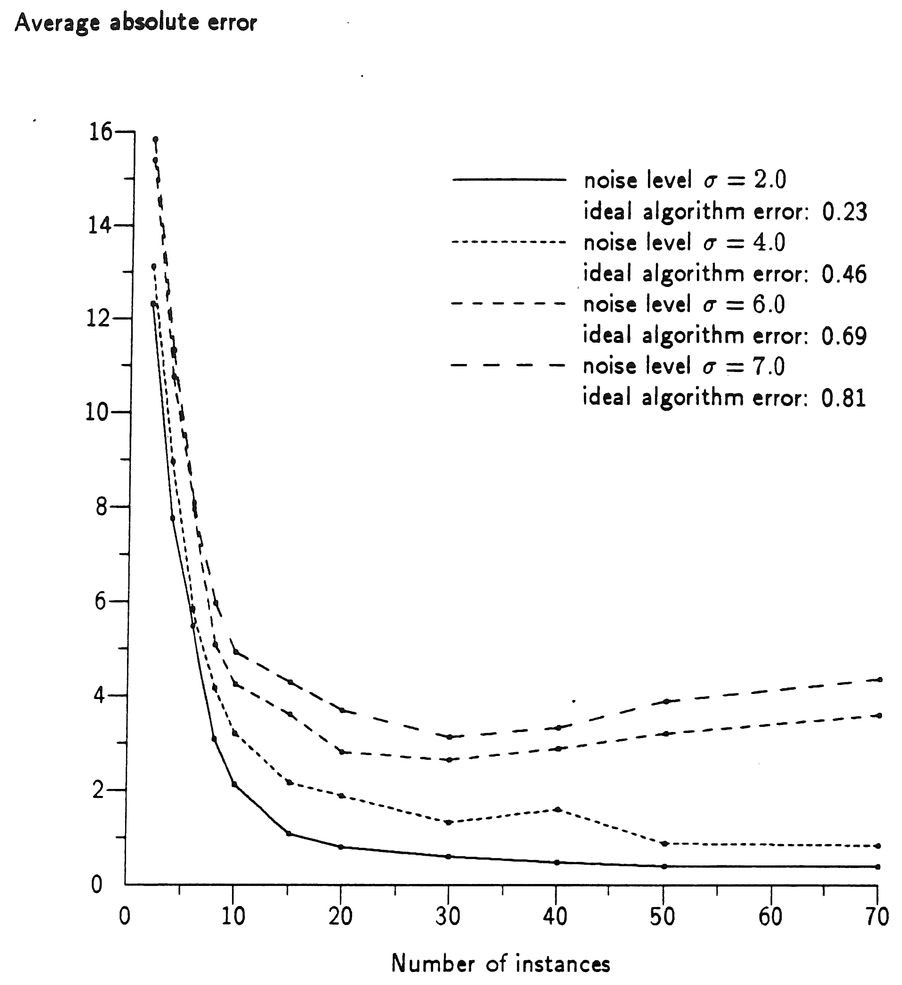


Figure 18. Learning curves for noisy numeric data.

The graph also presents ideal errors for the four domains; as before, these show the difference between the 150-instance training sets (the sample) and the ideal represented in the noise-free test sets.

With the first two noise levels (σ at 2.0 and 4.0), the system's performance is very close to the ideal error rate. With greater noise, CLASSIT shows some degradation in asymptotic accuracy. However, this drop in performance is not that great – as a comparison, the naive algorithm (for all noise levels) has an error of 20.0.

In conclusion, with either nominal or numeric data, CLASSIT shows an ability to learn with noisy data. Although asymptotic error does increase with higher levels of noise, the difference in performance is not large when compared to the error due to a naive algorithm. The ability to work in noisy domains should not be surprising: in contrast to systems that use logical concept definitions, CLASSIT's probabilistic representation is designed for precisely this type of domain. In effect, these noisy domains contain 'fuzzy' concepts, without strict boundaries. I expect that this type of domain is extremely common in real-world applications.

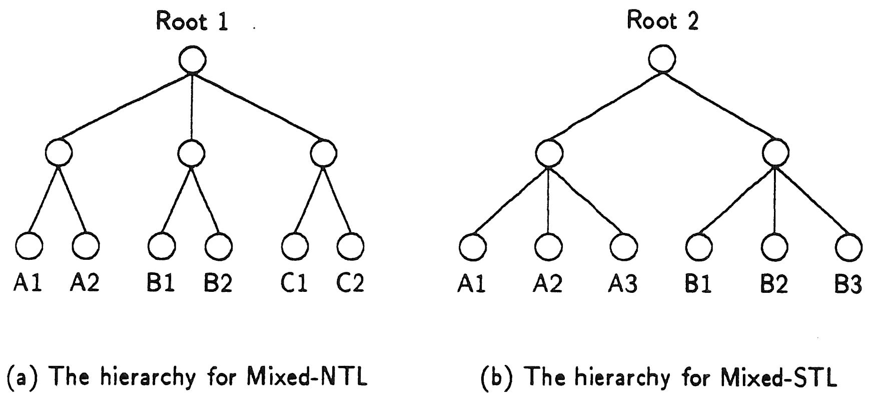

Root 1
A1
A2
B1
B2
C1
C2
(a) The hierarchy for Mixed-NTL
Root 2
A1
A2
A3
B1
B2
B3
(b) The hierarchy for Mixed-STL

Figure 19. Ideal concept hierarchies for two mixed-type data sets.

4. Mixed real and symbolic domains

One of the most unusual aspects of CLASSIT is that it allows for a mixture of symbolic and continuous attributes in the input domain. As described in Chapter 2, this has been rarely attempted in cluster analysis, and in machine learning only by a few researchers (Cheeseman et al., 1988; Reich, 1989; Anderson & Matessa, 1990). Although a number of the real domains presented earlier included both continuous and symbolic attributes, I have not yet provided strong evidence that the system can work with mixed data in general.

In Chapter 3, I showed that CLASSIT uses an evaluation function that sums $Info(C_i)$ across all attributes, i, regardless of whether i is a numeric or a symbolic attribute. However, it may be that one attribute type could unfairly outweigh the other in this summation. Although I have shown encouraging results with some mixed domains, I would like to show that CLASSIT is not unfairly biased toward one attribute type or the other.

In order to demonstrate this ability, I use an artificial domain with six numeric attributes and six symbolic attributes. Half of these attributes (three from each type) are irrelevant with respect to class structure. Additionally, I use two versions of this domain that have somewhat different class structure. In both cases the ideal concept hierarchy is two levels deep, with six subclasses at level two. In one case, the top level is best divided by the relevant three numeric attributes, while the subclasses are differentiated by the symbolic attributes. The ideal hierarchy for this domain (*Mixed-NTL*, for numeric-top-level) can be seen in Figure 19(a). For this domain, all instances in classes A1 and A2 have numeric values from the same distribution, and A1 and A2 are distinguished only by their symbolic attributes.

The second version of this domain (*Mixed-STL*) is the reverse of Mixed-NTL; its ideal hierarchy is shown in Figure 19(b). Here, the top level is best split by the symbolic attributes, while the subclasses are distinguished by the numeric attributes. Hence, classes A1, A2, and A3 all have the same symbolic attributes, but are differentiated by their numeric attributes. Details of this domain can be found in Appendix B.

The first experiment with these domains simply demonstrates that CLASSIT is able to build the appropriate hierarchies for these contrasting databases. After 120 training set instances (probably fewer could be used), with acuity at 1.0 and recognition criterion at 0.8, the concept hierarchies built by the system match the ideal hierarchies in Figure 19.[7] If the system were unfairly biased toward numeric attributes, then it would be unable to build the correct tree for Mixed-STL, building instead a tree with six top-level classes based on the numeric attributes. Similarly, if the system were biased toward symbolic attributes, it would have difficulty with Mixed-NTL. Hence, this demonstrates that Classit is not heavily biased toward one type of attribute, and that it does use both attribute types during learning.

Although this result is comforting, it is qualitative and does not test the predictive accuracy of the system. Additionally, I would like to compare Classit's hybrid approach to mixed data with other methods. As discussed in Chapter 2, researchers in cluster analysis usually do not use integrated evaluation functions, and instead use conversion techniques to coerce all attributes to the same type. One can compare the value of these techniques by comparing the performance of Classit with mixed data versus the performance of the system with converted attributes.

Attributes can be ordered by the amount of information necessary to define their scale. For example, the attribute types discussed in this thesis can be ordered in decreasing order of information as follows: continuous, ordinal, nominal, and binary. The fundamental problem with conversion techniques is that in order to convert from one attribute type to another, one must either lose information or add information to the domain. As Anderberg (1973, p. 30) states,

> Promoting a variable [attribute] implies the utilization of additional information or acceptance of a new assumption. Likewise, demotion of a variable involves relinquishing some information...one must use critical judgement as to whether the technique is appropriate to the particular problem. Indeed, informed judgement is an alternative to all these techniques and sometimes provides the only means of taking account of important but unquantifiable considerations.

In theory, the best that conversion techniques can hope for is to minimize the problems caused by adding or removing information. In practice, the choice of conversion method is often dependent on specific domain knowledge or expert 'informed judgement'.

In order to evaluate conversion techniques, I conduct three experiments. Using an artificial domain, I consider two simple-minded conversion techniques: converting attributes from continuous to nominal ones and converting attributes from nominal to ordinal.[8] Finally, with the heart-disease database, I consider the use of a more expert conversion to ordinal attributes.

For each of these experiments, I compare the predictive accuracy of Classit with two versions of the same data: one with attributes of mixed types, and one where the attributes have all been converted to a single type. Unless a conversion technique leads to significantly improved performance, one should prefer to use Classit with mixed data. This position is supported both

7. This result holds regardless of instance ordering – Classit could build the correct hierarchy for all ten orders of the training-set data.
8. Note that Classit treats ordinal and continuous attributes in the same way; in either case, μ and σ are stored rather than a list of values.

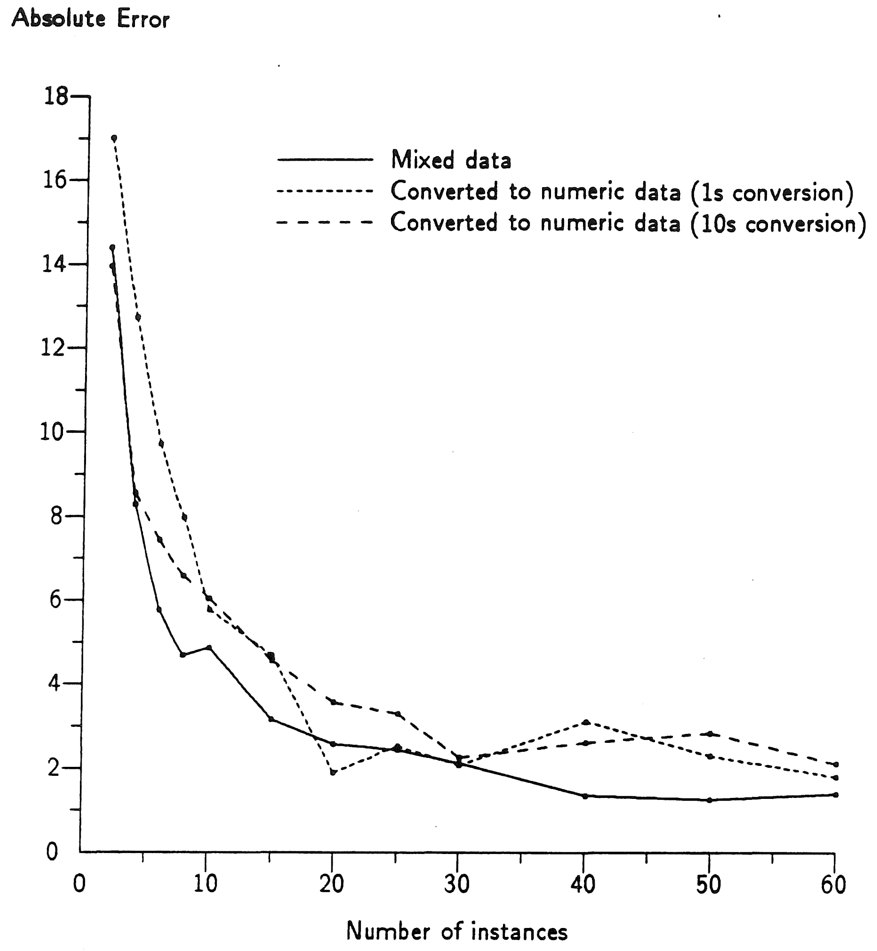


Figure 20. Predictive error when symbolic data is converted to numeric.

by the theoretical limitation of these techniques and by the effort required to choose and carry out any conversion of attributes.

The easiest way to convert symbolic attributes to numeric ones is simply to arbitrarily assign a number to every known attribute value. Figure 20 shows results when the mixed-STL domain is converted in this manner. The graph includes three learning curves. The first curve shows CLASSIT learning from data in mixed format, with six numeric and six nominal attributes. The next curve shows performance after converting the nominal attributes to numeric values in the sequence $(1, 2, 3, \ldots)$. The third curve shows a conversion to the sequence $(10, 20, 30, \ldots)$; these values are more consistent with other numeric attributes in the domain. For all curves, the predicted attribute is one of the original (unconverted) numeric attributes.

In general, conversion of the data in this manner leads to slightly worse predictive accuracy. For this domain, significantly worse performance is very unlikely: even if the six nominal attributes are eliminated or made completely irrelevant, the system should be able to cluster using only the remaining numeric attributes. However, Figure 20 does show that these conversion techniques do not lead to improved predictive performance.

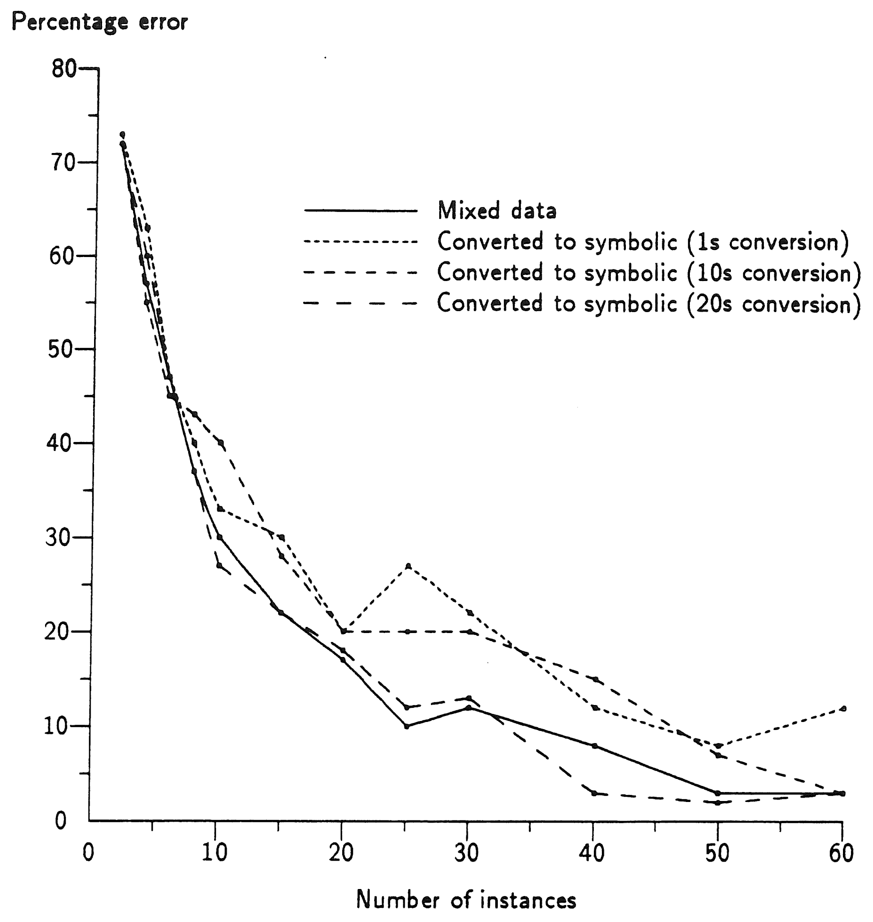


Figure 21. Predictive error when numeric data is converted to symbolic.

In order to convert a numeric attribute to a set of nominal values, one must divide the continuous scale into a set of classes, each of which is then assigned a symbolic label. In effect, this task is a one-dimensional version of the general concept formation problem. In fact, some of the more involved solutions to this problem employ approaches such as agglomerative clustering, finding cut points based on normal distributions, and other tools from cluster analysis.[9] However, it seems that these approaches are simply increasing the complexity of the task: before clustering the instances, one must first cluster along the values of each numeric attribute.

Figure 21 shows the result of applying a much simpler conversion technique for changing Mixed-NTL into a purely symbolic domain. As before, prediction is on an unconverted symbolic attribute, and the conversion technique simply changes the six numeric attributes in the original database. In order to convert these attributes, I impose cut-points that create divisions of equal size over the range. For this domain, the numeric attributes range from 3.16 to 157.15. I have tried three different sizes for the divisions: defining 8 values of size 20.0, 16 values of size 10.0, and 153 values of size 1.0 (not every possible value is present in the training set). Of these, it seems that the first choice is best, allowing CLASSIT to reach the same level of predictive ability as with mixed data.

9. For example, Lebowitz (1985) uses cut points to convert numeric data for his UNIMEM system.

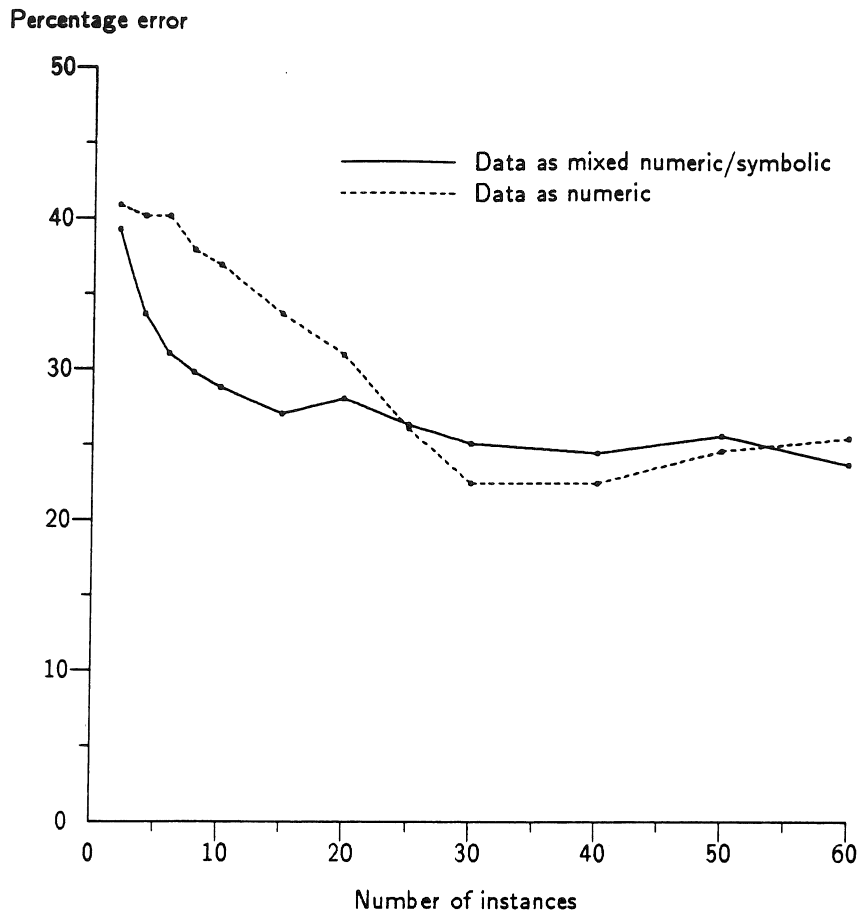


Figure 22. Predictive error when the heart disease data is converted to numeric.

However, even for this choice, there is no advantage gained by converting the data. The easiest and most accurate way to represent the data for CLASSIT is in the original, mixed format.

Both these experiments use an artificial domain and the simplest of measures. Perhaps there may be some advantage to converting attributes with real domains and with more informed experts. As mentioned in Section 2.1, the heart-disease database is also available in numeric form. The original data were converted to this numeric representation by domain experts, who presumably applied their knowledge to conversion decisions. Figure 22 shows results from this domain. As usual, the predicted attribute is symbolic with values sick or healthy. For both curves acuity is set at 1.0 and the recognition criterion at 0.8.

As with other results, this graph does not show any advantage for converting the data. If anything, it appears that CLASSIT learns more rapidly with the mixed data. In conclusion, converting attribute types as a pre-processing step to concept formation appears unnecessary for CLASSIT. Converting numeric attributes into nominal ones, or vice versa, does not improve the predictive performance of the system.

5. CLASSIT and instance ordering

As an incremental concept learner, CLASSIT is by necessity dependent on the order of instance presentation. Because an incremental system updates its knowledge after every instance, two different orderings of the same data will result in two different learning curves. If the system successfully finds the optimal hierarchy with both orderings, then both learning curves will terminate at the same level of performance. Unfortunately, a hill-climbing system like CLASSIT has the potential to get 'stuck' at a local maximum. When this occurs with some data orders, CLASSIT will never find the optimal hierarchy, and therefore does not reach the optimal asymptotic performance.

From an engineering point of view, this is a significant problem. For example, for all experiments in the thesis I have averaged results over ten different orders of the training-set instances. In this way, all experiments are likely to include both good and bad orders – orders where the system reaches optimal or near optimal performance and orders where it does not perform well. This technique makes it less likely that any result is due to a unusually good or unusually bad instance ordering.

In contrast, one can argue that from a psychological viewpoint, order effects are not such a problem. In some cases, one may prefer the system to have different results with different orders. For example, there are some well-known phenomena such as recency and primacy effects that depend on the order of instances. However, since CLASSIT is not intended as a model of human learning, I will leave this argument in support of hill-climbing approaches, and treat order effects as an engineering problem to be understood and solved.

The first thing to investigate is how often CLASSIT becomes stuck on a local optimum with respect to predictive accuracy. To test this, I used 25 different orders of an artificial data set with 100 instances.[10] Figure 23 shows both the mean and the standard deviations of the predictive error over the 25 different orders. The latter measures the difference in performance over different orders: the vertical lines indicate one standard deviation around the mean. This graph shows not only that the mean error descends, but that the standard deviation decreases as more instances are seen. Early on, there is considerable variability due to order effects, but after sufficient instances, CLASSIT usually does asymptote to the same performance level. For this domain, instance ordering has little effect on asymptotic predictive ability.

This is a surprising result: in other domains, especially with missing information and noise, the system can sometimes get stuck on a local optimum (see Section 3.3 in this chapter, and Section 1.3 in Chapter 6). This suggests that there may be a relationship between noise and order effects; perhaps the system only becomes stuck at local optimum with relatively high levels of noise.

In Chapter 3, I suggested that the *merge* and *split* operators give CLASSIT some ability to recover from local optimum. The next experiment tries to confirm this hypothesis with a lesion study. If the system is not allowed to use merge and split, its performance should be worse than when it uses these operators. In particular, I hypothesize that the decreasing standard deviations seen in

10. This study used the 'subset' domain from Section 3. The figures in this section show results up to the 60th instance, simply to allow for more detail. The curves from instance 60 to instance 100 were uninteresting: both performance and standard deviations were unchanged.

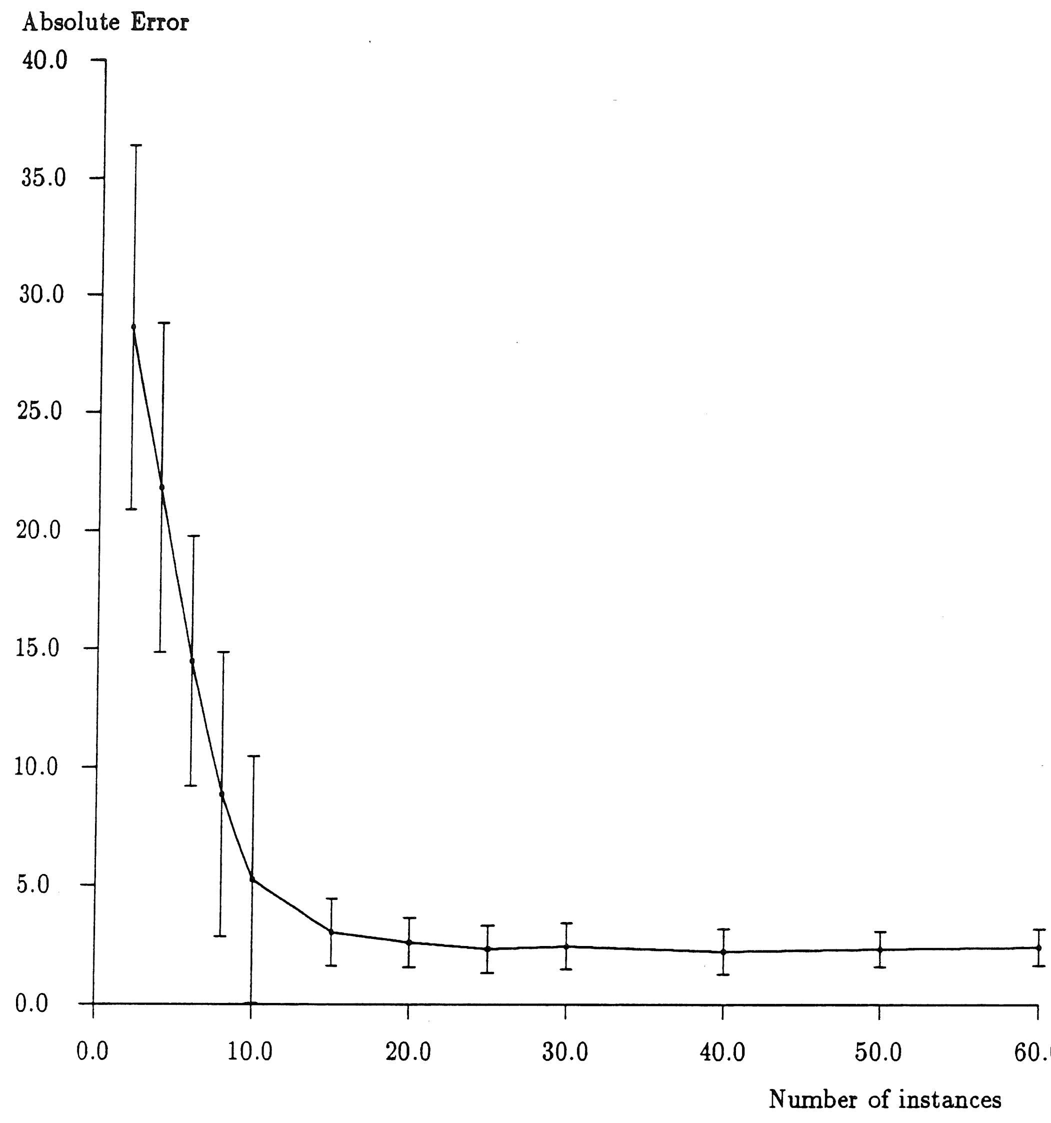


Figure 23. Mean and standard deviations of predictive ability for 25 orderings.

Figure 23 would not occur without merge and split. Using the same artificial domain as the last experiment, Figure 24 compares performance when merge and split are removed from CLASSIT's algorithm.

Surprisingly, accuracy is unaffected by the use of these operators. Not only are the learning curves nearly identical (as shown in Figure 24a), but the standard deviations also match. One possible explanation for this result is that merge and split are primarily operators on the structure of the hierarchy. That is, they do not affect the definition of a concept as much as the location of that concept in memory. This suggests that efficiency and not accuracy should be affected by these

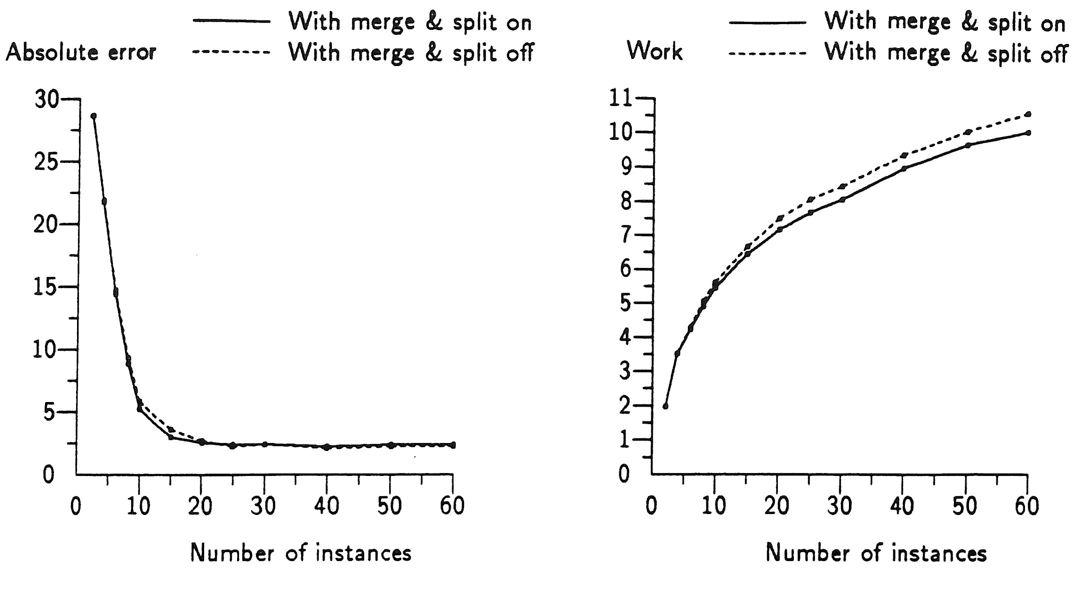


Figure 24. Performance with and without merge and split.

operators. This claim is supported by Figure 24b, which suggests that the use of merge and split increase the efficiency of the system.

However, this increase is relatively slight. In general, I also believe that CLASSIT does not select this type of re-organizing or backtracking operators frequently enough. Merge and split are applied very infrequently, and this may be a fault in the algorithm, rather than in the nature of the domain. For example, over 25 orders of this database of one hundred instances, CLASSIT performed on average 1.56 top-level merges and only 0.20 splits.

Ideally, one would prefer a system without order dependence. Since this may be impossible for CLASSIT, one would next hope that different orders only affect the shape of the learning curve, and not the asymptotic performance. For most domains, I expect that this is the case, as Figure 23 shows. Nonetheless, I do also believe that CLASSIT does not perform sufficient hierarchy reorganization. It may be that additional operators, such as Fisher's (1987b) *promote* or Wallace's (1989) *grab* operator, can alleviate this problem.

The previous experiment compared the performance of two different versions of CLASSIT. This leads directly to the next chapter, which considers a range of other variations and modifications to the system. Instead of exploring the generality of the system by trying a wide variety of input domains, I next explore the robustness and utility of the system by modifying the components of CLASSIT itself.

CHAPTER 5

Experimentation with Different Clustering Methods

There are two basic ways to evaluate a machine learning system: either one can test a single learning system over different domains, or one can compare different methods over a fixed domain. The previous chapter provided an evaluation of the CLASSIT system over a variety of domains. In contrast, this chapter presents a set of comparative studies in which the domain is held constant and the clustering method is modified.

In particular, I look at three types of modifications to the learning system. First, I study the effect of the two parameters defined for CLASSIT: acuity and the recognition criterion. Independent of any justification for the parameters of a learning system, one can empirically and analytically demonstrate the effect of those parameters on the system's performance. Not only does this let one better understand the use of the parameters, but it should also demonstrate that the system is not extremely sensitive to particular parameter settings (Iba, Wogulis & Langley, 1988; Nordhausen, 1989).

Next, I present a set of comparative studies that involve more substantial modifications of CLASSIT. As described in Chapter 2, a clustering method consists of two components: the algorithm used to cluster instances and the evaluation function that the algorithm uses to make decisions. Section 3 presents a study of evaluation functions, while Section 4 compares different algorithms. Each section presents a modified concept formation system in which either the algorithm or the evaluation function has been replaced. The ability of these alternative systems is then compared to the original version of CLASSIT.

1. Parametric studies

At one level, the effect of CLASSIT's parameters is straightforward: the recognition criterion affects the depth of the tree, while the acuity setting affects the breadth or branching factor of the tree. In particular, the recognition criterion affects how often individual instances are 'forgotten'; hence, the higher this parameter, the more instances CLASSIT saves, and the deeper (and larger) the concept hierarchy. Ultimately, with a criterion setting of 1.0, every instance is saved as a leaf node. Similarly, because acuity sets a maximum value for $1/\sigma$, this parameter affects the breadth of the concept hierarchy. Higher values of acuity increase the cost of a new disjunct, and thus bias the system to prefer narrower trees (ultimately a binary tree), while lower values for acuity create wider, 'bushier' trees.

Of course, with many domains, the shape of the learned hierarchy is dictated by the structure inherent in the data. For example, Chapter 4 showed that with a variety of input domains, CLASSIT can produce different concept hierarchies even with fixed values for acuity and the recognition criterion. In this chapter, I carry out the opposite experiments – observing performance with

Table 7. The effect of parameter settings on hierarchy depth and breadth

Recognition criterion	0.5	0.6	0.7	0.8	0.9	1.0
Average max. depth	3.8	3.9	4.1	4.5	5.1	5.9

Acuity	0.25	0.5	1.0	2.0	3.0	4.0	5.0	6.0
Average branching	5.85	4.22	3.52	3.33	3.10	2.96	2.98	2.77

different parameter settings on a fixed domain. In order to isolate the effect of the parameters, I begin by considering the situation in which the input data has no class structure.

Table 7 shows the results when CLASSIT with different parameter settings tries to build a concept hierarchy from instances that are all drawn from the same distribution. This situation can be either described as one where there are no classes, or as one with a single, very fuzzy class. In particular, the system sees 100 instances, each with three numeric attributes drawn from the same normal distribution with $\sigma = 15.0$. For a variety of acuity settings, Table 7 shows the average branching factor, and for a variety of recognition criterion settings, the average maximum depth.

1.1 Overfitting in concept formation

Of course, these results are largely qualitative and simply confirm that the parameters do affect the hierarchy as expected. A more interesting analysis considers the effect of parameter settings on predictive performance of the system. In particular, I claim that appropriate settings of acuity and recognition criterion lets the system avoid *overfitting* the data (Quinlan, 1986).

For concept formation, overfitting occurs when concepts are created that reflect noise rather than the actual class structure of the domain. As described in Chapter 4, most domains with continuous attributes are "noisy", and are therefore susceptible to this type of overfitting. For example, suppose a class of instances exists whose mean value on some attribute is $\bar{x}$. If CLASSIT creates a small subconcept, C, from this class, then the members of C may have an average attribute value considerably different than $\bar{x}$. (The size of this difference depends on the amount of noise in the domain, and on the particular members of C.) Hence, predictions based on C will have more error than those based on the more general concept that includes all instances of the class.

This type of overfitting can occur in two ways: the tree may be too deep or too wide. In either case, the system may be using a concept that is too specific for predictions, therefore leading to overfitting. Since acuity and recognition criterion affect the breadth and the depth of the tree, one would expect these parameters to affect overfitting. In the next two subsections, I report how these parameters affect predictive performance, and how they can be used to alleviate overfitting. In each section I also assess CLASSIT's dependency on the parameter, and suggest ideas for improving or eliminating the system parameter.

1.2 Overfitting and the recognition criterion

Without the use of a recognition criterion, the classification process often descends to a leaf node before making a prediction. When there is noise present, this value is not as accurate as one based on a larger class that is higher in the tree. Hence, high values for this parameter lead to to overfitting caused by a tree that is too deep. Conversely, an extremely low recognition criterion leads to extremely shallow trees. If the domain has subclass structure, this information may be lost, as the recognition criterion forces the system to truncate the tree. This leads to *underfitting* the data; predictive ability suffers because too many instances have been forced together into the same class. In summary, as I modify the recognition criterion, there should be a U-shaped curve in predictive ability; parameter values that are either too high or too low should result in lower scores than intermediate values.

In order to test this hypothesis, I use an artificial domain with two classes, each of which is divided further into two subclasses. Each instance in this domain has six numeric attributes; two of these contain information that distinguishes the major classes, two have information specific to each subclass, and two attributes are irrelevant. Additional details about this domain can be found in Appendix B. In order to demonstrate overfitting, all attributes include a medium amount of noise.[1] For this experiment I use four test set instances without noise: one for each of the four subclasses. These are presented to the system after 150 training set instances.

The results can be seen in Table 8. The error shown is an average absolute error over ten different orderings with acuity set at 3.0. CLASSIT tries to predict attribute number two; as can be seen from Appendix B, this is one of the sub-class attributes. For this domain, the best recognition criterion is 0.7; below this the subclass information is lost (underfitting), while settings greater than 0.7 lead to some degree of overfitting.

Also included in Table 8 are two boundary error scores. The "ideal" error indicates the average absolute error that would result if the learning system built the perfect hierarchy for the domain. One can define such a 'perfect' hierarchy for artificial domains because the generating function is known beforehand. For these data, the perfect hierarchy is a complete binary tree of depth two, having two major classes each with two subclasses.

This ideal value sets an upper bound on the predictive performance of the algorithm. At the other extreme, one would expect almost any learning algorithm to do as well as simply predicting the average value seen. For this domain, always predicting the mean value of attribute two gives 20.0 as the average absolute error.

The above study explores the relationship between the recognition criterion and predictive performance for a given amount of noise. A more interesting study would observe the effect of increasing

1. For this experiment, attributes were generated with a σ of 4.0; hence, the data are similar to that shown in Figure 17 (b). I will shortly examine the relationship between the amount of noise and overfitting more carefully.

Table 8. Predictive ability with different recognition criterion values

R.C. setting	0.5	0.6	0.7	0.8	0.9	1.0
No. of nodes	3.2	17.3	24.8	37.7	67.5	240.2
Max. depth	1.1	2.7	3.3	4.2	5.4	8.8
Error	10.00	7.48	0.71	1.41	1.57	2.85

Error for 'ideal' hierarchy	0.51
Error based on root node	20.00

amounts of noise on the ability of this parameter to alleviate overfitting. In particular, I hypothesize that as the noise in the domain increases, overfitting also increases, and a reasonable recognition criterion becomes more important.

In order to test this hypothesis I need an artificial domain where one can systematically increase the noise present. As described in Chapter 4, a noisy numeric domains suggests that the standard deviations of some attributes are large enough to overlap with other concepts. To experiment with noise, I use the same data as presented in Chapter 4: four classes and four numeric attributes. Since I am exploring overfitting, and not underfitting effects, I do not need a domain with subclasses. At low levels of noise (low values of σ), the classes are well-defined and one would not expect any error due to overfitting. However, at higher levels of noise, the class divisions are not as clean, and become jagged as if containing subclasses (see Figure 17, Chapter 4). In this case, the system is vulnerable to errors due to overfitting.

Figure 25 shows a three-dimensional plot of noise level versus recognition criterion versus predictive performance. Each point on the (x, y) plane corresponds to a particular noise level (σ ranges from 2.0 to 7.0) and a particular parameter setting (from 0.5 to 1.0). For each of these, the predictive error is shown in the vertical z direction. As before, acuity is held constant at 3.0.

As can be seen, for successively greater levels of noise, the magnitude of error due to overfitting increases, and it becomes necessary to set the recognition criterion to lower values. Readers should not infer from Figure 25 that CLASSIT is especially sensitive to noise: although overfitting causes some error at almost any level of noise, the highest level seen in the figure is 5.38 (when noise is at $\sigma = 7.0$ and the recognition criterion at 1.0). This value should be compared to 20.0, the error due to predicting the average value. In conclusion, this parameter has a relatively small effect on predictive accuracy when the domain is noisy, and almost no effect when the domain has little or no noise.

In addition to avoiding overfitting, the recognition criterion also slows the growth of the hierarchy. Without this parameter, the number of nodes increases at least linearly with the number of instances. When the recognition criterion is set to 1.0 (or 100 percent), then it is *off* – recognition

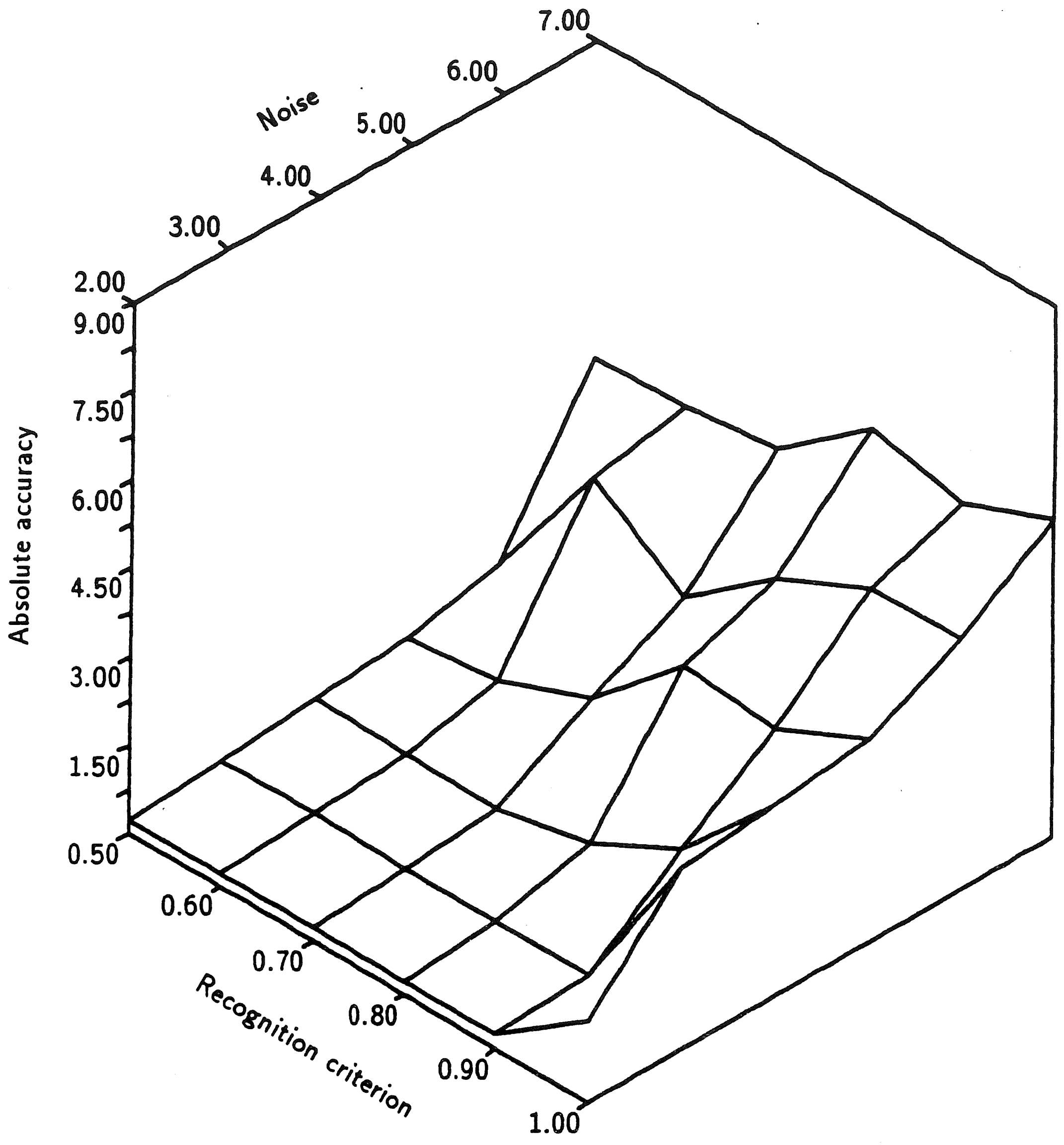


Figure 25. Overfitting as a function of noise and recognition criterion setting.

does not occur until a leaf node is reached. Thus, the difference between the last two columns of Table 8 shows a dramatic difference in hierarchy size. After 150 instances, using even a high parameter setting (90 percent) leads to a concept hierarchy of nearly one quarter the size, with an average maximum depth of 5.5 rather than 8.8.

As mentioned in Chapter 3, this parameter provides one model of forgetting. Currently, instances are 'forgotten' when the match score between instance and class is above the percentage specified by the recognition criterion. Ultimately, as I discuss in Chapter 7, one would prefer a higher-level forgetting mechanism that is based on a more comprehensive theory, which might set and modify the recognition criterion automatically.

1.3 Overfitting and acuity

Just as the recognition criterion alters the depth of the hierarchy, acuity settings have an effect on the breadth or average branching factor of the hierarchy. Just as a shallow tree can lead to overfitting, a tree that is too broad can produce the same effect. If the branching factor is greater than the number of classes present in the data, then predictions are based on much smaller subsets of instances, leading to overfitting when there is noise present.

Additionally, a poor setting for acuity can cause underfitting: when the parameter is set too high, it forces CLASSIT to build a binary tree, regardless of the class structure in the data. This causes predictive errors in two ways. First, instances of different classes may be placed in the same node in the hierarchy. If prediction is based on such a node, the data are underfit; the class is not specific enough for the data. Second, if the classification process descends far enough down the binary tree to distinguish all classes, then the data are overfit exactly as described in the previous section.

Figure 26 presents a three-dimensional plot of predictive performance versus noise levels and acuity settings. The domain for this experiment is the same as used in the previous section, with the recognition criterion set at 0.80. In general, for a given noise setting, performance follows a U-shaped curve as the acuity is raised. If acuity is too low, the hierarchy is too wide, and overfitting occurs. If acuity is too high, the hierarchy becomes a binary tree; with recognition criterion set at 0.80, this causes prediction to be based on concepts that are too general for accurate prediction (underfitting).[2]

Unlike results when the recognition criterion is modified (Figure 25), this graph does show some rather high error rates, especially with overfitting (low acuity and high noise). When acuity is low enough, CLASSIT makes so many disjuncts that instances are placed as singleton classes below the root node. When this happens with a test set instance, the only prediction the system can make is based on the root node, and as stated earlier, this node has an average error of 20.0.

Figure 26 shows that overfitting due to acuity is subject to the amount of noise present; at low noise levels there is relatively little error due to overfitting, but this error becomes more severe as noise increases. Unfortunately, one can not simply choose high settings for acuity to avoid overfitting; as we have seen, this leads to underfitting. Ideally, one would prefer the system to work well regardless of parameter settings, or for there to at least be one setting that performs well regardless of noise level. As one can see from the figure, this is not the case for CLASSIT and acuity. Although the system is more sensitive to acuity settings than I would prefer, Figure 26 shows that there is at least some flexibility possible for acuity settings. This means that for any noise level, one can get reasonable predictive accuracy without determining the ideal acuity setting.

For most acuity settings, as the noise level increases, the predictive error increases. However, for high settings of acuity, the error actually decreases initially, as underfitting is alleviated by higher levels of noise. This is because a hierarchy that is too narrow actually helps in noisy domains; it forces the system to place together objects that may not look very similar. Figure 27 shows the

2. I have also run this experiment with the recognition criterion at 1.0. As expected, this leads to somewhat better results with high levels of acuity settings. However, the error remains fairly high due to overfitting; with this recognition criterion setting, the tree is too deep.

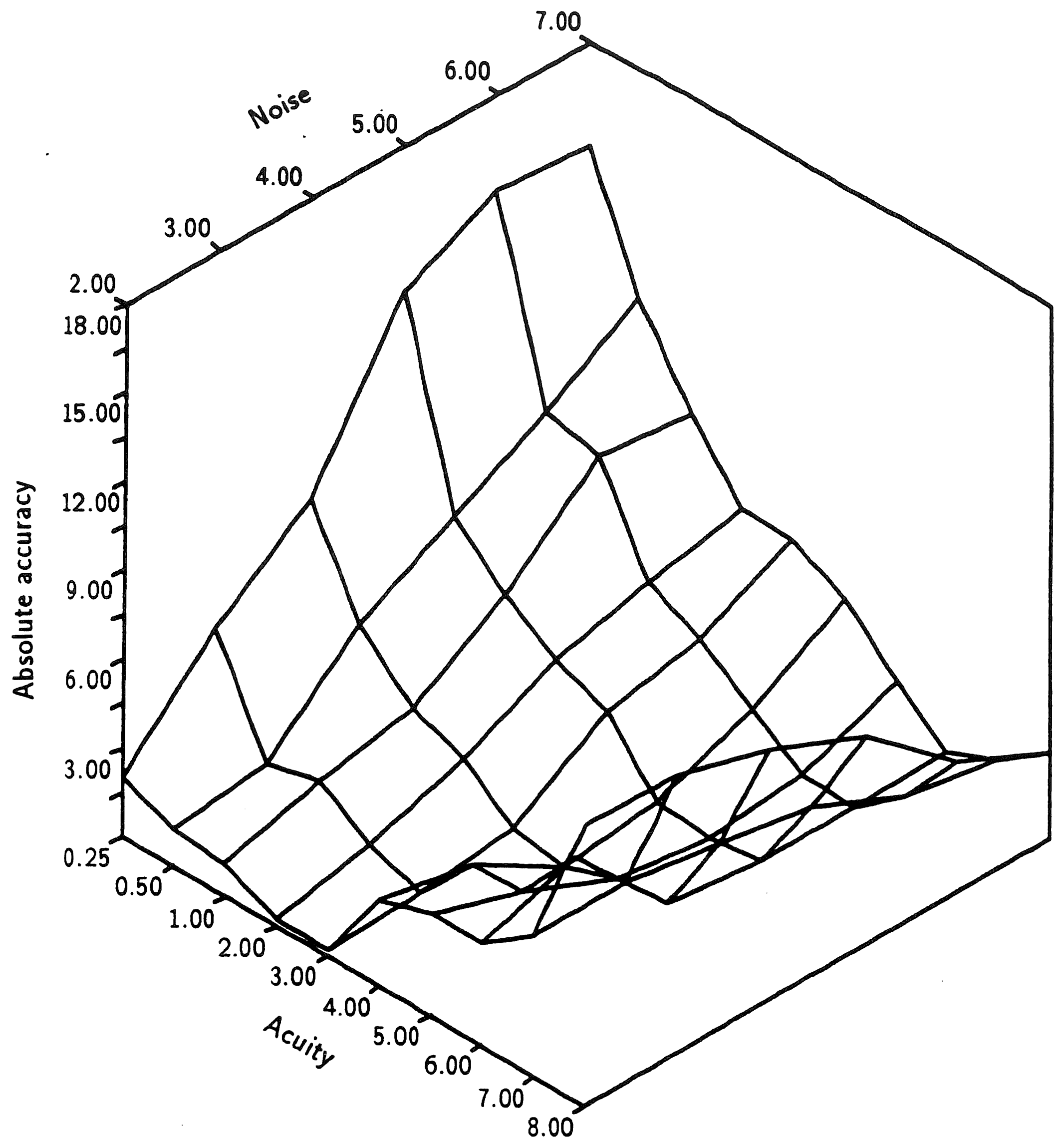


Figure 26. Performance as a function of acuity and noise.

results of this effect. This presents a plot of CLASSIT's asymptotic error rate (after 150 training instances) with acuity set at two particular levels (3.0 and 6.0) versus the performance of an ideal learner (where every instance is clustered correctly, in this case, into one four classes defined by the generator). As before, the 'naive algorithm' error is 20.0, the average error based on the root.

As described in Chapter 3, acuity is motivated by defining the smallest 'noticeable' distance for continuous-valued attributes. Hence, the value chosen should be tied to the expected range for attributes. This suggests that a more sensible way to implement the acuity parameter would be to have a different setting for every attribute. This also corresponds better to the definition of "just noticeable differences" presented in Chapter 3. As I will discuss in Chapter 7, an alternative

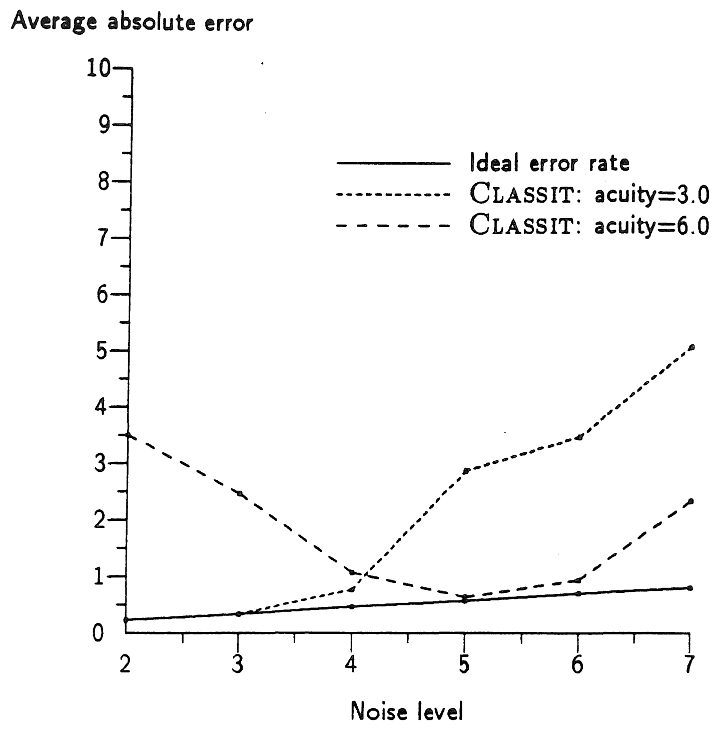


Figure 27. Best possible score versus two selected acuity settings.

approach to acuity would use such an 'acuity vector' that would allow the system to dynamically modify each of these by observing the known range of each attribute.

These studies have shown how different parameters settings affect CLASSIT's performance. They have shown that for noisy domains, the parameters can be used to avoid overfitting errors. Unfortunately, the system is more sensitive to parameter settings than one would like. However, this is only a problem with noisy domains. For example, when experimenting with artificial domains in Chapter 4, I was able to use the same parameter settings with a variety of input domains. Although the real domains required different parameter settings, modifications to these settings usually caused only small changes in performance. Finally, a better understanding of these parameters has led to some ideas for improving or eliminating them.

2. Modifying the evaluation function

One characterizing feature of any clustering system is the evaluation function used to make classification decisions. For CLASSIT, this is category utility, as described and motivated in Chapter 3. In order to study the ability of this function, and to assess the contribution it makes to the complete system, this section compares CLASSIT to similar systems that use alternative evaluation functions.

Ideally, the evaluation function should be independent from the algorithm employed by the system. In this case, one could substitute a number of different evaluation functions for category utility, assessing each in turn, until arriving at the 'best' measure. It is important to use the same algorithm for such experiments, so that the evaluation function is evaluated in isolation. In practice, there are a number of evaluation functions that cannot be used without substantially changing the algorithm, to the point where CLASSIT's basic assumptions about concept formation would be violated.

In this section I consider a variety of different measures. I begin by describing those that require a considerably different algorithm than CLASSIT's. I then turn to two functions that may be used in place of category utility. In each case, I present results from some experimental comparisons, as well a qualitative discussion of the new evaluation function's effect.

2.1 CLASSIT with other evaluation functions

In Chapter 2, I outlined a variety of similarity measures and evaluation functions that clustering systems can use. Unfortunately, many of these cannot simply be substituted into CLASSIT. For example, the system needs to evaluate the quality of a set a classes; hence, similarity measures, which compute some form of distance between two instances, are largely inapplicable. Instead, I must use an evaluation function that can compute a score from a set of classes.

Of course, there are ways to turn a similarity measure into an evaluation function. For example, the Euclidean distance between an instance x, and the mean of some class $\bar{x}_j$ is

$$Distance(x, \bar{x}_j) = \left[\sum_{k=1}^{K} (x_k - \bar{x}_{jk})^2\right]^{1/2} .$$

This is clearly related to the evaluation function $trace(\mathbf{W})$ for a set of instances:

$$trace(\mathbf{W}) = \sum_{k}^{K} \frac{1}{N_j} \sum_{i}^{N_j} (x_i - \bar{x}_{jk})^2 .$$

As described in Chapter 2, the latter measure is a sum of the diagonal elements of the co-variance matrix (or scatter matrix). Wallace (1989) uses the entire co-variance matrix when computing an evaluation measure, since this is mathematically more complete. However, this approach adds computational and storage costs of K^2 for K attributes, and I do not believe that the gain in performance is worth this cost in complexity.

CLASSIT can use $trace(\mathbf{W})$ as its evaluation function. In order to use this function, the system must store the variance for each attribute in each concept. Then, to choose among competing operators, the system would try to minimize this average variance over all concepts and attributes. Of course, the existing system does something very similar to this. For numeric attributes, category utility computes the gain in average $1/\sigma$, and then CLASSIT tries to maximize this value. Since variance is σ^2, the only significant difference between category utility and $trace(\mathbf{W})$ is that the latter subtracts the parent information: it looks at the gain in information between parent and child, rather than the absolute information (or variance) in the child. As we will see in the next section, this difference does have some effect on performance.

Since I suggest that a trace of the scatter matrix should be used instead of working with the entire co-variance matrix, perhaps I should go further and suggest that even computing the variance (or standard deviation) is too expensive. An equivalent, cheaper measure is the first moment about the mean. Just as variance is related to Euclidean distance, this measure is related to the city-block distance. In particular, for I instances, with mean of $\bar{x}$, the first moment about the mean is

$$\sum_{i=1}^{I} |x_i - \bar{x}| \quad .$$

One would not expect this measure to do better than variance,[3] but it may do almost as well and it should be considerably cheaper to compute.

However, unlike variance, this measure cannot be computed incrementally. In particular, in order to know the first moment about the mean for a class of instances, one must store the value of each member instance. Not only is this a prohibitive storage cost, but accessing these values every time that the class is updated would contradict the basic incremental nature of CLASSIT. Hence, although it appears cheaper than a measure such as *trace*($\mathbf{W}$), it is actually more expensive and cannot be readily incorporated into the system.

It is interesting that both *trace*($\mathbf{W}$) and the first moment about the mean give the best score to a hierarchy where every instance is in its own class. As discussed in Chapter 3, this is also true for category utility with numeric domains. This occurs because because all of these functions are based on some sort of distance, which is always zero for singleton classes. Previous clustering systems have avoided this issue by either building only binary trees (such as agglomerative methods), or by employing a user-supplied parameter k that defines the number of classes *a priori* (as in most iterative optimization methods). As described earlier, CLASSIT avoids this problem with the acuity parameter. In addition to corresponding to the just noticeable differences for continuous attributes, acuity sets an upper bound on the score for a new singleton class. This type of maximum score (or minimum distance) parameter can be used with any of these evaluation functions.

There is probably no limit to the number of different evaluation functions one could try, especially if some creativity is used to adapt the function to CLASSIT's algorithm. For example, it may be possible to build an evaluation function based on some of the similarity measures for symbolic attributes presented in Chapter 2. For this study, I limit my comparisons to evaluation functions for numeric attributes, and to those that can be computed without great cost. The next two sections present comparative studies between category utility and two other evaluation functions: a variation on *trace*($\mathbf{W}$) and a function that measures the gain ratio from parent to child.

2.2 Category value versus category utility

The first alternative evaluation function I consider is a simple variation on category utility. Instead of measuring the information gain between parent and child levels, this function simply measures the absolute information of the child level. As defined in Chapter 3, category utility for K classes and I attributes is

3. The only case where city-block distance may be more appropriate than Euclidean distance occurs when the attributes are interval rather than continuous or ratio scaled.

$$\frac{\sum_{i}^{I}\sum_{k}^{K} P(C_k)Info(C_{ik}) \quad - \quad Info(C_{ip})}{I \cdot K} \quad . \tag{1}$$

With this equation, the system sums the value of every child concept, C_k, and subtracts the information at the parent, C_p. Hence, this measures the *gain* in $Info(C)$ from parent to child levels of the hierarchy. Rather than using the gain in concept value, one can define an evaluation function simply with the average value of the child concepts. Thus, I define *category value* as maximizing the average $Info(C)$ without subtracting the parent information:

$$Category\ Value \quad = \quad \frac{\sum_{i}^{I}\sum_{k}^{K} P(C_k)Info(C_{ik})}{I \cdot K} \quad . \tag{2}$$

This evaluation function is of interest for two reasons. First, it can be used to carry out a simple lesion study for category utility. By deleting the parent term from the evaluation function and comparing performance, one can empirically observe the effects of that term. Since I expect that the parent term is useful, I hypothesize that when category value is used, performance will drop in some way. Second, some simple analysis shows that this function is closely related to $trace(\mathbf{W})$.

When comparing evaluation functions, it is possible that two functions may give different magnitude scores yet still make the same clustering decisions given the same set of instances. More formally, I define two evaluation functions $f(x)$ and $g(x)$ to be *isotonic* if,

$$\forall(x,y), f(x) < f(y) \iff g(x) < g(y) \quad ,$$

where x and y are competing partitions of instances into classes. An intuitive example of this relationship is between $sin(x)$ and $2sin(x)$; these functions have maximums and minimums at the same places and are thus isotonic.

Given this definition, we can see that for numeric attributes, category value is isotonic to $trace(\mathbf{W})/K$ (where K is the number of classes). In particular, $P(C_k)$ is identical to $1/N_j$, and the difference between variance and standard deviation does not affect the maximums or minimums of the two equations. This means that the comparison between category value and category utility is also a comparison between an averaged $trace(\mathbf{W})$ score and category utility.

Surprisingly, category value is also isotonic to category utility *if* the number of classes, K, is fixed. This means that for a static concept hierarchy, these two evaluation functions will classify an instance the same way. However, whenever the system considers merging, splitting, or creating a new disjunct, the evaluation functions perform differently. In particular, category value prefers fewer children than category utility.

For example, suppose that both evaluation functions are considering a split operation that increases the number of concepts in the partition from three to six. Suppose that category value prefers not to perform the split, keeping the number of children at three. In contrast, category utility subtracts the parent term from these scores: $Info(C_p)/3$ for when there are three concepts and $Info(C_p)/6$ for the split. Since the latter value is smaller, this subtraction may be enough to make category utility choose in favor of the split. This demonstrates that in some cases, category value prefers narrower trees than does category utility.

If one considers that category value is isotonic to $trace(\mathbf{W})$, then it is not so surprising that category value and category utility differ over how many children to create. Research in cluster analysis has never successfully addressed how many classes to create during concept learning (see Everitt, 1979); this number is either supplied, or if an agglomerative algorithm is used, there are always two children per level. The parent term $(Info(C_p))$ in category utility is needed to help discover the right branching factor for the hierarchy.

Although this analysis says something about the shape of the hierarchy produced by one evaluation function versus another, nothing can be concluded about the predictive ability of these alternative hierarchies. In fact, as I show next, it is quite possible for very different hierarchy structures to have the same predictive accuracy.

Figure 28 shows the learning curves for predictive performance of CLASSIT with category utility (henceforth, $Classit_{CU}$) versus CLASSIT with category value $(Classit_{CV})$. This figure shows results with two domains: the heart-disease database and the glass forensic database. Both domains have mixed numeric/symbolic instances, and in both cases the task is to predict a binary symbolic attribute: sick or healthy for the disease database, and window glass or not window glass for the forensic database.[4]

Although there is some difference in performance, using category value does not seem to change either the learning rate or the asymptotic predictive accuracy to a large degree. If anything, $\text{Classit}_{\text{CV}}$ performs slightly better. This empirical demonstration certainly does not prove that the evaluation functions are equivalent with respect to predictive accuracy.[5] However, to date, I have not found any real domains for which the alternative clustering systems have significantly different predictive ability.

At first blush this was a very surprising result, suggesting that category utility has an unnecessary term: the parent information, $Info(C_p)$, in equation (1). However, this is not a fair conclusion. Figure 28 only shows predictive accuracy, and this does not evaluate the structure of the concept hierarchies. As I suggested above, I expect that the hierarchies built by $\text{Classit}_{\text{CV}}$ and $\text{Classit}_{\text{CU}}$ are different, even though their predictive ability is about the same.

In Chapter 4, I described two general performance measures for a concept learning system: accuracy and efficiency. The former primarily evaluates the quality of the learned concepts, while the latter evaluates the quality of the hierarchy used to organize and retrieve concepts. Since I hypothesized earlier that $\text{Classit}_{\text{CV}}$ and $\text{Classit}_{\text{CU}}$ would build different hierarchies, I must look at average retrieval efficiency, a measure of hierarchy performance, in addition to predictive accuracy. Finally, in order to carefully control the complexity of the domain, I should use artificial data. I also hypothesize that since $\text{Classit}_{\text{CV}}$ prefers narrow hierarchies, the wider the class structure in the data, the more problems this system may encounter.

4. Chapter 4 describes these domains more completely. As before, acuity is set at 3.0 and recognition criterion at 0.7 for the heart-disease domain, while acuity is 0.1 and recognition criterion is 0.8 for the glass database.
5. In fact, one can construct an artificial domain where different concept hierarchies produced by the different evaluation functions have different predictive abilities.

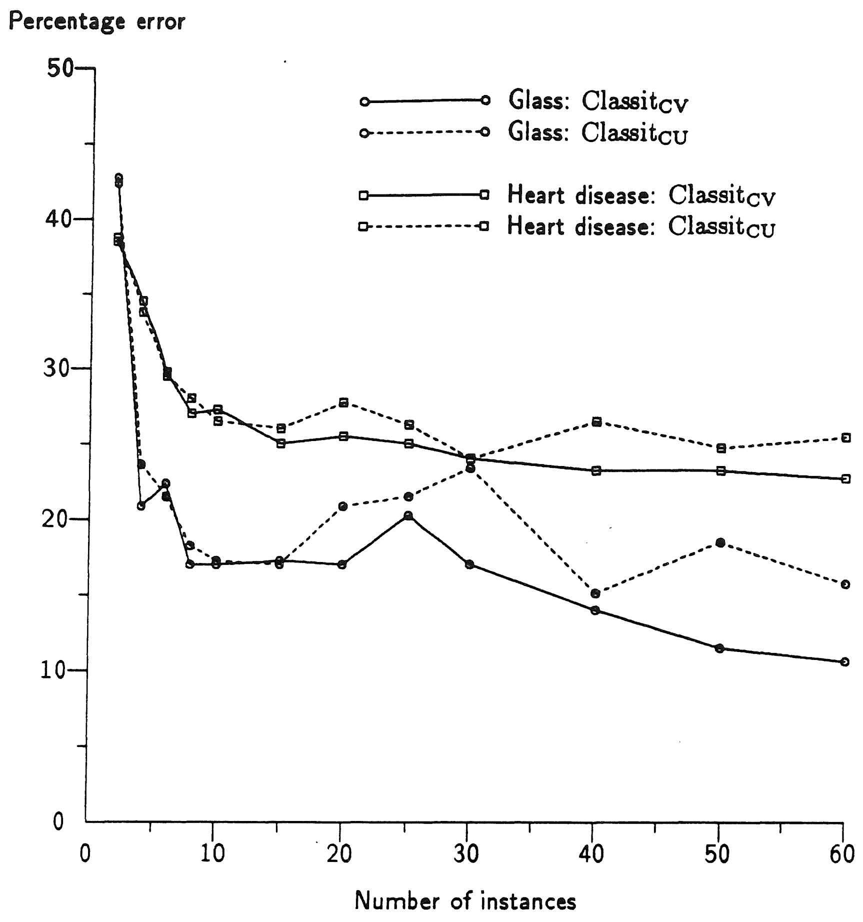


Figure 28. Category value versus category utility on two real domains.

Table 9 shows results from two simple variations on the artificial database described in Section 1.1. Each domain has a noise level of 5.0 (see Figure 17), acuity is set at 3.0, and the recognition criterion at 0.80. Results are shown after Classit$_{CV}$ and Classit$_{CU}$ have learned from 150 instances. I also include performance for an 'ideal' learner and a 'naive algorithm' method. Both of these are as defined in Section 1.2: the ideal learner is created from the domain generator, and the naive method makes predictions based on the average value.

Table 9 shows both predictive accuracy and retrieval efficiency for these two domains. Efficiency is defined as in Chapter 4: the average number of nodes inspected during classification of test set instances. Once again, this experiment produced some surprising results: category utility and category value produce hierarchies with roughly the same efficiency. Although Classit$_{CV}$ builds deeper hierarchies, which decrease retrieval efficiency, it also builds narrower hierarchies than Classit$_{CU}$, which increase retrieval efficiency. However, my original hypothesis that the structure of the hierarchies are different still holds true.

One way to demonstrate the difference between these hierarchies is to use *top-level prediction.* This is a simple variation of the prediction task. Instead of allowing Classit to descend through the hierarchy before making a prediction, classification is halted after the first level in the hierarchy.

Table 9. Classit$_{CV}$ versus Classit$_{CU}$ on artificial domains.

	Predictive error	Top level prediction	Average depth	Retrieval efficiency
Cat. Utility	2.15	0.54	1.23	6.90
Cat. Value	2.05	6.81	2.40	6.87
Ideal Learner	0.51	0.51	1.00	3.00

(a) Three-class domain. Naive method error: 13.33

	Predictive error	Top level prediction	Average depth	Retrieval efficiency
Cat. Utility	2.08	2.41	1.35	9.65
Cat. Value	1.99	23.28	3.77	9.53
Ideal Learner	0.68	0.68	1.00	6.00

(b) Six-class domain. Naive method error: 30.00

Not surprisingly, Classit$_{CV}$ does very poorly with this task, since the top level rarely corresponds to the actual classes in the data. As expected, this problem is more serious for the six-class data, as in Table 9(b), than when there are only three classes present, as in Table 9(a). In contrast, Classit$_{CU}$ does as well (or better) with top-level prediction than with the usual prediction task.

In summary, both the predictive power and the retrieval efficiency of CLASSIT with category value is equal to that of the system with category utility. However, the hierarchy produced is quite different. This may be important to some applications of concept formation. For example, Iba and Gennari (in press) describe an application in which the top level of the concept hierarchy is used to define a motor schema. Each node at the top level corresponds to a state in the motor schema. If the top level of the hierarchy does not reflect the class structure of the data (as with Classit$_{CV}$), then the resulting motor schema will not represent a correct movement. For this application, category value would be a poor choice for an evaluation function.

2.3 Gain ratio versus category utility

For a second comparison of evaluation functions, I look at another simple modification to category utility. In contrast to category value, this function is not a simplification of category utility, but rather a modification that is designed to make the function more robust in the face of different amounts of information across attributes.

In particular, I define the *gain ratio* function as

$$GainRatio \;=\; \frac{\sum_i^I \sum_k^K P(C_k)Info(C_{ik}) \;-\; Info(C_{ip})}{I \cdot K \cdot Info(C_{ip})} \quad . \tag{3}$$

The only difference between this function and category utility (equation 1) is that the information gain is normalized by $Info(C_{ip})$. Instead of using the absolute gain in information from parent to child, this evaluation function measures the percentage change in $Info(C)$: the information gain over the information already present.[6]

This modification is motivated by the concern that different attributes may have $Info(C_i)$ scores that are very different in magnitude. This is clearly the case for continuous attributes, since the magnitude of $1/\sigma_i$ depends on the magnitude of the values for the attribute i. This can lead to a problem if the 'important' attributes are somewhat noisy and have low $Info(C)$ scores: for category utility, these attributes may be overwhelmed by some 'unimportant' attribute that has $Info(C)$ scores of a higher magnitude. This problem should be alleviated by the gain ratio evaluation function.

In order to clarify the intuition behind this function, consider an example in which the system is learning from a sequence of flower descriptions. It may be that the attributes associated with species (such as leaf type or petal size), have relatively low scores when compared to an 'irrelevant' attribute such as color (irrelevant at least for distinguishing species). Hence, Classit$_{\text{CU}}$ may learn classes based on colors: grouping all red flowers together, all yellow flowers together, etc. In contrast, Classit$_{\text{GR}}$ (CLASSIT with the gain ratio evaluation function) should notice that, although the species attributes have low scores, they have a large percentage gain between levels in the hierarchy, and hence should outweigh the color attribute. Thus, Classit$_{\text{GR}}$ should build the preferred hierarchy, dividing the instances into roses, carnations, irises, and the like.

Although the motivation behind gain ratio appears reasonable in some cases, it can cause problems in other situations. As I showed above, the function can lead to useful hierarchies if the 'important' attributes are noisy, and have low scores compared other 'unimportant' attributes. If this is not the case, then Classit$_{\text{GR}}$ may perform poorly. Gain ratio emphasizes attributes with less information, and decreases the weight of attributes with more information (those with higher $Info(C)$ scores). If the former, low-score attributes really are 'noise' with respect to the classes present, then the system will have more difficulty finding that class structure. In this respect, gain ratio is similar to an attribute-weighting scheme, such as discussed in Chapter 2. If the assumptions underlying the weights are correct, then the scheme will work well. If the assumptions are incorrect, the weights may actually hinder the system's ability to learn.

Figures 29 and 30 compare the predictive performance of Classit$_{\text{GR}}$ to Classit$_{\text{CU}}$ with some real-world domains. Figure 29 shows results with the glass database, where Classit$_{\text{GR}}$ performs about as well as Classit$_{\text{CU}}$, and with the heart-disease database. In the latter case, Classit$_{\text{GR}}$ performs significantly worse, suggesting that this domain is not suited for the gain ratio evaluation function.

6. This function is very similar to the gain ratio criterion defined by Quinlan (1986); the most significant differences involve the definition of $Info(C)$.

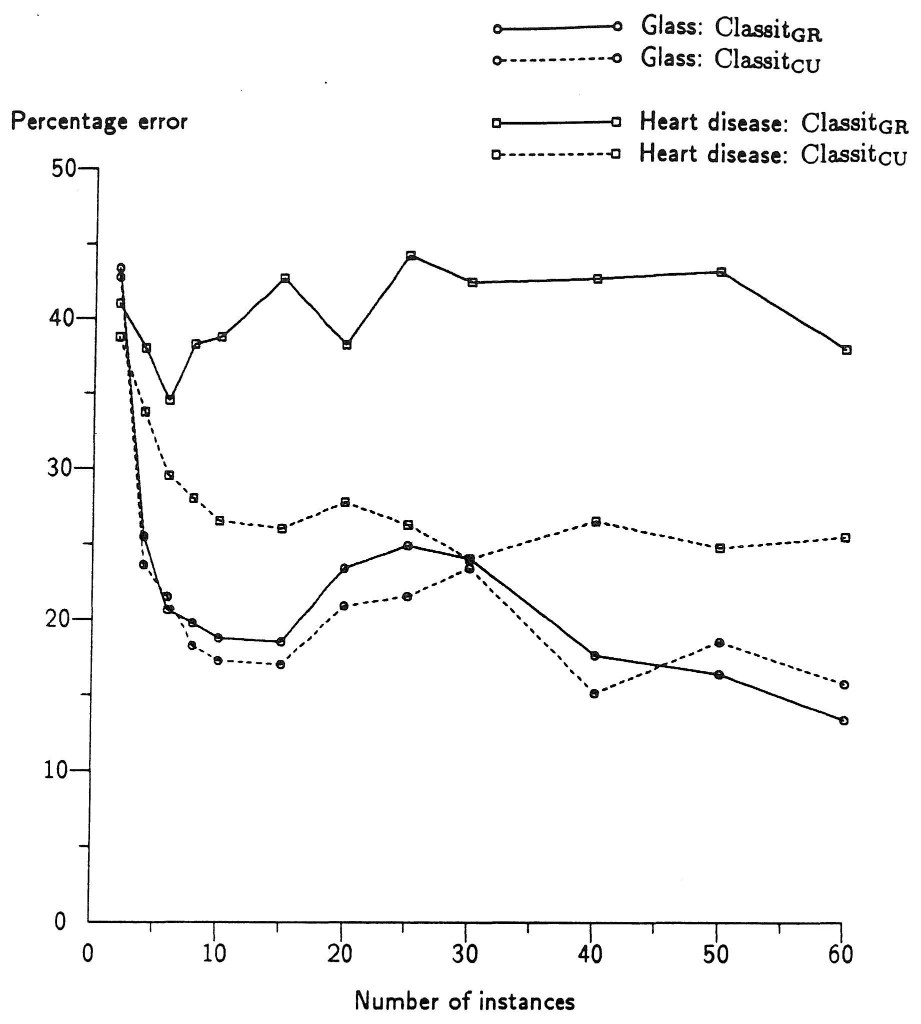


Figure 29. $\text{Classit}_{\text{GR}}$ with heart disease and glass databases.

In Figure 30, $\text{Classit}_{\text{GR}}$ also performs poorly. This result is from the automobile insurance database: acuity is 1.0, the recognition criterion is 0.9, and the predicted attribute is **engine-horsepower**, a continuous attribute. [7] Also shown is the naive algorithm error, when the overall average is used to predict the attribute. As the graph shows, even this rather simple-minded attribute prediction method outperforms $\text{Classit}_{\text{GR}}$.

These results show that the gain ratio evaluation function has some problems, especially in real-world, noisy domains. In at least one domain, $\text{Classit}_{\text{GR}}$ performs as well as $\text{Classit}_{\text{CU}}$ on predictive accuracy. However, evidence from real domains does not identify the conditions under which $\text{Classit}_{\text{GR}}$ is accurate or inaccurate. In order to control the learning environment, I turn once again to artificial domains.

In particular, I use a domain where the assumptions behind the gain ratio evaluation function are correct: the absolute $Info(C)$ scores should be normalized so that higher magnitude values are not given too much weight. Although this domain is purely imaginary, it can be understood

7. I have found similar results to these with attribute **risk-symbol.**

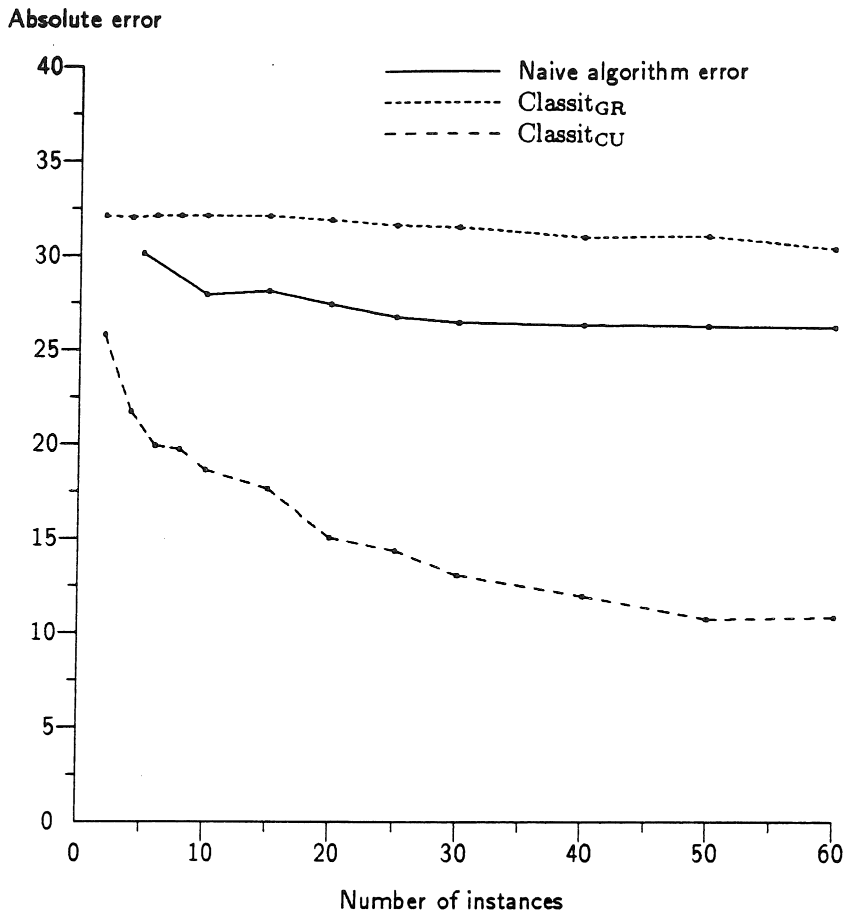


Figure 30. Classit$_{GR}$ with the automobile insurance database.

by analogy with the flower example presented earlier. The data generator, which is detailed in Appendix B, creates eight subclasses of instances corresponding to eight possible flowers. There are eight attributes per instance, and these are arranged so that four attributes suggest a hierarchy based on 'color', while the four noisier attributes suggest a hierarchy based on 'species'.

Figure 31 depicts these competing hierarchies, with the leaf nodes in the center, labeled A1, A2, etc., in correspondence with the labels used in the data generator. From these leaf nodes a tree extends upward, corresponding to the hierarchy Classit$_{CU}$ builds where instances are first divided by the less noisy 'color' attributes. Extending downward is an alternative hierarchy, corresponding to the hierarchy that Classit$_{GR}$ builds where the instances are divided by their 'species' attributes.

Although I can predict that Classit$_{GR}$'s hierarchy will have a different structure than Classit$_{CU}$'s hierarchy, the effect on predictive accuracy is less clear. In fact, as we will see, once the two systems have seen sufficient instances, they have the same predictive accuracy. Their differing hierarchies correspond to alternative retrieval methods, yet both systems arrive at the same leaf concepts in memory.

In particular, given enough instances, both systems establish all eight of the "leaf" categories

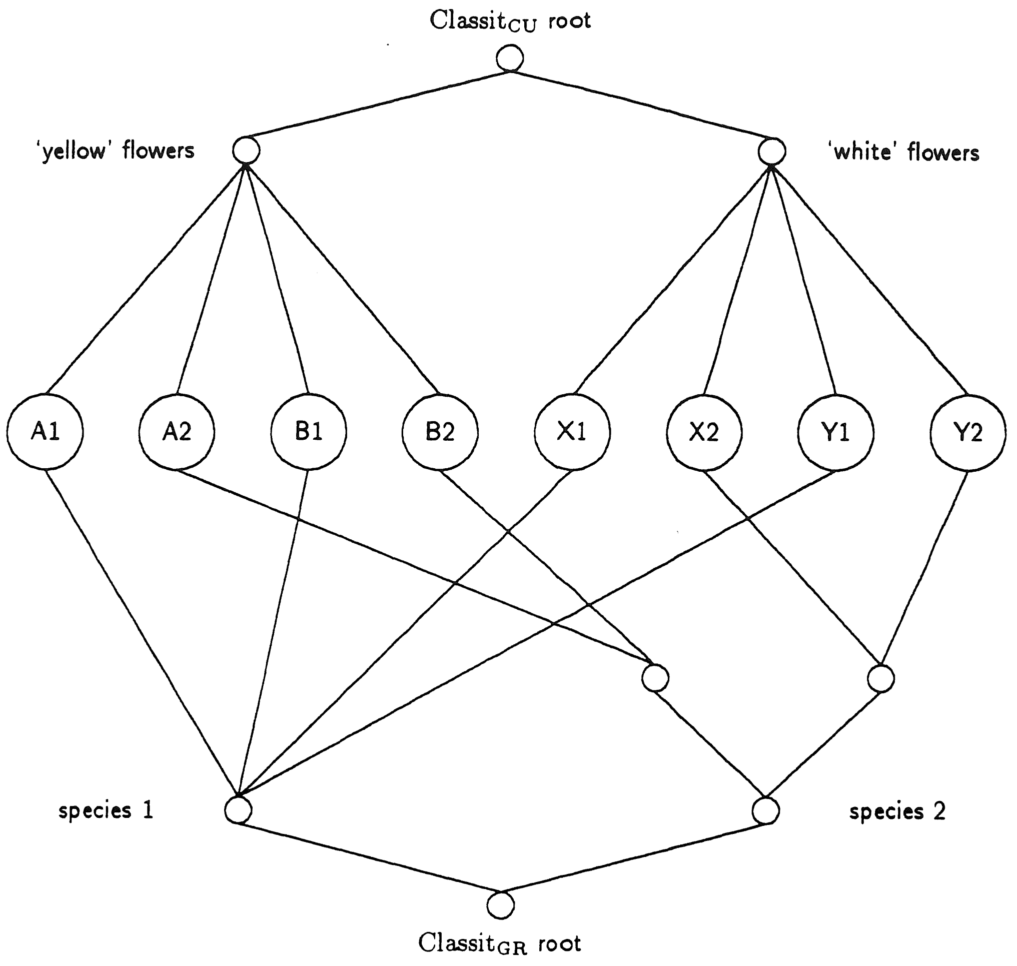


Figure 31. Competing hierarchies for Classit$_{CU}$ and Classit$_{GR}$.

shown in Figure 31. Once this happens, prediction is based on classes at that level (or lower). For example, suppose the system must classify a yellow, Species 1 instance (a 'B1' flower). Initially, Classit$_{GR}$ will group the instance with other Species 1 flowers, while Classit$_{CU}$ will cluster the instance with yellow flowers. However, the general CLASSIT algorithm continues sorting an instance until it creates a new disjunct. Assuming Classit$_{GR}$ and Classit$_{CU}$ have seen other 'B1' flowers, both systems will continue sorting the instance through their hierarchies until they reach a concept corresponding to the set of 'B1' flowers. At this point, the instance will be classified as a new disjunct under 'B1', and so prediction will be based on that class.

Figure 32 compares predictive ability with two attributes from the artificial domain: Attribute 1 is a 'color' attribute, so it has lower magnitude scores (and hence higher $Info(C)$ scores), while Attribute 5 is a 'species' attribute. Figure 32(a) shows learning curves for Attribute 1, while Figure 32(b) shows curves for Attribute 5. In both cases, Classit$_{GR}$ and Classit$_{CU}$ asymptote to the same level of predictive accuracy. Inspecting the hierarchies after they reach asymptote revealed that both systems have the eight 'leaf' categories well established.

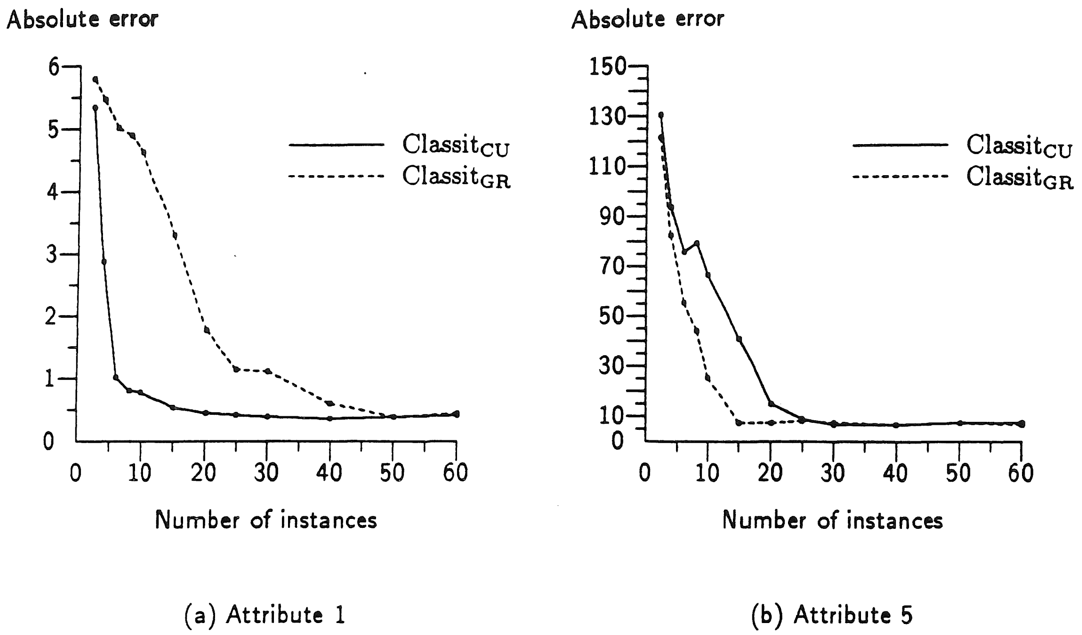


Figure 32. Learning curves for Classit$_{GR}$ and Classit$_{CU}$.

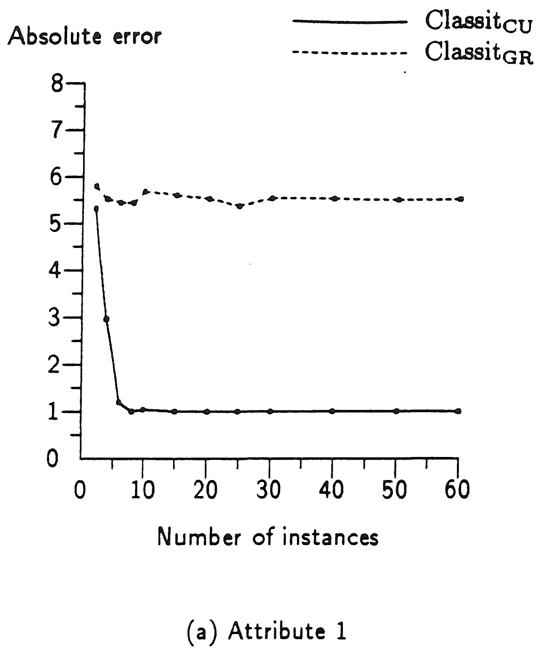
However, before CLASSIT defines the 'leaf' classes, one would expect a difference in predictive performance. As soon as Classit$_{GR}$ builds the level one 'species' classes, then it can predict species attributes. Meanwhile, Classit$_{CU}$ is building the 'color' classes, and therefore may not be able to make predictions about species. This effect can be seen in Figure 32(b); learning about Attribute 5 is slower for Classit$_{CU}$ than for Classit$_{GR}$. The converse is true for 'color' attributes, such as Attribute 1; this is demonstrated in Figure 32(a), where Classit$_{GR}$ is slower than Classit$_{CU}$.

This effect is more dramatic if one forces the systems to use only the level-one classes to make predictions. (This is the top-level prediction task described earlier.) In this case, Classit$_{GR}$ cannot make successful predictions about color attributes, while Classit$_{CU}$ cannot make prediction about species. This result is seen in Figure 33, which shows learning curves with top-level prediction for Attribute 1 (a 'color') and Attribute 5 (a 'species').

In summary, the use of gain ratio in this domain lets the system learn some attributes more quickly (though only at the expense of other attributes), and does not lead to better asymptotic performance. From this, one might conclude that gain ratio is almost useless, especially if one also considers its poor performance on real domains. However, there my be some situations where the gain ratio hierarchy is preferable.

For example, suppose one uses the hierarchies shown in Figure 31 to predict information about a new white flower of Species 1. That is, suppose the test set contains instances that are from the same species as instances seen during training, but of a different, unseen color. In this case, the Classit$_{GR}$ tree will still be able to predict species information, while the regular Classit$_{CU}$ tree may make a new disjunct out of the instance, and therefore make predictions based on the root node (as a 'naive' algorithm would).

In effect, gain ratio assumes that the lower noise attributes (the 'color' attributes, in this example)



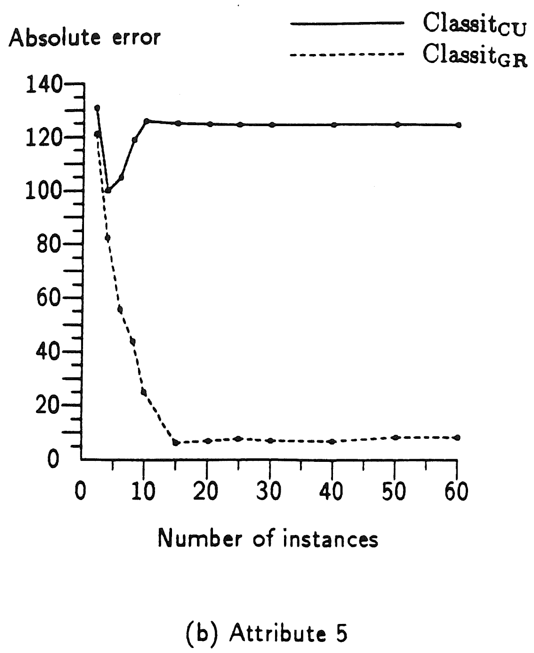


Figure 33. Top-level prediction with Classit$_{GR}$ and Classit$_{CU}$.

are not important and will not be used for predictive tasks. If there are test instances in new colors, this should not prevent the system from making predictions about species information. Silverstein and Pazzani (1990) suggest an extension of this idea where the system is told which attributes are likely to be used for prediction. The hope is that such a 'partially supervised' system would perform better than CLASSIT.

The use of the gain ratio evaluation function affects the predictive accuracy of CLASSIT in at least two situations: if the test set contains new types of instances, as described above, and if the assumptions behind gain ratio are not met (as seen with the automobile database). However, as with the previous section, the most direct effects from the use of gain ratio are on the the structure of the hierarchy, rather than predictive accuracy. Recall that I reported a similar result when experimenting with the merge and split operators (Chapter 4, Section 5). Just as with these alternative evaluation functions, removing merge and split affected the hierarchy but not the predictive accuracy of the system.

In many ways this parallels the results of Mingers (1989) for supervised learning. He found that for decision trees, the function used to select attributes during learning had an effect on the size of the learned tree, but little effect on its accuracy. The experiments in this chapter suggest that these results can be extended directly to unsupervised concept formation. Since the learning that occurs during concept formation is much less constrained than for the supervised construction of decision trees, this suggest that Mingers' results may be more far-reaching than anticipated.

3. Comparisons to other clustering algorithms

A significant criticism of machine learning work on concept formation suggests that researchers are 're-inventing the wheel' with respect to clustering algorithms. As I described in Chapter 2, researchers in cluster analysis have been working on the same basic problem since the 1960s. In fact, early machine learning work in conceptual clustering grew directly out of this tradition (Michalski & Stepp, 1983a). However, since most work in concept formation has not been compared to cluster analysis, the distinction between these two paradigms is unclear. Since concept formation is the more recent field, researchers in this field should make clear the strengths and advances of their work over older cluster analysis methods.

In this section I try to answer these concerns with respect to CLASSIT. I begin by reviewing some important qualitative differences between cluster analysis and concept formation, focusing on those differences that affect an experimental comparison between methods. I then look at the two most well-used algorithms in cluster analysis: agglomerative clustering and iterative optimization. In contrast to the previous section that evaluated algorithms, here I compare algorithms, keeping the evaluation function across methods as similar as possible. For both algorithms, I present an experimental comparison with CLASSIT on both an artificial and a real domain. Finally, I include some qualitative judgements about these different algorithms, making some assessment of CLASSIT's incremental approach to the clustering problem.

3.1 Concept formation versus cluster analysis

As discussed earlier, perhaps the most important distinction between cluster analysis and concept formation is the difference between incremental processing and nonincremental processing. Concept formation takes as input a sequence of instances, and it learns from each instance. The input to a cluster analysis algorithm is a fixed set of instances, and the learned concepts are created by processing that entire set of instances.

Since incremental methods do not reprocess instances during learning, one would hope that this approach is computationally more efficient than nonincremental clustering methods. This may or may not be true for a fixed set of n instances, but becomes much more likely if one wishes to reuse the learned set of concepts. For example, suppose a system has learned from a set of instances, A. If the system next needs to produce concepts for $A + \delta$, where δ is a smaller set of new instances, then an incremental approach will be much more efficient. A cluster analysis method would require re-computing all $A+\delta$ instances, while an incremental method will only process the δ new instances.

Because cluster analysis algorithms are not incremental, this experimental comparison will not include learning curves that show improved ability as a function of observed instances. It is possible to construct these for nonincremental methods, but only at a high cost: for every test point on the learning curve, cluster analysis must be reapplied to the complete set of instances observed so far. For applications that require this type of incremental response, it seems clear that a nonincremental clustering system is inappropriate. Rather than comparing learning curves, the next sections compare the performance of CLASSIT and cluster analysis algorithms with a fixed set of instances.

A second obstacle to a comparison between cluster analysis and concept formation is the lack of a well-defined performance measure for cluster analysis methods. As described in Chapter 2, the standard use for cluster analysis is simply to have an expert examine the output and subjectively decide the value of the concepts produced. In order to quantitatively compare performance, I will apply the performance tasks defined for concept formation to cluster analysis. Specifically, I use the set of concepts or tree produced by a cluster analysis algorithm to predict missing attributes of test set instances. This compares the accuracy of concepts; if cluster analysis produces a tree of concepts, I can also look at the average retrieval efficiency of that tree.

3.2 Agglomerative algorithms versus CLASSIT

Undoubtably the most well-used cluster analysis methods are those that use a hierarchical agglomerative algorithm. As described in Chapter 2, this algorithm begins by treating every instance as a separate class, and repeatedly combining these small classes until it puts together a single hierarchy that includes all instances. In order to decide which instances to combine, the algorithm requires a similarity matrix that shows how similar every instance is to every other instance.

More specifically, in this section I will use an average linkage agglomerative clustering algorithm (henceforth, *ALA*) as implemented in the statistical package, SPSS-X.[8] This particular clustering method is useful for comparative studies because it is readily available, it is well known, and it makes clustering decisions in a way that is related to CLASSIT's use of category utility. ALA uses Euclidean distance as its similarity measure, and represents classes by their average values. As we showed in Section 5.2, this type of average distance measure is closely related to category utility.

Before presenting an empirical comparison between ALA and CLASSIT, I should point out some important differences that are apparent from a simple analysis of the algorithms. First, all agglomerative methods require the use of a similarity matrix of size $O(N^2)$, where N is the number of instances. Although both algorithms build and use a concept hierarchy of size $O(N)$, only ALA requires this additional N^2 memory requirement. Second, the hierarchies that ALA builds are always *binary* trees. This means that these trees will usually be both deeper and contain more nodes than equivalent concept hierarchies built by CLASSIT. These additional memory requirements may be a significant drawback for ALA, especially if N is large.

In order to measure the predictive performance of the agglomerative method, the final binary tree produced by ALA is passed to CLASSIT for testing. This lets one use the same test set and the same parameter settings (acuity and the recognition criterion) to evaluate the concept hierarchies created by the two methods. This means that the same performance system is used with both clustering methods, and allows for a direct comparison of the predictive accuracy of the competing concept hierarchies.

For this experiment I use both an artificial domain and the glass database. Since the glass domain includes a symbolic attribute, glass-type, this must be converted to a numeric attribute in order to be processed by ALA. Since the attribute is binary, I simply converted it to an attribute with range $(0, 1)$. The artificial domain is of medium complexity with a fair degree of noise: there are

8. This is also known as the *centroid* method.

Table 10. Agglomerative clustering versus CLASSIT.

	Predictive error	Average depth	Retrieval efficiency
CLASSIT	2.40	2.63	8.87
Agglomerative	1.71	4.67	9.33
Ideal Learner	1.81	1.67	5.33

(a) An artificial domain. Naive algorithm error: 30.00

CLASSIT	0.139	2.93	9.09
Agglomerative	0.182	11.73	24.2

(b) The glass domain. Naive algorithm error: 0.452

four top-level classes, two of which are further subdivided into two subclasses. Each instance has nine attributes, with four of these irrelevant to the class structure. The six subclasses overlap in the same way as the databases in Section 1; for this domain, $\sigma = 5.0$. (See Appendix B for the data generator.) I use this artificial domain with both ALA and with iterative optimization methods; it tests these methods along a number of dimensions such as noise, number of classes, and class structure.

For both the real and the artificial domain, CLASSIT and ALA learn from a training set of 60 instances. Because agglomerative clustering is order independent, it only makes one pass over the data, while the results for CLASSIT are averaged over ten random orders. After learning from the training set instances, the accuracy of the competing hierarchies are compared with the same test set. For the artificial domain, I used a noise-free test set of six instances, while with the glass database I used an unseen set of 33 instances. The missing attribute during testing is `glass-type` for the real database and a subclass attribute for the artificial domain.[9] Table 10 shows the average absolute error, the average retrieval depth, and the average retrieval efficiency (the number of nodes visited) for this experiment.

The basic result demonstrated by this experiment is that, although ALA does as well or slightly better than CLASSIT on predictive accuracy, it does so only with a less efficient tree. This can be seen by the greater retrieval depths of ALA on both domains and by its poorer retrieval efficiency. For both domains I show the error for a 'naive algorithm' which makes bases predictions on the

9. Because `glass-type` has been converted to a numeric attribute, the results in Table 10 are *not* comparable to the predictive performance reported when this attribute is symbolic.

mean value of the missing attribute. For the artificial domain I also include an 'ideal case' based on the data generator.

Just as the naive algorithm does not provide a real upper bound, neither is the ideal performance really a lower bound. Whenever there is noise in the data generator and a finite training set, it is possible to outperform the ideal learner. The 'ideal' error given in Table 10(a) indicates the amount that the training set deviates from the model. ALA outperforms this value by modeling the training set more exactly than is merited by the data generator. This difference should disappear as the size of the training set increases.

Although these are preliminary results, they do not support the use of this agglomerative clustering method. Unless there exist some domains where ALA significantly outperforms CLASSIT, the memory and retrieval cost of the agglomerative method suggest that one should prefer an incremental clustering algorithm. For the domains I have tested, this seems to be true even when the application is not incremental. For any environment where instances must be processed continuously, an incremental algorithm would be desirable.

3.3 Iterative optimization versus CLASSIT

The second cluster analysis algorithm I compare to CLASSIT is known as *iterative optimization.* As this method was developed partly in response to the computational and memory costs of agglomerative methods, iterative optimization is considerably more efficient than an agglomerative approach. As described in Chapter 2, this algorithm assumes that the number of classes, k, is known a *priori*. Given this number and an evaluation function, the algorithm iteratively assigns and re-assigns instances to one of the k classes, improving the score of the evaluation function with each assignment. When this score no longer rises, the algorithm terminates and returns the final set of classes.

The two most important differences between these algorithms and CLASSIT are that k is required beforehand and that a set of classes is created rather than a hierarchy. Although these are clearly drawbacks if one is interested in building a general-purpose concept learner, there are many situations where neither difference is important. For example, users can easily apply the algorithm with a number of different settings for k, choosing the best output from these. Also, as I have suggested earlier, many users are only interested in the number and definition of the major classes, rather than an entire hierarchy of classes. In fact, there are routines designed to extract class information from a hierarchy, such as would be built by an agglomerative method (Everitt, 1980).

As before, the instantiation of iterative optimization (henceforth 'IO') I use in this section is from the SPSS-X statistical package. This method uses Euclidean distance, and therefore its evaluation function is equivalent to $trace(\mathbf{W})$ (Hand, 1981). In order to compare the set of classes produced by IO to the concept hierarchy produced by CLASSIT, I construct a one-level hierarchy from the set of classes. As with ALA, I then pass this simple hierarchy to CLASSIT for testing on predictive accuracy. Unlike agglomerative methods, iterative optimization is order dependent, so the results I present for both IO and CLASSIT have been averaged over the same ten random orders.

Table 11 presents the average absolute predictive error in the same domains as in the previous

Table 11. Iterative optimization versus CLASSIT.

	Ideal	CLASSIT	Iter. Opti. $k = 8$	Iter. Opti. $k = 6$	Iter. Opti. $k = 4$	Naive Alg.
Accuracy	1.81	2.40	2.82	5.07	9.47	30.00

(a) An artificial domain.

	CLASSIT	Iter. Opti. $k = 6$	Iter. Opti. $k = 4$	Iter. Opti. $k = 2$	Naive Alg.
Accuracy	0.139	0.185	0.234	0.452	0.405

(b) The glass database

section. I chose values for k based on the domain. In the artificial domain, there are four major classes, and total of six subclasses. Since iterative optimization cannot capture subclass structure, it is unsurprising that it does poorly when $k = 4$. As the results show, this algorithm does best when k is set higher than the actual number of classes present. For the glass database, IO does not achieve good results until $k = 6$.

As with agglomerative methods, these results show that for these values of k, iterative optimization has somewhat poorer predictive accuracy than CLASSIT. One may wonder how IO performs with higher values of k. While this may lead to some additional improvement, higher values of k must eventually lead to poorer predictive scores, as the system begins to overfit the data.

In conclusion, for many applications iterative optimization seems less desirable than CLASSIT for two reasons: it requires the number of classes beforehand, and it builds a flat list of classes rather than a hierarchy. And as with any nonincremental system, iterative optimization is inappropriate when the application demands incremental response to new information.

4. Chapter summary

This chapter has evaluated CLASSIT by assessing various parts of the system independently. Experiments focused on three aspects of the system: the user-supplied parameters, the evaluation function, and finally, the algorithm itself. Before moving on to additional modifications to CLASSIT, I should summarize the main results that have emerged thus far.

During development and the initial testing of CLASSIT, I believed that settings for the recognition criterion and the acuity parameter had relatively little effect on predictive performance. Unfortunately, more extensive parametric studies show that the system is somewhat sensitive to these parameters, but I should stress that this problem is severe only with noisy domains. As shown in Chapter 4, for cleaner domains, a single setting of the parameters (usually 1 or 2 for acuity and 80% for the recognition criterion) can be used with a wide range of input domains. These studies also demonstrated that the parameters were useful for avoiding errors due to overfitting. In addition to correcting my initial hypotheses, these experiments have led to some ideas for improving or eliminating the parameters (see Chapter 7).

A comparison of alternative evaluation functions for CLASSIT also produced some unexpected results. In particular, I found that the choice of function affected the structure of the concept hierarchy rather than the predictive accuracy of the system. Recall that the two functions I tested were quite similar to category utility; I would expect that a significantly different function would affect predictive accuracy. Although somewhat surprising, these results with unsupervised learning confirm and extend the work of Mingers (1989) with supervised learning methods.

Finally, I compared CLASSIT to some older cluster analysis methods for clustering. This comparison showed that CLASSIT performs about equally with these systems, at least on the domains examined. However, there were a number of important qualitative differences. Cluster analysis requires numeric data, while CLASSIT allows for symbolic or mixed formats as well as numeric attributes. Moreover, cluster analysis methods are nonincremental, and are therefore most appropriate in applications without a need for incremental response. Finally, some cluster analysis methods have significant memory costs, while others need to know the number of classes *a priori*. These extra requirements suggest that one would prefer an incremental algorithm, such as CLASSIT, to cluster analysis methods.

In addition to these observations, I would like to stress the value of a careful experimental study. As I have indicated, the experiments with CLASSIT have usually resulted in a surprise of one kind or another. This demonstrates the importance of experimentation; by forcing researchers to support their hypotheses about the system's behavior, their research becomes more accurate and precise. I hope that together, Chapters 4 and 5 present a clear picture of the scope and limitations of CLASSIT's ability.

Chapter 6
Partial Information and Attention

Concept formation in a 'real' domain usually occurs without complete information about instances. In an attribute-value representation, this situation occurs when some attributes of an instance have an 'unknown' value . There are two reasons why a learning agent may need to work with this type of partial information. First, the environment may include instances with missing information. In diagnostic domains, information can be missing whenever a test takes too much time (or is too expensive) to carry out. In visual domains, information may be absent when features are obscured, or because there is noise in the visual sensor.

Second, a learning agent may work with incomplete information when the environment provides too much data, and the agent prefers using partial descriptions. Since any learning agent has limited processing capabilities, it may not be able to attend to all available information about an instance. Instead, it may prefer to focus on *salient* attributes, and ignore (treat as missing) the less important ones. In cognitive psychology, the ability of a person to focus on only part of the input data is known as using *attention* to select from perceptual data.

This chapter describes CLASSIT's approach to missing information, whether this occurs naturally in the environment or as a result of attention. The first section presents the system's method for working with missing information in the environment, along with some experimental results with the voting database and with an artificial database. The next section presents an extension to CLASSIT that enables the system to use an attention mechanism. This mechanism includes a definition of *salience*: a score that indicates which attributes are most important for making clustering decisions. In this way, clustering can occur by only observing the more salient attributes, and ignoring (treating as unknown) the less important attributes.

When describing the attention mechanism, I also briefly discuss its relationship to other work on attention, especially in cognitive psychology. Unlike much of the work in this thesis, this extension is motivated mostly by concerns for psychological plausibility. I further discuss this motivation for CLASSIT and the attention mechanism in Chapter 7, where I present the system as part of a larger cognitive architecture for intelligence.

1. Concept formation with partial information

Most real-world environments do not have complete descriptions for every instance. One of the most interesting applications for CLASSIT is in the domain of visual perception, as described in Chapter 1. In such a domain, instances may be partially occluded, and attributes may be unobservable from some views. Additionally, the visual sensors may not provide a perfect description of the environment: the image may be degraded due to lighting, visibility, distance, or even errors in the sensor. All of these problems can lead to an instance description with missing information.[1] The task for a learning agent is to recognize and learn from instances in spite of these omissions.

Of course, missing information is not restricted to perceptual domains. As mentioned earlier, diagnostic databases are often incomplete since it is not usually practical or possible to include every piece of information about an instance. In general, a learning agent should be able to learn from degraded instances as long as most of the 'important' information is still present. If too much information is missing, one would expect the learner's asymptotic accuracy to suffer. Additionally, one may expect slower learning with even a relatively small amount of missing information. After I describe CLASSIT's mechanism for missing information, I present some empirical results that provide initial support for these claims.

1.1 CLASSIT and missing information

As with a number of design decisions, I have chosen to use one of the simplest methods for dealing with missing information. Because CLASSIT's evaluation function can be defined *per attribute*, one can simply leave unknown attributes out of its calculation. For example, for instances with complete information, the category utility score may be averaged over ten attributes. If an instance appears with three attributes as 'unknown', then CLASSIT uses an average over the seven known attributes when computing scores as this incomplete instance is sorted through the concept hierarchy. In effect, this method lets instances be described by different numbers of attributes.[2] Except for this modification to the evaluation function, CLASSIT learns from instances with partial information in the usual manner. Instances are sorted through the hierarchy and concepts are modified and created as dictated by the evaluation function.

Because this approach is so simple, it has a number of advantages over other schemes. For example, one alternative for missing information would be to fill in some 'expected' value for the unknown attributes (Quinlan, 1986).[3] However, it is unclear what expected value to use. One could either use the expected value stored in each child concept (an optimistic assumption, since it implies a good match between concept and instance), or one could use the expected value at the parent concept (the most likely value for the entire partition). This approach is also more expensive than simply leaving missing attributes out of the computation. With CLASSIT's method, if 40% of the attributes are missing, the system performs 40% less computation.

CLASSIT treats missing information in the same way whether values are unknown because of the environment, because of some attentional mechanism, or because they have been omitted during testing. The prediction task used to evaluate the system (defined in Chapter 4) includes one

1. This should be distinguished from noise in the data. Noise occurs if a sensor gives an incorrect value for an attribute; an attribute is missing if the sensor does not give any value for that attribute. In Chapter 4, I discussed CLASSIT's approach to noise, whereas this chapter is devoted to missing information.
2. This implies that concept descriptions may not always include information about every possible attribute. In general, I have not explored the ability of using different attributes to describe different instances. The current system still requires a list of all possible attributes, although computationally this input could be ignored.
3. I should point out for Quinlan's (1986) ID3 system, finding an expected value is a more critical problem than for CLASSIT. Because each decision in the tree is based on a single attribute, ID3 cannot simply ignore a missing attribute as CLASSIT can.

missing attribute: therefore, during testing, all test-set instances have one fewer attribute than those observed during learning. In the next sections, I look at the effect of missing information during training; in other words, I evaluate the system's ability to learn from incomplete information.[4]

1.2 Missing information in the voting database

Of the five natural domains described in Chapter 4, only the voting database includes significant amounts of missing information. Certainly, future experimentation should include additional testing in other real domains, but for now, this is the only domain I have used to test the system's ability with missing information. The results shown in Chapter 4 for this domain were based on a subset of the database (50 training-set instances) in which all instances contained complete information. In order to assess CLASSIT's ability with missing information, I have run two comparative experiments with voting records that include unknown attributes values.

Figure 34 shows the learning curves from these experiments. As in Chapter 4, acuity is set to 1.0, the recognition criterion is 0.8, and results are averaged over ten orders. To compare performance, I used the same test set (of 30 instances) for all three runs. In order to isolate learning ability with missing information, rather than performance ability with missing information, all test set instances have complete information (except for the attribute to be predicted).

For the first experiment, I randomly chose another training set of size 50 where each instance contained at least one unknown attribute value. For this training set, 11.8 percent of the attributes were 'unknown'. Unfortunately, it is not very fair to compare performance with this training set against performance with a complete information training set. It is very likely that a sample of congressional representatives who always vote (no missing information) is qualitatively different than a sample that includes representatives with missing votes or abstentions. Since the test set contains complete information, the system is learning from one type of instance, and being tested with instances from a somewhat different population. Thus, I would expect predictive ability to suffer, even if CLASSIT has a perfect mechanism for working with missing information. This result can be seen in Figure 34, where there is a difference in accuracy of about 10 percent.

In order to avoid this problem and more directly evaluate the system, I compared results with a third training set. These data are simply an artificially degraded version of the original 'complete information' data set. The instances are degraded by randomly changing some percentage of their attribute values to 'unknown'. For this experiment, I used the same percentage of missing information (12%) as the second training set.

As Figure 34 shows, predictive accuracy with the degraded data is almost as good as with complete information. After 40 instances, accuracy with the degraded data shows some signs of overfitting; however, the error rate is still quite close to CLASSIT's ability with complete information. In general, all three curves show reasonably similar predictive ability. This may be due to the relatively small amount of missing information. Additionally, the class attribute (democrat or

4. An alternative approach is to test the system with different amounts of missing information in the test set. This is closely related to the *cued recall* test from cognitive psychology; I do not explore it here, although Martin (1989) has carried out some preliminary experiments with COBWEB, the predecessor CLASSIT.

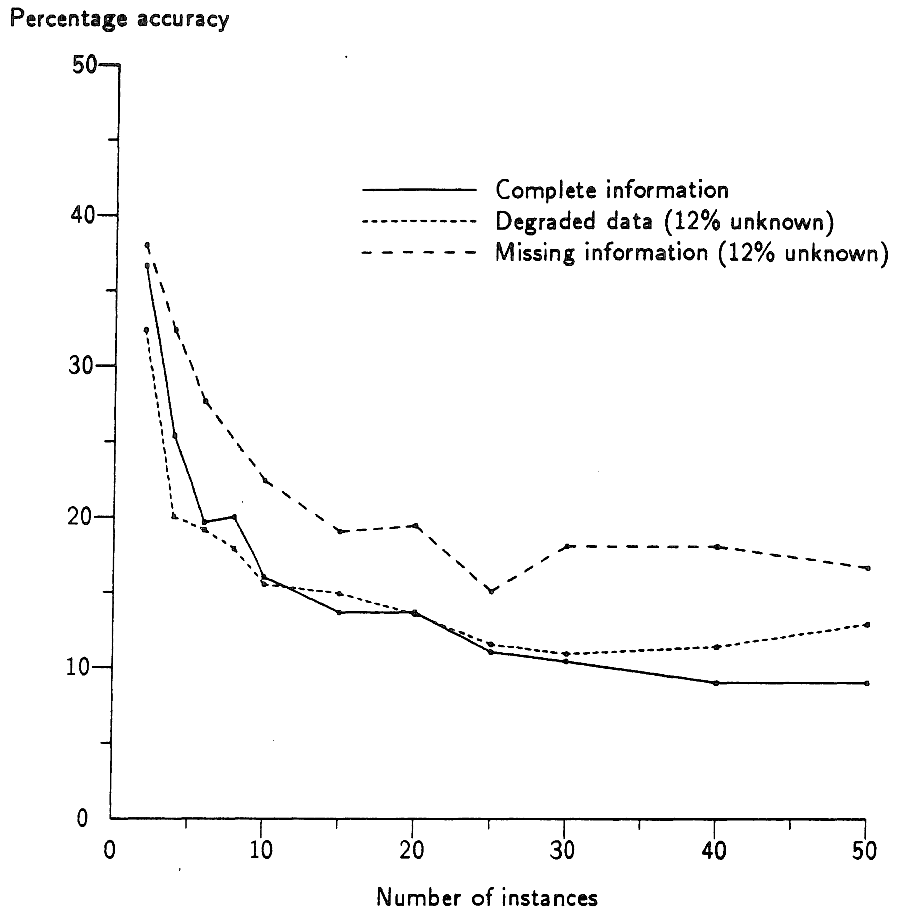


Figure 34. Performance with missing information in the voting domain.

republican) is never set to 'unknown'. In order to better evaluate CLASSIT's ability with missing information, I next turn to experiments with artificial databases.

1.3 Missing information in an artificial domain

To test CLASSIT's ability in an artificial domain, I use the 'Mixed-NTL' domain as defined in Section 4 of Chapter 4. This domain contains both continuous and symbolic attributes; I use a symbolic attribute for the prediction task. In order to introduce missing information into the domain, I used a filter that randomly sets an attribute value to 'unknown' at some desired probability. Figure 35 shows results as one adds 5%, 10%, 20% and finally 30% missing information into this domain. In each case, the figure shows a learning curve over 200 training-set instances.

These results show both that CLASSIT can learn in the face of significant amounts of missing information, and that learning proceeds more slowly under these conditions than with complete information. With up to ten percent missing information, there seems to be no difference in asymptotic accuracy, although the ten percent curve shows somewhat slower learning. For higher levels of missing information, the system learns both more slowly and asymptotes at a higher

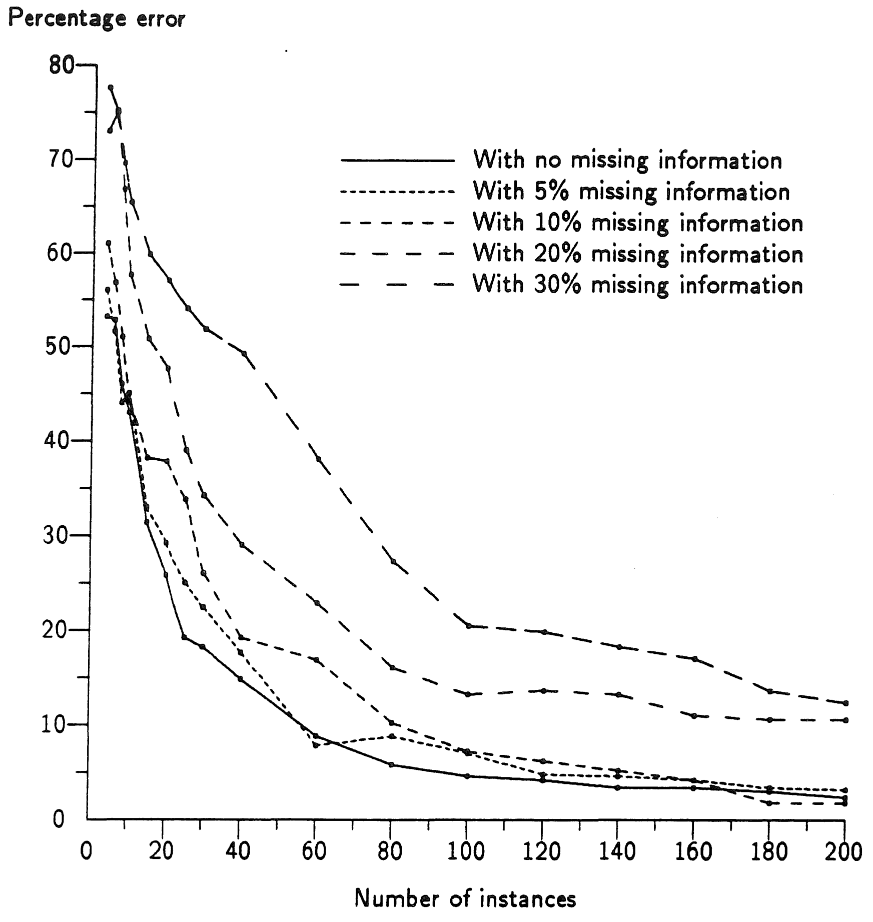


Figure 35. Accuracy with missing information in an artificial domain.

error rate. One would expect missing information to lead to slower learning: when 30% of the data is missing, the system must observe 30% more instances before receiving the same amount of information as when learning from complete instances.

Unfortunately, with 20 and 30 percent missing information, the system also shows a higher asymptotic error rate: about 10% rather than close to 0%. These rates can be compared to that for a naive algorithm, which has an error of about 83%. The higher asymptotes may be due to the system performing poorly with some instance orderings. Over 10 orders, after all 200 instances had been seen, the standard deviation for 10% missing information was only 1.6, whereas for 20%, $\sigma = 7.0$ and for 30%, $\sigma = 8.9$. This indicates that the variation in ability over different instance orders is much greater with higher levels of missing information. When a hill-climbing learning algorithm learns more slowly, it has more opportunity to get stuck at local optimum.

2. Concept formation with attention

For concept formation, attention means selectively observing and using only some of the attributes in an instance description. Especially in domains with many attributes, it seems unnecessary to inspect and process every attribute in a description. Instead, an attention mechanism should let the system classify an instance based only on a few important attributes. In order to do this, one must know the relative importance, or *salience* of attributes. Since one cannot expect this information to be supplied *a priori*, the system has a secondary learning task. In addition to learning concepts from the environment, CLASSIT must also learn the relative saliences of attributes in that environment.

Such a mechanism for attention is useful for two reasons. First, it allows for more efficient processing; it should decrease the number of attributes the system must inspect before reaching a classification decision. I expect that an attention mechanism will improve the efficiency of the system but not its accuracy.[5] For this reason, I stress retrieval efficiency as an important performance measure for attention. Second, research on incremental methods for attention is important because it takes a step toward a psychological model of human behavior.

I begin this section by motivating this research with a brief review of other work in attention, especially in cognitive psychology. This is followed by a careful description of the attention mechanism for CLASSIT, defining both the algorithm and the measure for determining salience. Finally, the section concludes with some experimental results that demonstrate the ability of the system with an attention mechanism.

2.1 Previous work on attention

Although attention is not often studied in conjunction with concept formation, there are a number of characteristics of the CLASSIT learning framework that led to an attentional mechanism as an obvious extension. First, although I am primarily interested in a learning system that is effective and computationally feasible, I have always tried to keep the design 'psychologically plausible'. Although the system cannot be considered an accurate model of human concept formation, I am interested in additions and modifications that could lead to such a model. An attention mechanism is exactly this type of addition. Second, I am most closely interested in applications of concept formation to perceptual processes, including vision. In this field, there are a number of "attention" theories, suggesting that such an addition would be useful in visual or perceptual domains.

Treisman (1969) describes two stages in the human perceptual process. First, there is a fast, parallel processing of low-level features. For example, the eye seems to process a large amount of sensory data (such as a retinal image) in parallel. However, before recognition can occur, this parallel feature-extraction process reaches a perceptual limit, and must give way to a serial selection among sensory input (or among a set of perceptual analyzers). This sequential selection among input features, when it occurs during recognition and concept formation, is exactly the process I wish to model with an attention mechanism.

It is important to note the relationship between this approach to attention and that of most research on visual attention. For example, LaBerge and Brown (1989) carefully detail a model of attention for shape recognition where the choice of what to focus on is known. This research begins with the assumption that a higher-order process tells the visual system to focus on certain features

5. It is interesting to consider whether an attentional mechanism can ever improve the accuracy of a learning system. After all, it has less information than a system with attention.

in the display.[6] In contrast, determining the saliences of features, or choosing what to focus on is part of the problem for the attention mechanism I present here. In fact, using relative saliences is one way to model the higher-order focusing process. Of course, if some other focusing information is available, it can be used to either pre-set the salience levels, or even to remove the less salient attributes from training instances.

Nosofsky (1986) uses a much looser definition of attention in his work with categorization. Here, selective attention takes place by defining a set of weights for the features in a domain. Higher weights on more salient features let the system effectively ignore the less salient features with lower weights. In contrast to the attention mechanism I propose, Nosofsky does not hypothesize any serial selection of attributes; all attributes are always processed. In addition, as Aha and McNulty (1989) point out, he does not describe how these weights may be learned.

There is at least computational model from cognitive psychology that explicitly looks at attentional learning without external feedback. Billman and Heit (1988) describe a model of learning from internal feedback that includes a focused sampling mechanism. Their system, CARI, learns relative saliences exactly as CLASSIT does: over time, the agent acquires knowledge about which features are most helpful for classification. Later, when the system must make decisions about a new instance, it inspects salient features with a higher probability than less salient ones. However, CARI is designed with a different representation and different goals from CLASSIT: it learns only simple rules about pairs of features, and does not produce a concept hierarchy of any kind.

2.2 The CLASSIT attention algorithm

At a computational level, the use of an attentional mechanism is similar to learning from incomplete information. In both cases, the basic concept learning algorithm is unchanged. As each instance is sorted through the hierarchy, the system must make decisions based on partial information. With attention, decisions about what information to ignore are made by the system; with missing information in general, information is removed by an external process, possibly in a random fashion. The attention mechanism I present here describes how the system chooses which attributes to ignore.[7]

There are two parts to CLASSIT's attention mechanism. First, the system learns and stores relative saliences among attributes. Then, the system uses a sequential focusing algorithm that chooses among attributes, inspecting only enough information to make a confident clustering decision. Salience is defined as the per-attribute contribute to category utility (see Equation 1 from Chapter 3). Hence, for a given attribute i,

6. The open and interesting questions for this line of research involve how attention affects visual processes. Schneider, Dumas, and Shiffrin (1984) also work from this starting point, while McNulty (1988) looks at these questions from a computational point of view.

7. This chapter gives a more detailed presentation of the attention mechanism originally described by Gennari (1989).

$$Salience_i = \frac{\sum_k^K P(C_k) Info(C_{ik}) - Info(C_{ip})}{K},$$

where $Info(C_i)$ is the information of attribute i in concept C, $P(C_k)$ is the probability of the class k, and K is the number of classes at the current level.

These scores produce an ordering of the attributes from most salient (attributes that should be inspected first) to least salient (attributes that probably need not be inspected). This ordering is dynamic because it is connected to the basic learning process. As the set of concepts changes durings learning, the salience scores also change, allowing for different attributes to emerge as most salient at different times.

Given an order in which to inspect attributes, the system must decide how many attributes it should inspect before making a clustering decision. CLASSIT resolves this 'stopping condition' problem by imagining a worst-case scenario: it imagines that the unobserved attributes match some other concept perfectly, and then considers whether this information would change the current clustering decision. If so, then one must continue inspecting attributes; if not, one is guaranteed to have inspected sufficient attributes to make a decision.

Finally, an attention mechanism needs an ability to recover from order effects. Like any incremental system, CLASSIT can be fooled by unrepresentative initial instances: these may cause the system to have low initial estimates of the saliences for important attributes. To recover from this situation, the system probabilistically inspects attributes as a function of their salience scores, allowing even low-scoring attributes to be occasionally inspected. If such an attribute is 'noticed' in this way, and if that attribute actually is salient for the new instance, then its salience score is updated, and it will be more likely to be inspected in the future.

Table 12 presents the attention algorithm used in CLASSIT. This mechanism is embedded within the basic concept formation algorithm as described in Table 2 in Chapter 3. In particular, attention is used whenever making a clustering decision: adding to an existing node, making a new disjunct, or carrying out a merge or a split operation. Attention and concept formation together carry out a nested hill-climbing search: a search to learn the salient attributes within a search to learn the best concept descriptions. Note that for each level in the concept hierarchy, there may be very different salience scores. Hence, CLASSIT restarts the attention algorithm at each level, inspecting completely different attributes at different levels.

In order to determine if enough attributes have been inspected, the stopping condition (Step 4) considers two alternatives. First, the unobserved attributes may indicate membership in an alternative class. Although each existing class could be tested, I have limited this comparison to the second-best concept (just as the merge operator is limited to the best two concepts). Second, if this test indicates that further attribute inspection is unnecessary, the system considers a new disjunct. This condition covers the situation in which the unseen attributes indicate that this instance should be placed into a new singleton class. Only if both tests yield a lower score than that based on the observed attributes does the system ignore the remaining attributes and continue with the classification.

Table 12. An algorithm for attention

```
1.  Select an unseen attribute with probability based on the salience scores
    stored at the parent node.
2.  Compute the salience of the selected attribute; store this new score at
    the parent.
3.  Compute the category utility score for the best classification, $X$, based
    only on attributes inspected thus far.
4.  Consider all remaining unseen attributes and compute scores if these
    attributes were to match either of the following alternative
    classifications:
       a) The second best concept.
       b) A disjunct.
5.  If either of these scores is better than $X$, then go to step 1.
    Else, ignore remaining attributes.
```

With little or no previous information, all attributes are equally salient, and the system must inspect most or all attributes before choosing a clustering operator. However, as more instances are observed, concepts should emerge in which some attributes contribute heavily to the total category utility score, while others contribute less. This means that the salience scores for attributes become more disparate, letting the system inspect only those attributes that have high scores. The attribute learning process is synchronous with the concept learning process: as the system defines concepts, it learns which attributes are more salient.

As stated earlier, the purpose of an attention mechanism is to improve efficiency: by looking at fewer attributes, the system should be able to recognize or classify instances more quickly. Yet there is no improvement in computational efficiency with this algorithm. In particular, by applying the halting condition after observing each of n attributes (step 4 in the table), I have added an $O(n^2)$ cost to the algorithm.[8] However, I am assuming that the cost of observing an attribute is far greater than the time required for an internal computation. This seems very reasonable if one imagines an application to robotics, where considerable work and real time may be needed to observe features (Tan & Schlimmer, 1989), or to diagnosis, where observing an attribute may mean carrying out a lengthy test. Given this assumption, efficiency can be defined as the number of attributes inspected, since this cost outweighs other computational costs.

8. Note that there is a small constant on this cost: on average, if attention observes half of the attributes, then the cost of the halting condition is at most $(n^2 - n)/4$.

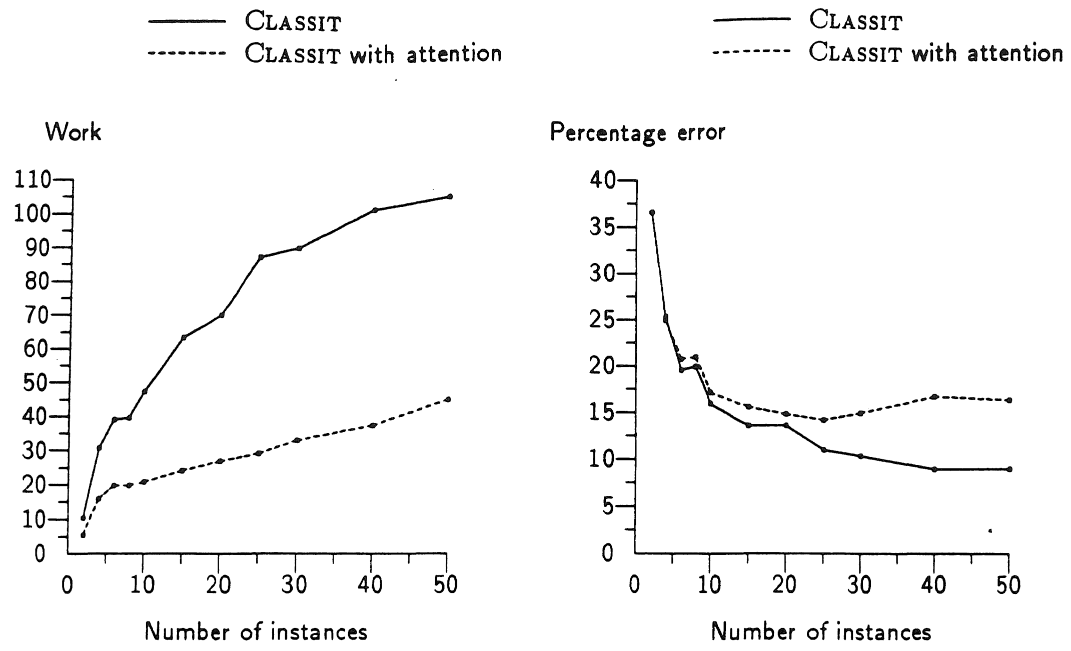


Figure 36. Performance with attention on the voting database.

2.3 Experimentation with attention

This section reports tests of the basic hypothesis about an attention mechanism: with attention, the concept formation system can process instances more efficiently without a loss in predictive accuracy. To check this claim, I use four of the real domains described in chapter 4: the voting database, the glass database, the heart-disease database, and the automobile database.[9]

For each domain, I show performance with and without attention, comparing both efficiency and accuracy. Figures 36 through 39 show results with these data sets. For each domain, the experimental procedure and parameter settings (and results without attention) are the same as those described in Chapter 4. Unfortunately, the attention algorithm is non-deterministic: because it inspects attributes with probabilities based on the salience scores, performance may vary with different executions, even when the same data is presented in the same order. Hence, predictive performance of the attention algorithm is averaged both over ten orderings of the training data, and over five executions for each ordering.

Efficiency was defined in Chapter 4 as the average number of nodes inspected during test-set classification. In order to evaluate the ability of an attention mechanism, this definition must be augmented to include attributes as well as nodes. More exactly, during the classification of a test set instance, I count the number of nodes in the current partition and multiply this by the number of attributes inspected at that partition. The sum over levels is the 'work' reported in Figures 36 through 39.

9. Because of the size of the database, it was impractical to experiment with attention using the LRS star database.

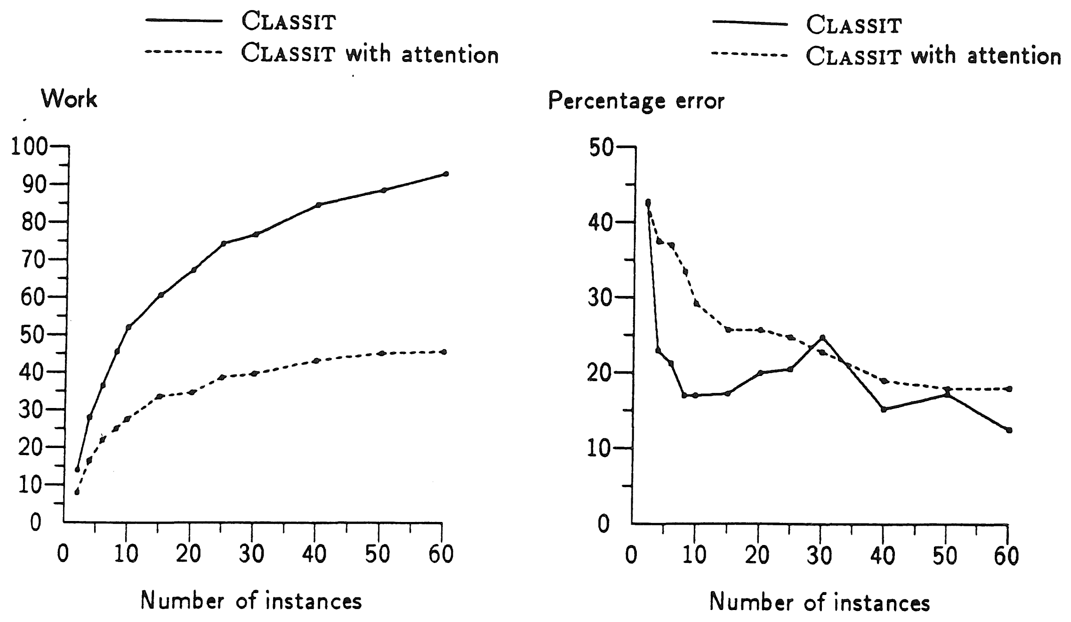


Figure 37. Performance with attention on the glass database.

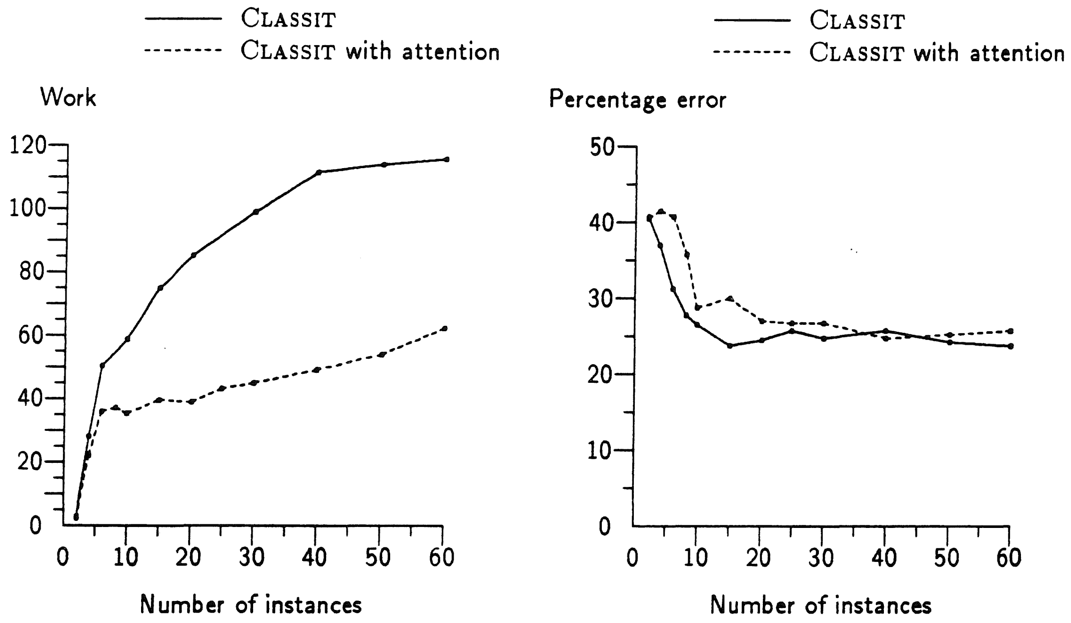


Figure 38. Performance with attention on the heart-disease database.

I also have included an artificial domain that was designed especially for an attention mechanism. This domain is quite clean, with little or no overlap between classes. Instances are described by twenty attributes, of which twelve are completely irrelevant to class structure. The results in Figure 40 show that learning is slower with attention, since the system must search for the salient

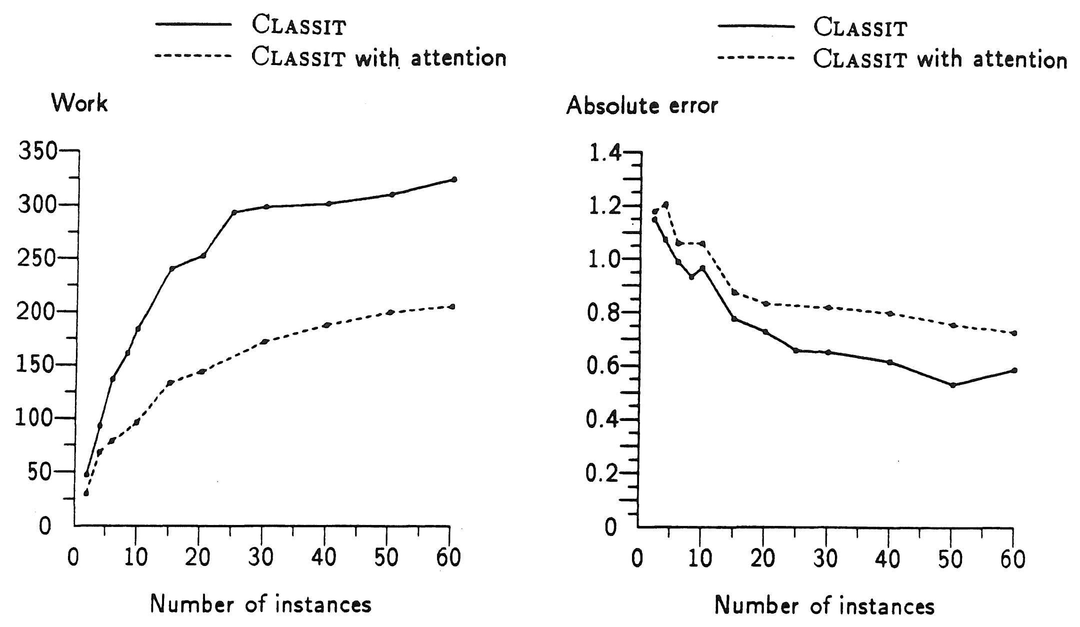


Figure 39. Performance with attention on the automobile database.

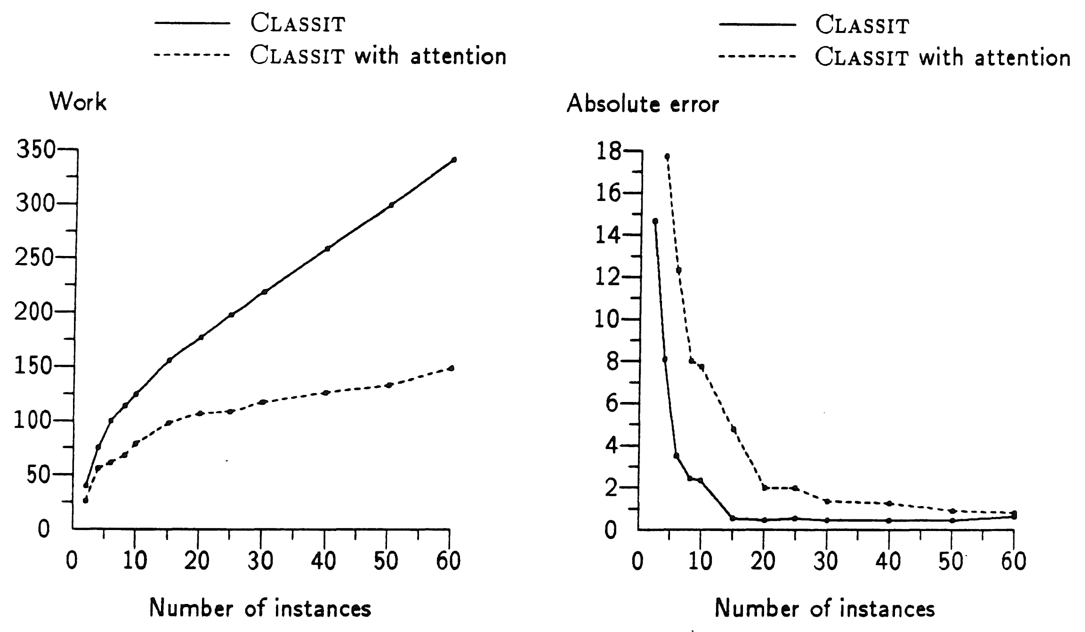


Figure 40. Performance with attention on an artificial database.

attributes. However, once the system finds these attributes, predictive performance is equal with and without attention. This slower learning can also be seen to some degree in the glass and heart-disease domains.

In all these domains, the attention mechanism roughly doubles the efficiency of CLASSIT. As expected, the **shape** of the efficiency curve does not change: with or without attention, the work appears to rise as the log of the number of instances seen. As hypothesized, this improvement in efficiency does not cause a major loss in predictive accuracy, although the auto and voting databases show slightly better predictive performance without attention.

3. Discussion

At one level, the attention mechanism described in this chapter resulted from exploratory research with concept formation and missing information. That is, I believed that CLASSIT did not need to use all attributes in order to build useful (accurate) concepts. Instead, I posited that the most salient attributes could be discovered by the system, and that the system could perform well while inspecting only the salient features of its environment, just as it could perform well with only partial descriptions of the data. The mechanism presented here is the result of this line of inquiry.

I certainly that the results shown here have satisfied these initial goals. As a rough approximation over all domains, CLASSIT seems able to learn accurate concept hierarchies while only inspecting (on average) half of the available attributes. However, this initial success has led to a set of additional objectives, most of which are only partially fulfilled.

For example, I would like to pursue the relationship between attention as defined by cognitive psychology and the algorithm used by CLASSIT. Also, I am not satisfied with the way that the attention mechanism re-starts with each new level in the concept hierarchy. They may also be other, more successful ways of defining salience and of determining a 'stopping condition' when inspecting attributes. The next chapter includes a closer look at these issues, and describes some preliminary ideas for solutions.

Chapter 7
Reaching beyond Classit

Any new scientific theory or engineering advance can be evaluated in two ways. First, the work can be evaluated simply by comparing its abilities to the researcher's goals: a new model should explain phenomena and a piece of engineering should meet its specifications. Second, any significant scientific advance should motivate additional research in new directions and should be useful in a variety of applications. I hope that this thesis provides an evaluation of Classit along the first dimension. My goals should be clear, and I have given evidence that Classit meets these goals. Along the second dimension, perhaps the best way to evaluate a system is observe the effect of that research over the course of time. Another way is to examine the new ideas and related work that result from the new system. Therefore, this chapter presents some initial ideas for extending Classit in new directions and in new applications.

Before suggesting new directions to move in, one should present a clear picture of what has been accomplished so far. Hence, this chapter begins with a summary of the research contributions of Classit. Next, I describe some potential improvements to the system parameters, and some possible modifications to the data structures and algorithm. The final two sections of the chapter describe how Classit might be used as part of a larger system. First, I describe how concept formation might used as part of a vision system, aiding the transition from pixel information to higher-level representations for visual recognition. Second, I describe a general purpose architecture, Icarus, designed as an integrated model of an intelligence agent (or robot) reacting with the environment. Classit was developed with this architecture in mind, and is designed to be used in a number of Icarus components that are currently under development and testing.

1. Summary of contributions

Although Classit does not provide a truly new framework for concept formation, it is an extension of existing work, especially Fisher's (1987) Cobweb, in a number of important ways. By building on Cobweb's algorithm for concept formation, I have been able to experiment with a number of previously untested hypotheses and assumptions. In particular, some of the featured abilities of Classit are also true of Cobweb, but were never tested or carefully evaluated. This research also goes beyond Cobweb in generality – the system can be used in domains where Cobweb is not applicable. In summary, I believe that Classit is a robust and general-purpose system for concept formation. Its abilities include:

- The ability to learn a variety of class structures. Classit (like Cobweb) can work with domains that have many or few classes, organized in a hierarchy or as a flat list.
- The ability to learn from noisy domains. Classit's probabilistic representation of concepts allow it to easily work in domains with noise. (This is also true of Cobweb, but had not been well-tested.)

- The ability to learn with symbolic, numeric, and mixed-format data. CLASSIT's evaluation function is defined for either numeric or symbolic attributes. Likewise, its concept representations allow for a hybrid representation of numeric and symbolic information. The ability to work with mixed-format data allows the system a much greater range of application than most other concept formation systems.
- The ability to ignore irrelevant attributes. Regardless of the number of irrelevant attributes, CLASSIT (and COBWEB) can learn concepts based on only a few distinguishing attributes.
- The ability to learn from data with missing information. Since CLASSIT's evaluation function is computed per-attribute, the system can ignore attributes that are missing or deleted from the input. (Again, this is true for COBWEB, but had not been well-tested.)
- The ability to learn the relative salience of attributes in an input domain. CLASSIT can then use this information with an attention mechanism, focusing on the salient attributes, and making clustering decisions with only a subset of the available information. This mechanism is an exploratory attempt to define attention for concept formation in a psychologically consistent way.

I have demonstrated each of these abilities with both real and artificial domains. Experiments with real domains provide a demonstration that a particular ability is useful in the natural world, while experiments with artificial domains more precisely demonstrate the scope and limitations of the system.

In addition, I have experimented with some of the components of CLASSIT independently. These experiments include comparative studies, where I compare different versions of the learning system with the same training and test-set instances. I have compared CLASSIT with:

- *Other evaluation functions.* I found that predictive accuracy was not strongly affected by any of a set of related evaluation functions. Instead, these alternatives had a more direct effect on the structure of the concept concept hierarchy built by CLASSIT.
- *Cluster analysis methods.* Compared to CLASSIT, traditional cluster analysis methods are special purpose, allowing only numeric attributes, working only in nonincremental settings, and sometimes requiring additional domain information. However, even in settings where cluster analysis was applicable, CLASSIT appeared to perform as well as these more specific methods.

These studies give additional evidence that CLASSIT is a robust system.

Finally, an important contribution of the thesis is the experimental methodology followed throughout the work. This methodology stresses the importance of well-specified experiments with a variety of domains. In order to evaluate a learning system, the researcher must define both a learning task and a quantitative performance task. For CLASSIT, I define performance tasks for both the accuracy of the learned concepts, and for the efficiency of the system's concept hierarchy. Using these measures, my claims were supported and sometimes overturned by experimentation. This approach to research helps avoid ad hoc learning systems, and forces the researcher to support hypotheses about the behavior of the system.

2. Extensions to CLASSIT

Before considering completely new research directions and applications for CLASSIT, I begin by addressing some problems and possible improvements to the existing system. This section makes suggestions for improvement in three areas – improvements for the system parameters; a more general organization of concepts; and some alternative algorithms for attention.

2.1 Improving CLASSIT's parameters

As described in Chapter 5, CLASSIT currently uses two parameters: an acuity parameter and a recognition criterion. Ideally, a learning system should be parameter-free: a system that includes parameters suggest that it must be "fine tuned" to every new domain. Hence, one area for improvement would be to either remove these system parameters or to make CLASSIT less sensitive to their values.

The recognition criterion is used to determine when an instance is close enough to an existing concept that it need not be saved or classified further through the hierarchy. The easiest way to eliminate this parameter is to save all instances, and never 'recognize' an instance until the classification process reaches a leaf node. However, as shown in Chapter 5, this is too high a price to pay. Not only is retrieval less efficient, but in noisy domains, accuracy is lower as the system begins overfitting the data.

One way to improve this parameter is to recognize that its value represents a tradeoff between careful learning and rapid recognition. High values suggest that the system learn precisely where the new instance belongs in the concept hierarchy, and that the hierarchy may need modification to accommodate this instance. Low values for this parameter mean that the system is interested in a quickly recognizing the new instance. In this case, the system carries out minimal learning and treats the hierarchy more as a static knowledge structure.

This suggests some ways to automatically set the recognition criterion. For example, there may be distinct 'learning' and 'recognition' episodes for a given domain. A larger system may also have timing constraints – when it needs to recognize something quickly, it can use a lower recognition criterion. More generally, the system could track the changes to its concept hierarchy and modify the recognition criterion based on the amount of learning occurring. Initially, the recognition criterion should have a high value, possibly even 100%, thus forcing the system to save all instances. However, as the concept hierarchy becomes relatively static, the system can assume that most of the learning for this domain is complete, and therefore the recognition criterion can be lowered.

In general, this suggests a dynamic parameter that adjusts to the amount of learning that occurs. For example, if the environment changes after some point in time, one would expect the system to emphasize learning rather than recognition until it became familiar with the new environment. Schlimmer and Granger (1986) address similar issues as "tracking concept drift". One simple way to track changes is to record how often the system invokes the operators "merge" and "split". When these are used, the tree is reorganized, and hence the recognition criterion should probably be raised. If these are rarely used, then the system can assume it has learned about the current environment, and can lower the recognition criterion.

For any continuous attribute, CLASSIT also employs an acuity parameter. This parameter is used to determine when two continuous data values are so close as to be effectively the same. Unfortunately, I expect that this parameter will be very difficult to completely eliminate. Because continuous data can take an infinite variety of ranges and values, it seems natural to adjust the system to account for differences in the range of values. Acuity is similar to the coupling parameter of Anderson (1988) and Anderson and Matessa (1990). Fried and Holyoke (1984) also use an analogous initial variance for continuous attributes.

As described in Chapter 5, acuity is based on the idea of 'just noticeable differences' from psychophysics. This implies that there should be a different acuity value for every attribute. A possible modification of the acuity parameter is to use a vector of acuity values, and then to require that the system learn the values for this vector. In particular, each value in this acuity vector can be dependent in some way on the range (or variance) of the corresponding attribute. Higher variances suggest that greater acuity values are needed, and vice versa. Of course, this does not eliminate acuity: an initial setting is needed and the user must specify the relationship between variance and acuity values. Since the variances change over time, the acuity values would also change. The implications of changing what was a constant parameter into a dynamic variable are unclear.

2.2 Modifying CLASSIT's concept hierarchy

CLASSIT currently uses a concept tree and enforces a strict partitioning of instances into classes at each level. One possible improvement for the system is to relax these assumptions. For example, one can allow an instance to be a member of two classes, rather than a single class. This implies that concepts may have more than a single parent, indicating that it is a member of both concepts. This is a step toward the approach taken by Cheeseman et al (1988), where each instance belongs to all classes with some probability. Lebowitz (1987) also supports a form of 'clumping' in his UNIMEM system.

This changes the basic data structure from a tree to a directed acyclic graph (DAG), which is a more general structure than a strict hierarchy. There may be a number of applications and domains that include instances that should properly be in more than one category. Also, one can consider a path through the concept DAG as a chain of reasoning, or as the steps needed before recognition occurs. A concept hierarchy allows only one route to each concept, whereas a DAG would allow a concept to have more than one 'recognition chain'.

In fact, this more flexible recognition can lead to more accurate predictive ability. Martin (1990) is currently investigating a closely related idea of "overlapping categories". For certain domains that are poorly represented with disjoint concepts, he reports greatly improved accuracy when the system builds overlapping categories.

This change in data structure necessitates a change in both the algorithm and the evaluation function. The algorithm must be changed since the system no longer chooses a single concept at each level. The evaluation function must also be changed, since it cannot simply compute the information at the parent when there is more than one parent. These are only some of the design decisions needed to modify CLASSIT so that it builds concept DAGs.

2.3 Modifying the attention algorithm

As suggested at the end of Chapter 6, I am not completely satisfied with the attention algorithm currently used by CLASSIT. In particular, the system treats attention as an isolated subroutine that is restarted at each level as the recognition process sorts an instance through the concept hierarchy. Although this approach has some advantages, in this section I present two alternative ways of integrating recognition and attention.

One way to easily improve efficiency is to modify the attention algorithm so that an attribute is only observed once during the recognition process. The observations made at one level are inherited at the next level; hence, more and more attributes are 'known' as one descends the hierarchy. Except for at the top level, this variation would have more attributes available when it makes clustering decisions. Since the system can ignore irrelevant attributes, this suggests that accuracy may be improved, or at least not worsened by this alternative attention method. Whenever CLASSIT uses an attribute value more than once during classification, this version of attention would only 'observe' the attribute once. If work is measured by counting observations, and does not include the cost of storing and retrieving attribute values, then this modification will improve efficiency. Thus, efficiency is improved without sacrificing accuracy.

A second alternative is to completely invert the relationship between the classification process and the attention mechanism. Instead of classification calling attention as a subroutine to make a clustering decision, the attention mechanism would call the entire recognition process each time it considers inspecting a new attribute. This approach is closer to the psychological view of attention: a poor job of recognition may demand that the system observe more attributes, while a 'good' recognition may let the system stop observing attributes, even if only a few are known.

Unfortunately, this modification raises a number of unsolved design problems. First, salience scores cannot be stored as they were, with a different set of values at every parent node in the tree. The simplest solution is to store only a single set of saliences for the entire hierarchy. This would define salience for the entire domain, rather than for a particular set of concepts. Second, the stopping condition currently used by CLASSIT cannot easily be extended to this version of attention. One possibility for a stopping condition is to compare alternative clustering paths through the hierarchy, rather than the second-best class at a given partition. However, this idea raises a number of additional problems when sorting an instance down more than one path through the concept hierarchy. In spite of these dilemmas, allowing attention to call recognition rather than the reverse is an interesting research direction.

3. Classit and visual processing

As described in Chapter 3, much of the motivation for this research arose from the needs of an unsupervised robot as it explores an environment. This section focuses on one problem from this application: visual processing and concept formation.

Although there are many representation schemes for visual information, one common way to represent input is as a two dimensional grey-scale grid of pixels. To apply concept formation to this domain, I treat each 'view' as an instance and each pixel location in a view as a single attribute. Thus, instances in this domain have an extremely large number of attributes. As in the current system, a visual concept is similar to the representation for an instance. However, concepts summarize a number of views, so each attribute in a visual concept includes a mean and a standard deviation for the grey-scale values. These concepts are "probabilistic grids" that indicate the probabilities of values in the visual field.

Matching and retrieval in this framework would differ little from the current system, with each cell contributing to the overall category utility score.[1] This approach to recognition in visual domains is similar to simple *template matching*, where an instance is matched, pixel for pixel, to a template of the desired concept. However, here the templates are not exact, but may be very 'fuzzy' if they include high standard deviations. Also, the match process is not all-or-none, but instead returns a score. The system can then make decisions simply by choosing the match with the highest score.

The sheer number of attributes, and the difficulty of observing all attributes, make this application an obvious one for some form of an attention mechanism. However, the approach described in Chapter 6 is probably too fine grained for visual data. Rather than attending to one pixel at a time, the system could focus on small regions of the instance. This is a coarser-grained attention mechanism, but should still avoid inspecting all attributes in the instance. These ideas are similar to those of McNulty (1988) in the domain of letter recognition.

It should be clear that there remain a large number of open research questions for this application of concept formation. For example, it may be possible to organize visual concepts into a different type of specific to general hierarchy, where the concept grids have higher resolution and detail for specific concepts and lower resolution for more abstract, general concepts. Another open issue is to evaluate how well 'fuzzy' matching will work with three-dimensional objects or with multiple views of the same object. In summary, I believe that Classit suggests a promising approach to this area of research, especially because of its ability to deal with numeric values and noise.

4. The Icarus architecture

The Icarus project is an ongoing research effort to define and build on an integrated cognitive architecture (Langley, Thompson, Iba, Gennari & Allen, in press). By an 'architecture' we mean a single framework of research assumptions and constraints that can be used to address a variety

1. Preliminary experiments with this method (along with techniques to handle variations in scaling, translation, and rotation) on a two-dimensional letter-recognition task have produced encouraging results (Tami Tracey, personal communication).

of tasks. In general, we are attempting to build a system that could control the behavior of an autonomous agent (e.g., a robot) in a reactive environment. Since this architecture encompasses a variety of abilities under one theoretical framework, it is comparable to Anderson's (1983) ACT* architecture and Laird, Rosenbloom and Newell's (1986) SOAR system.

The ICARUS architecture includes a number of important assumptions and biases about representation, performance and learning. For example, it states that an agent should have direct interaction with the environment via sensory input and robotic effectors. However, we are currently working with a simulated world rather than actual robotics, and we treat early vision and primitive motor control as solved problems, leaving this work to other researchers. Thus, we are working at a somewhat more abstract level than robotics.

ICARUS also assumes that all symbols and representations of knowledge are ultimately *grounded* in some sensori-motor description. This implies that concept formation must work with real-valued attributes such as the size, position, and velocity of objects, rather than higher-level symbolic features. This assumption forces us to model cognition at a 'lower', more primitive level than many AI researchers.

An important goal of this project is to create a well-integrated system. Hence, although ICARUS supports a number of distinct research projects, we ultimately expect these to be tied together in a single system. For this reason, the design of ICARUS includes a number of shared data structures and control mechanisms. For example, there is a a sensory (or perceptual) buffer for receiving information about the environment, and a motor buffer for sending commands to move or affect the environment in some way. In addition, ICARUS assumes a single long-term memory structure, represented as a probabilistic concept hierarchy.

Finally, ICARUS uses concept formation as the single retrieval and learning mechanism for long-term memory. Because this is so central to the architecture, CLASSIT provides a foundation for ICARUS. There are currently three major components of ICARUS: a system for planning, DÆDALUS; a system for motor control, MÆANDER; and a system for object recognition, LABYRINTH. For each of these, concept formation is the central learning mechanism, and some form of the CLASSIT system has been incorporated into these systems. I next briefly describe these three components, emphasizing the role of concept formation in each.

4.1 Concept formation with composite objects

CLASSIT and most of its predecessors in machine learning and cluster analysis use instances that are described by a simple list of attribute-value pairs. However, objects in the physical world may have components organized in a complex relational structure. LABYRINTH (Thompson & Langley, in press) is a component of ICARUS that carries out concept formation over *composite* objects, i.e., instances with attribute values which are objects that can be further decomposed. For example, a cup may be described as having two components: a handle and a body. These components may be represented as simple attribute-value lists, or they may be further divided into subparts.

LABYRINTH includes a number of extensions to CLASSIT that let it operate in a domain with composite objects. First, the representation of instances and concepts include part-of links. Instances are presented to the learning agent as a tree whose root is an entire composite object and whose leaves are primitive components. Likewise, concepts can include part-of pointers to other nodes in the hierarchy that may act as components for that concept. Recognition of a structured instance occurs in a bottom-up fashion, beginning by classifying the simple components of the object, and only later the composite objects. Second, although LABYRINTH uses the same evaluation function as CLASSIT, it requires an additional operator when classifying composite objects. This is used to generalize similar components when incorporating the composite instance into a concept. Since there are a number of applications that are best represented with complex instance descriptions, the ability to work with structured instances is an important advance for concept formation.

4.2 Concept formation and plan acquisition

In order to achieve its goals, an intelligent agent must be able to *plan*; that is, to order its actions in the world. DÆDALUS (Allen & Langley, 1990) is a means-ends planner that uses a concept hierarchy to retrieve and organize plan knowledge. Instances and concepts for DÆDALUS consists of three parts: a state description, the differences between this current state and a desired state, and the operator that should be applied. These descriptions include many attributes that contain relational information, such as **(on block1 block2)**. Unlike CLASSIT, DÆDALUS can work with this type of attribute, using partial matching to compare attributes between a concept and an instance.

Given a state description and a desired state (an instance without an operator), the performance task for DÆDALUS is to predict the missing operator. Like CLASSIT, the system finds this operator by sorting the instance through the concept hierarchy, and returning the operator stored with the best match. Initially, it operates as a simple means-end planner, applying an operator that reduces differences between the goal state and the current state. As it gains knowledge about successful plans, it operates more like a case-based planner, using its experience from previous plans to constrain operator selection. Finally, when the concept hierarchy includes abstract, internal nodes that summarize many experiences, the system acts as a schema-based planner, using more abstract knowledge to select operators.

4.3 Concept formation and motor learning

In order to carry out actions, ICARUS must include a component for motor control. The MÆANDER system represents and carries out primitive motor control operations, such as manipulating a limb or moving through the environment. Like other components of ICARUS, this system organizes and retrieves its knowledge in a concept hierarchy. However, unlike high-level planning, instances and concepts in this hierarchy are simple *motor schemas*: a sequence of locations and velocities of body parts that specify a movement.

MÆANDER uses OXBOW, a variation of CLASSIT, to learn to recognize and classify observed movements (Iba & Gennari, in press). Unlike instances in CLASSIT, movements are not simple attribute value lists, but are instead a restricted type of composite objects. A movement is made up of an ordered list of *states*, each of which includes a time stamp, together with the velocity and positions of all joints used in the movement. Different movements may have different numbers of states. Like Labyrinth, Oxbow incorporates a number of advances to support concept formation with this more complex type of instance.

5. Conclusion

Progress in a scientific field requires two types of research: uninhibited leaps into unexplored territory, and careful evaluation of a particular set of ideas. If there is only the latter type of work, then the field would miss the revolutionary new ideas of an Einstein. However, without careful testing and evaluation, there would be no one to repudiate the claims of creative charlatans.

Much as it might be more exciting to claim otherwise, this thesis falls mainly into the testing and evaluation side of machine learning. My system is not a radical breakthrough; instead, it verifies and expands on some known ideas for concept formation. I state this without apology for without this type of work, no science can sustain itself.

Having placed my work on the humble side of the fence, I should make clear that I am not resigned to forever plodding through endless tests and experiments with concept formation. My interest in Icarus, and in psychological phenomena such as attention, shows that my heart still yearns for the larger, more exciting goals of machine learning. A robot that can process perceptual information and learn to build up useful memory structures would be a very dramatic application of concept formation.

Yet even these 'engineering' breakthroughs are of smaller scope than the goals of cognitive psychology. If concept formation research can lead to an understanding of human learning abilities, then it will speed the development of systems capable of real human intelligence. However, understanding human learning is an extremely difficult task, and the ultimate research goals of cognitive psychology are dauntingly distant.

My practical approach is to tackle the problem from the far side of the fence – breaking it apart and carefully understanding the myriad issues piece by painstaking piece. This slow, arduous work includes the research in this thesis: understanding the behavior of algorithms under different conditions, comparing variations of the system, and systematic experimentation with a variety of domains. Although I have pursued some rather humble tasks in this dissertation, it is the higher level goals from cognitive psychology, robotics, or artificial intelligence that drive my research. I hope that in some small way, Classit will someday lead to a breakthrough in the study of intelligence.

References

Aha, D. W., & Kibler, D. (1989). Noise tolerant instance-based learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 794–799). Detroit, MI: Morgan Kaufmann.

Aha, D. W., & McNulty, D. M. (1989). Learning relative weights for instance-based concept descriptions. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 530–537). Ann Arbor, MI: Lawrence Erlbaum.

Aldenderfer, M., & Blashfield, R. (1984). *Cluster analysis.* Beverly Hills: Sage Publications.

Allen, J. A., & Langley, P. (1990). *The acquisition, organization, and use of plan memory.* Unpublished manuscript, NASA Ames research Center, Moffett Field, CA.

Anderberg, M. (1973). *Cluster analysis for applications.* New York: Academic Press.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge: Harvard University Press.

Anderson, J. R. (1988). The place of cognitive architectures in a rational analysis. *Proceedings of the Tenth Annual Cognitive Science Conference* (pp. 1–10). Montreal, CA: Lawrence Erlbaum.

Anderson, J. R. (in press). The place of rational analysis in a cognitive architecture. In K.Van Lehn (Ed.), *Architectures for intelligence.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., & Matessa, M. (1990) A rational analysis of categorization. *Proceedings of the Seventh International Workshop on Machine Learning* (pp. 76–84). Austin, TX: Morgan Kaufmann.

Billman, D., & Heit, E. (1988). Observational learning from internal feedback: A simulation of an adaptive learning method. *Cognitive Science, 12,* 587–625.

Barsalou, L. (1987). The instability of graded structure: implications for the nature of concepts. In U. Neisser (Ed.), *Concepts and conceptual development: Ecological and intellectual factors in categorization.* New York: Cambridge University Press.

Buntine, W., & Niblett, T. (1990). *A further comparison of splitting rules for decision-tree induction.* Unpublished manuscript, Turing Institute, Glasgow, U. K.

Carbonell, J. G. (1989). Introduction: Pardigms for machine learning. *Artificial Intelligence, 40,* 1–9.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AUTOCLASS: A Bayesian classification system. *Proceedings of the Fifth Inter-national Conference on Machine Learning* (pp. 54–64). Ann Arbor, MI: Morgan Kaufmann.

Corter, J., Gluck, M., & Bower, G. (1988). Basic levels in hierarchically structured categories. *Proceedings of the Tenth Annual Cognitive Science Conference* (pp. 118–124). Montreal, CA:Lawrence Erlbaum.

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). *International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology, 64*, 304–310.

Dietterich, T. G. (1990). Machine learning. *Annual Review of Computer Science, 4*, 255–306.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis.* New York: John Wiley and Sons.

Everitt, B. (1979). Unresolved problems in cluster analysis. *Biometrics, 35*, 169–181.

Everitt, B. (1980). *Cluster analysis.* London: Heinemann Educational.

Evett, I. W. & Speihler, E. J. (1987). *Rule induction in forensic science.* Technical Note, Home Office Forensic Science Service, Reading, UK.

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought.* New York: McGraw–Hill.

Fisher, D. (1984). *A hierarchical conceptual clustering algorithm* (Technical Report 85-21). Irvine: University of California, Department of Information & Computer Science.

Fisher, D. (1987a). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139–172.

Fisher, D. (1987b). *Knowledge acquisition via incremental conceptual clustering.* Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.

Fisher, D., & Langley, P., (1986). Methods of conceptual clustering and their relation to numerical taxonomy. In W. Gale (Ed.), *Artificial intelligence and statistics.* Reading, MA: Addison Wesley.

Fisher, D. H., & Langley, P. (in press). The structure and formation of natural categories. In G. H. Bower (Ed.), *The psychology of learning and motivation: Advances in research and theory* (Vol. 26). Cambridge, MA: Academic Press.

Fried, L. S., & Holyoak, K. J. (1984). Induction of category distributions: A framework for classification learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 10*, 234–257.

Gennari, J. H. (1989). Focused concept formation. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 379–382). Ithaca, NY: Morgan Kaufmann.

Gennari, J. H., Langley, P., & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence, 40*, 11–61.

Gluck, M., & Corter, J. (1985). Information, uncertainty and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.

Hamming, R. W. (1980). *Coding and information theory.* Englewood Clifs, NJ: Prentice Hall.

Hand, D. J. (1981). *Discrimination and classification.* New York: John Wiley & Sons.

Hanson, S. J., & Bauer, M. (1989). Conceptual clustering, categorization, and polymorphy. *Machine Learning, 3*, 343–372.

Haussler, D. (1987). Bias, version spaces and Valiant's learning framework. *Pro-ceedings of the Fourth International Workshop on Machine Learning* (pp. 324–336). Irvine, CA: Morgan Kaufmann.

Iba, W., & Gennari, J. H. (in press). Learning to recognize movements. In D. Fisher & M. Pazzani (Eds.), *Computational approaches to concept formation.* San Mateo, CA: Morgan Kaufmann.

Iba, W., Wogulis, J., & Langley, P. (1988). Trading off simplicity and coverage in incremental concept learning. *Proceedings of the Fifth International Workshop on Machine Learning* (pp. 73–79). Ann Arbor, MI: Morgan Kaufmann.

Jaynes, E. T. (1986). Bayesian methods: General background. In J. H. Justice (Ed.), *Maximum entropy and Bayesian methods in applied statistics* (pp. 1–25). Cambridge, MA: Cambridge University Press.

LaBerge, D., & Brown, V. (1989). Theory of attentional operations in shape identification. *Psychological Review, 96*, 101–124.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.

Langley, P. (1988). Machine learning as an experimental science. *Machine Learning, 3*, 5–8.

Langley, P., Gennari, J., & Iba, W. (1987). Hill climbing theories of learning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 312–323). Irvine, CA: Morgan Kaufmann.

Langley, P., Thompson, K., Iba, W., Gennari, J. H., & Allen, J. A. (in press). An integrated cognitive architecture for autonomous agents. In W. Van De Velde (Ed.), *Representation and Learning in Autonomous Agents.* Amsterdam: North Holland.

Lebowitz, M. (1985). Categorizing numeric information for generalization. *Cognitive Science, 9*, 285–309.

Lebowitz, M. (1986). Concept learning in a rich input domain: Generalization based memory. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). San Mateo, CA: Morgan Kaufmann.

McNulty, D. M. (1988). *Recognition by directed attention to recursively partitioned images* (Technical Report 88-08). Irvine: University of California, Department of Information & Computer Science.

MacNaughton-Smith, P., Williams, W. T., Dale, M. B., & Mockett L. G. (1964). Dissimilarity analysis. *Nature, 202*, 1034–1035.

Martin, J. D. (1989). Reducing redundant learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 396–399). Ithaca, NY: Morgan Kaufmann.

Martin, J. D. (1990). Learning overlapping categories. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 166–173). Cambridge, MA: Lawrence Erlbaum.

Mervis, C., & Rosch, E. (1981). Categorization of natural objects. *Annual Review of Psychology, 32*, 89–115.

Michalski, R. S., & Stepp, R. (1983a). Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Michalski, R. S., & Stepp, R. (1983b). Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 5*, 396–409.

Mingers, J. (1989). An empirical comparison of selection measures for decision tree induction. *Machine Learning, 3*, 319–342.

Nordhausen, B. E. (1989) *A computational framework for empirical discovery.* Doctoral dissertation, Department of Information & Computer Science, Univer-sity of California, Irvine.

Nosofsky, R. M. (1986). Attention, similarity, and the identification- categorization relationship. *Journal of Experimental Psychology: General, 115*, 39–57.

Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science, 12*, 410–430.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Reich, Y. (1989). *Combining nominal and continuous properties in an incremental learning system for design.* Unpublished manuscript, Engineering Design Research Center, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

Romesburg, H. C. (1984). *Cluster analysis for researchers.* Belmont, CA: Lifetime Learning Publications.

Schlimmer, J. C., & Granger, R. (1986). Beyond incremental processing: Tracking concept drift. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 502–507). Philadelphia, PA: Morgan Kaufmann.

Schneider, W., Dumas, S. T., & Shiffrin, R. M. (1984). Automatic and control processing and attention. In R. Parasuraman & D. R. Davies (Eds.), *Varieties of attention.* San Diego, CA: Academic Press.

Silverstein, G., & Pazzani, M. (1990). *Maxweb: A semi-supervised conceptual clustering algorithm.* Unpublished manuscript, Department of Information & Computer Science, University of California, Irvine.

Smith, E. E., & Medin, D. L. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Stepp, R. (1987). Concepts in conceptual clustering. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 211–213). Milan, Italy: Morgan Kaufmann.

Tan, M., & Schlimmer, J. C. (1989). Cost-sensitive concept learning of sensor use in approach and recognition. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 371–374). Ithaca, NY: Morgan Kaufmann.

Thompson, K., & Langley, P. (in press). Concept formation in structured domains. In D. H. Fisher & M. Pazzani (Eds.) *Computational approaches to concept formation.* San Mateo, CA: Morgan Kaufmann.

Treisman, A. M. (1969). Strategies and models of selective attention. *Psychological Review, 76,* 282–299.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27,* 1134–1142.

Wallace, R. S. (1989). *Finding natural clusters through entropy minimization.* Doctoral dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision.* New York: McGraw–Hill.

Appendix A

Sample instances from the five natural databases.

1. The voting database

The voting database consists of instances with 17 binary attributes, with the last attribute indicating political party (the class attribute). Chapter 4 uses a subset of this database with no missing information; Chapter 6 shows results with missing information.

Complete information:

```
y y y n n y y y y n n n n n y y dem
n n n y y n y y y y n y y y n y rep
n y y n y y y n y y y n y y n y dem
y n y n n n n y y y n n n n y y dem
n n n y y y n n n y n y y y n n rep
y y n y n n y y y n y n n y n y dem
n n n y y n n n n n n y y y n y rep
y n n n y y n n n n y y n y n y dem
y y n y y y n n n y n y y y n n rep
n n y n n y y y y y n n n y n y dem
y y y n n n y y y n n n n n y y dem
n n y n n y y y y y y n y y n y dem
```

Missing information:

```
y ? y n n n y y y n ? n n n y ? dem
y n y n n n y y y n y n n n y ? dem
n y n y y y n n n n n y y ? n y rep
n n y n n y y y y y n y n y y ? dem
y y y n n n y y ? y n n n n y ? dem
n n n y n y y ? y n n y y y n y rep
n n n y y n n n n n n y n y ? y rep
y n y n n ? y y y n ? ? n ? ? ? dem
y n y n n n y y y n n n n n y ? dem
y n y n n n y y y n n n n n y ? dem
```

2. The glass database

The glass database consists of 9 numeric attributes and a symbolic class attribute that indicates the window type. The numeric attributes are the refractive index, and percentage of weight of various oxides. These oxides are: sodium, magnesium, aluminum, silicon, potassium, calcium, barium and iron (attributes #2 – #9). For all experiments I predicted the class attribute as a binary attribute with values of 'window glass' (**winfl, win**) and 'not window glass' (**table, head, cont**).

1.51761	12.81	3.54	1.23	73.24	0.58	8.39	0.00	0.00	**winfl**
1.51779	13.21	3.39	1.33	72.76	0.59	8.59	0.00	0.00	**winfl**
1.51806	13.00	3.80	1.08	73.07	0.56	8.38	0.00	0.12	**win**
1.52315	13.44	3.34	1.23	72.38	0.60	8.83	0.00	0.00	**head**
1.51888	14.99	0.78	1.74	72.50	0.00	9.95	0.00	0.00	**table**
1.51590	13.24	3.34	1.47	73.10	0.39	8.22	0.00	0.00	**win**
1.52369	13.44	0.00	1.58	72.22	0.32	12.24	0.00	0.00	**cont**
1.51824	12.87	3.48	1.29	72.95	0.60	8.43	0.00	0.00	**winfl**
1.51778	13.21	2.81	1.29	72.98	0.51	9.02	0.00	0.09	**winfl**
1.51769	12.45	2.71	1.29	73.70	0.56	9.06	0.00	0.24	**winfl**
1.51653	11.95	0.00	1.19	75.18	2.70	8.93	0.00	0.00	**head**
1.51838	14.32	3.26	2.22	71.25	1.46	5.79	1.63	0.00	**head**
1.51594	13.09	3.52	1.55	72.87	0.68	8.05	0.00	0.09	**win**
1.51751	12.81	3.57	1.35	73.02	0.62	8.59	0.00	0.00	**winfl**
1.51905	13.60	3.62	1.11	72.64	0.14	8.76	0.00	0.00	**winfl**

3. The heart-disease database

This database is used in both mixed format, with 8 symbolic and 6 numeric attributes, and as a 'numeric' database, with 13 numeric attributes and a symbolic class attribute.

In order, the attributes are age; sex; chest pain type (angina, abnang, notang, asympt) Resting blood pressure; cholesterol; fasting blood sugar less than 120 (true or false); resting ecg (norm, abn, hyper); max heart rate; exercise induced angina (true or false); oldpeak; slope (up, flat, down) number of vessels colored; thal (norm, fixed, reverse). Finally, the patient is either healthy (0) or with some degree of heart-disease (1, 2, 3).

As a mixed database:

```
57.0  fem asympt 120.0 354.0 fal  norm 163.0 true 0.6   up 0.0 norm fine
63.0 male asympt 130.0 254.0 fal   hyp 147.0 fal  1.4 flat 1.0  rev sick
53.0 male asympt 140.0 203.0 true  hyp 155.0 true 3.1 down 0.0  rev sick
57.0 male asympt 140.0 192.0 fal  norm 148.0 fal  0.4 flat 0.0  fix fine
56.0  fem abnang 140.0 294.0 fal   hyp 153.0 fal  1.3 flat 0.0 norm fine
56.0 male notang 130.0 256.0 true  hyp 142.0 true 0.6 flat 1.0  fix sick
44.0 male abnang 120.0 263.0 fal  norm 173.0 fal  0.0   up 0.0  rev fine
49.0 male abnang 130.0 266.0 fal  norm 171.0 fal  0.6   up 0.0 norm fine
58.0  fem angina 150.0 283.0 true  hyp 162.0 fal  1.0   up 0.0 norm fine
58.0 male abnang 120.0 284.0 fal   hyp 160.0 fal  1.8 flat 0.0 norm sick
```

As a numeric database:

```
48.0  1.0  2.0 130.0 245.0  0.0  2.0 180.0  0.0  0.2  2.0  0.0  3.0 fine
52.0  1.0  2.0 120.0 325.0  0.0  0.0 172.0  0.0  0.2  1.0  0.0  3.0 fine
53.0  0.0  4.0 138.0 234.0  0.0  2.0 160.0  0.0  0.0  1.0  0.0  3.0 fine
56.0  1.0  4.0 125.0 249.0  1.0  2.0 144.0  1.0  1.2  2.0  1.0  3.0 sick
55.0  1.0  4.0 140.0 217.0  0.0  0.0 111.0  1.0  5.6  3.0  0.0  7.0 sick
45.0  0.0  2.0 130.0 234.0  0.0  2.0 175.0  0.0  0.6  2.0  0.0  3.0 fine
56.0  0.0  4.0 200.0 288.0  1.0  2.0 133.0  1.0  4.0  3.0  2.0  7.0 sick
54.0  1.0  4.0 110.0 239.0  0.0  0.0 126.0  1.0  2.8  2.0  1.0  7.0 sick
62.0  0.0  4.0 124.0 209.0  0.0  0.0 163.0  0.0  0.0  1.0  0.0  3.0 fine
43.0  0.0  3.0 122.0 213.0  0.0  0.0 165.0  0.0  0.2  2.0  0.0  3.0 fine
41.0  1.0  3.0 112.0 250.0  0.0  0.0 179.0  0.0  0.0  1.0  0.0  3.0 fine
64.0  1.0  4.0 120.0 246.0  0.0  2.0  96.0  1.0  2.2  3.0  1.0  3.0 sick
```

4. The auto insurance database

This database contains 16 numeric and 8 symbolic attributes about vehicles as follows: risk factor; losses; vehicle make; aspiration; doors; body; drive; wheel-base; length; width; height; weight; engine-type; cylinders; engine-size; fuel-system; bore; stroke; compression; horsepower; rpm; city-mpg; hwy-mpg; price.

```
  { 0  78  honda  std four wagon fwd  96.50 157.10 63.90 58.30 2024  ohc four
92 1bbl 2.92 3.41  9.20  76 6000 30 34  7295}
  { 0 118  mazda  std four sedan rwd 104.90 175.00 66.10 54.40 2670  ohc four
140 mpfi 3.76 3.16  8.00 120 5000 19 27 18280}
  { 0 106  honda  std  two hatch fwd  96.50 167.50 65.20 53.30 2289  ohc four
110 1bbl 3.15 3.58  9.00  86 5800 27 33  9095}
  { 0  85  honda  std four sedan fwd  96.50 175.40 65.20 54.10 2304  ohc four
110 1bbl 3.15 3.58  9.00  86 5800 27 33  8845}
  { 1 158   audi  std four sedan fwd 105.80 192.70 71.40 55.70 2844  ohc five
136 mpfi 3.19 3.40  8.50 110 5500 19 25 17710}
  { 1 118  dodge  std  two hatch fwd  93.70 157.30 63.80 50.80 1876  ohc four
90 2bbl 2.97 3.23  9.41  68 5500 37 41  5572}
  {-1  74  volvo  std four wagon rwd 104.30 188.80 67.20 57.50 3042  ohc four
141 mpfi 3.78 3.15  9.50 114 5400 24 28 16515}
  {-1  74  volvo turb four wagon rwd 104.30 188.80 67.20 57.50 3157  ohc four
130 mpfi 3.62 3.15  7.50 162 5100 17 22 18950 }
  { 3 194 nissan  std  two hatch rwd  91.30 170.70 67.90 49.70 3071 ohcv  six
181 mpfi 3.43 3.27  9.00 160 5200 19 25 17199}
  { 2 134 toyota  std  two  hard rwd  98.40 176.20 65.60 52.00 2536  ohc four
146 mpfi 3.62 3.50  9.30 116 4800 24 30  9639}
```

5. *The LRS star database*

As used by CLASSIT, this database contains class information, fluxes from 44 blue-band channel wavelengths, followed by fluxes from 49 red-band channel wavelengths. The class information is a number, but without more information about its semantics, it must be treated as a symbolic attribute. The original database also includes the astronomical position of each star (right-ascension and declination), and scaling factors, but these were not used for clustering.

Here are two sample instances:

```
{ 42 14956.77 14945.297 15342.541 15137.868 14112.801 13309.645 12771.742
12148.88 11471.069 10313.045 9171.77 8754.345 9037.418 8396.656 7866.022
7764.041 7451.989 7121.642 7134.8906 7216.566 7286.6123 7008.154 7250.06
7255.193 6803.421 6680.2197 6833.8926 7070.9893 6759.4443 6332.8643 5707.8613
5564.8525 5660.3535 5347.3003 4972.568 4859.6934 4651.785 4183.7456 4631.292
4566.8843 4155.619 3567.601 3673.3662 3694.1016 7388.07 7001.4297 6786.5522
6801.809 5776.417 5638.231 4652.851 4446.2773 3924.505 3831.324 3419.6096
3186.3337 3059.786 3016.619 2581.541 2820.2068 2548.556 2055.0981 2015.2505
2027.0983 1905.528 1825.6473 1562.3181 1592.5006 1349.1791 1536.1202 1374.5115
1294.6177 1519.9097 1141.4951 1108.2911 1151.7996 888.45044 1077.0446 726.9725
1224.0869 1177.5319 1114.2985 890.9651 857.74756 937.93176 938.0656 843.13947
888.2985 755.0131 958.60065 887.0177 877.13837 675.47327}

{ 28 7729.6216 7749.091 7315.261 8026.9287 8011.0083 8054.817 8796.448
9337.731 9981.371 10953.328 12137.065 12745.263 12726.759 12379.991 12408.432
11858.486 11331.301 11162.265 10404.891 10058.25 9995.856 9312.437 8796.73
8435.943 8136.2837 7550.9854 7085.078 6742.826 6235.732 5833.817 5515.762
4796.954 4818.579 4685.31 4105.8076 4013.773 3863.481 3555.4517 3615.5098
3615.89 3721.4443 3360.8691 3389.6987 3064.67 9799.441 8831.7295 7338.2847
6318.171 5160.2827 4532.556 3956.0564 3317.087 3184.103 2955.9543 2834.441
2670.3384 2408.2185 2485.7952 2229.438 2511.7346 2353.5068 2145.5713 2356.9065
2400.5754 2361.8445 2391.3555 2337.0442 2208.8572 2431.7185 2235.362 2039.0265
2045.937 1946.2815 1975.9698 1975.8325 1739.8008 1630.2704 1702.6637 1576.1118
1611.5867 1357.1617 1355.7169 1460.8734 1374.1876 1242.0381 1124.1136 1262.0094
1102.8966 1243.6593 1118.6782 1152.8777 1006.6016 782.1728}
```

Appendix B

Data Generators

This appendix includes descriptions of the data generators used to build all of the artificial data sets used in the dissertation. For each generator, I include references to the chapter and section that used the artificial domain created by that generator.

These generators were written in C, and although I do not give complete program listings, there should be enough detail to easily re-create the data sets. Each generator includes a set of class definitions, where there is a function that prints a single instance from that class. Unless otherwise noted, all classes are chosen with equal probability. Numeric attributes values are usually created with a call to bellrand(μ,σ), where μ is the mean and σ is the standard deviation. 'bellrand' uses a standard polynomial approximation to produce values with a normal distribution.

1. The class complexity generators

Chapter 4, Section 3.1 describes an experiment with three artificial data sets. There are three generators for these: 'makebinary' for the binary domain, 'make-six' for the six-class domain, and 'make-sub' for the subset domain. All three generators create instances with nine numeric attributes.

Makebinary has two classes:

```
/*-----------------------------------------------------------------------*/
make-class1(name)
char name[];
{
    float bellrand();
    printf("%6.2f %6.2f %6.2f ",
        bellrand(90.0, 2.0), bellrand(14.0, 1.0), bellrand(40.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
        bellrand(50.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
       bellrand(20.0, 3.0), bellrand(20.0, 3.0), bellrand(20.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class1 */

make-class2(name)
char name[];
{
    float bellrand();
```

```
     printf("%6.2f %6.2f %6.2f ",
         bellrand(90.0, 2.0), bellrand(14.0, 1.0), bellrand(40.0, 2.0));
     printf("%6.2f %6.2f %6.2f ",
         bellrand(50.0, 5.0), bellrand(90.0, 10.0),  bellrand(120.0, 20.0));
     printf("%6.2f %6.2f %6.2f ",
         bellrand(20.0, 3.0), bellrand(20.0, 3.0), bellrand(20.0, 3.0));
     printf(" %s\n", name);
} /* end of make-class2 */
/*-----------------------------------------------------------------------*/
```

Make-six has six classes:

```
/*-----------------------------------------------------------------------*/
make-class1(name)
char name[];
{
    float          bellrand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(100.0, 2.0), bellrand(10.0, 1.0), bellrand(35.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(30.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
         bellrand(80.0, 3.0), bellrand(80.0, 3.0), bellrand(80.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class1 */

make-class2(name)
char name[];
{
    float          bellrand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(90.0, 2.0), bellrand(14.0, 1.0), bellrand(40.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(50.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
         bellrand(20.0, 3.0), bellrand(20.0, 3.0), bellrand(20.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class2 */

make-class3(name)
char name[];
{
    float          bellrand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(80.0, 2.0), bellrand(18.0, 1.0), bellrand(45.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(70.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
         bellrand(100.0, 3.0), bellrand(100.0, 3.0), bellrand(100.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class3 */

make-class4(name)
char name[];
{
    float       bellrand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(70.0, 2.0), bellrand(22.0, 1.0), bellrand(50.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(90.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
         bellrand(40.0, 3.0), bellrand(40.0, 3.0), bellrand(40.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class4 */

make-class5(name)
char name[];
{
    float       bellrand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(60.0, 2.0), bellrand(26.0, 1.0), bellrand(55.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(110.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
         bellrand(60.0, 3.0), bellrand(60.0, 3.0), bellrand(60.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class5 */

make-class6(name)
char name[];
{
    float       bellrand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(50.0, 2.0), bellrand(30.0, 1.0), bellrand(60.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(130.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
          bellrand(90.0, 3.0), bellrand(90.0, 3.0), bellrand(90.0, 3.0));
    printf(" %s\n", name);
} /* end of make-class6 */
/*-----------------------------------------------------------------------*/
```

Make-sub has two classes, each of which has three sub-classes:

```
/*-----------------------------------------------------------------------*/
make-class1(name)
char name[];
{
    float    bellrand();
    int         rand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(100.0, 2.0), bellrand(20.0, 2.0), bellrand(20.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(50.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    switch (rand(3)) {
        case 1: printf("%6.2f %6.2f %6.2f ",
                 bellrand(80.0, 3.0), bellrand(80.0, 3.0), bellrand(80.0, 3.0));
                printf(" %s1\n", name); break;
        case 2: printf("%6.2f %6.2f %6.2f ",
                 bellrand(20.0, 3.0), bellrand(20.0, 3.0), bellrand(20.0, 3.0));
                printf(" %s2\n", name); break;
        case 3: printf("%6.2f %6.2f %6.2f ",
                 bellrand(100.0, 3.0),bellrand(100.0, 3.0),bellrand(100.0, 3.0));
                printf(" %s3\n", name); break; }
} /* end of make-class1 */

make-class2(name)
char name[];
{
    float    bellrand();
    int         rand();
    printf("%6.2f %6.2f %6.2f ",
            bellrand(120.0, 2.0), bellrand(60.0, 2.0), bellrand(60.0, 2.0));
    printf("%6.2f %6.2f %6.2f ",
            bellrand(70.0, 5.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    switch (rand(3)) {
        case 1: printf("%6.2f %6.2f %6.2f ",
                 bellrand(40.0, 3.0), bellrand(40.0, 3.0), bellrand(40.0, 3.0));
```

```
               printf(" %s1\n", name); break;
        case 2: printf("%6.2f %6.2f %6.2f ",
              bellrand(60.0, 3.0), bellrand(60.0, 3.0), bellrand(60.0, 3.0));
               printf(" %s2\n", name); break;
        case 3: printf("%6.2f %6.2f %6.2f ",
              bellrand(90.0, 3.0), bellrand(90.0, 3.0), bellrand(90.0, 3.0));
               printf(" %s3\n", name); break; }
} /* end of make-class2 */
/*-----------------------------------------------------------------------*/
```

2. The irrelevant attribute generator

In Chapter 4, Section 3.2, I describe an experiment with successively more and more irrelevant attributes. Below, I list the generator for this domain. This program produces instances with four relevant attributes and either 0, 4, 8 or 16 irrelevant attributes, depending on the user's input (the parameter 'nirr'). There are four classes, and all attributes are numeric.

```
/*-----------------------------------------------------------------------*/
make-class1(name, nirr)
int nirr;
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f %6.2f", bellrand(20.0, 3.0),
          bellrand( 6.0, 1.0), bellrand(12.0, 2.0), bellrand(60.0, 4.0));
    switch (nirr) {
      case 0:  break;
      case 4:  printirr(1); break;
      case 8:  printirr(2); break;
      case 16: printirr(4); break;
    }
    printf(" %s\n", name);
} /* end of make-class1 */

make-class2(name, nirr)
int nirr;
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f %6.2f", bellrand(40.0, 3.0),
          bellrand( 9.0, 1.0), bellrand(22.0, 2.0), bellrand(80.0, 4.0));
    switch (nirr) {
      case 0:  break;
```

```
        case 4:  printirr(1); break;
        case 8:  printirr(2); break;
        case 16: printirr(4); break;
      }
    printf(" %s\n", name);
  } /* end of make-class2 */

  make-class3(name, nirr)
  int nirr;
  char name[];
  {
      float    bellrand();
      printf("%5.2f %5.2f %5.2f %6.2f", bellrand(60.0, 3.0),
            bellrand(12.0, 1.0), bellrand(32.0, 2.0), bellrand(100.0, 4.0));
      switch (nirr) {
        case 0:  break;
        case 4:  printirr(1); break;
        case 8:  printirr(2); break;
        case 16: printirr(4); break;
      }
      printf(" %s\n", name);
  } /* end of make-class3 */

  make-class4(name, nirr)
  int nirr;
  char name[];
  {
      float    bellrand();
      printf("%5.2f %5.2f %5.2f %6.2f", bellrand(80.0, 3.0),
            bellrand(15.0, 1.0), bellrand(42.0, 2.0), bellrand(120.0, 4.0));
      switch (nirr) {
        case 0:  break;
        case 4:  printirr(1); break;
        case 8:  printirr(2); break;
        case 16: printirr(4); break;
      }
      printf(" %s\n", name);
  } /* end of make-class4 */

  /*-----------------------------------------------------------------------*/
  printirr(cnt)
  int cnt;
```

```
{
   int i;
   for (i=0; i< cnt; i++)
     printf(" %6.2f %6.2f %5.2f %5.2f", bellrand(90.0, 8.0),
          bellrand(100.0, 5.0), bellrand(15.0, 1.0), bellrand(40.0, 3.0));
}
/*-----------------------------------------------------------------------*/
```

3. The numeric noise generator

In Section 3.3 of Chapter 4 and in the parametric studies of Chapter 5 (Sections 1.2 and 1.3), I use artificial numeric domains where the amount of noise varies. As described in the dissertation, this is accomplished by changing the standard deviation of each class. For this generator, this is done with the parameter 'sig':

```
make-class1(name,sig)
char name[];
float sig;
{
    float    bellrand();
    printf("%6.2f %6.2f %6.2f %6.2f ", bellrand(100.0, sig),
            bellrand(120.0, sig), bellrand(120.0, sig), bellrand(120.0, sig));
    printf(" %s\n", name);
} /* end of make-class1 */

make-class2(name, sig)
char name[];
float sig;
{
    float    bellrand();
    printf("%6.2f %6.2f %6.2f %6.2f ", bellrand(100.0, sig),
            bellrand(140.0, sig), bellrand(140.0, sig), bellrand(140.0, sig));
    printf(" %s\n", name);
} /* end of make-class1 */

make-class3(name, sig)
char name[];
float sig;
{
    float    bellrand();
    printf("%6.2f %6.2f %6.2f %6.2f ", bellrand(100.0, sig),
            bellrand(160.0, sig), bellrand(160.0, sig), bellrand(160.0, sig));
    printf(" %s\n", name);
} /* end of make-class1 */

make-class4(name, sig)
char name[];
float sig;
{
    float    bellrand();
    printf("%6.2f %6.2f %6.2f %6.2f ", bellrand(100.0, sig),
            bellrand(180.0, sig), bellrand(180.0, sig), bellrand(180.0, sig));
    printf(" %s\n", name);
} /* end of make-class4 */
```

4. The generators for mixed domains

In Section 4 of Chapter 4, I describe "Mixed-NTL" and "Mixed-STL", a pair of artificial domains with both numeric and symbolic attributes. The generators for these domains each produce four classes of instances with six numeric and six symbolic attributes. However, as seen in Figure 19, these six classes are structured differently for the two domains. For Mixed-NTL, the attribute values are such that the system should build three classes divided by the numeric attributes, each with two subclasses distinguished by the symbolic attributes. For Mixed-STL, the situation is reversed: the system should build two main classes divided by the symbolic attributes, each with three subclasses distinguished by the numeric attributes.

Both domains also include six irrelevant attributes: these are the first three numeric attributes, and the first three symbolic attributes. For the symbolic attributes, irrelevant values are generated by the function "pickaval", while relevant attributes are generated by the "chosen1", "chosen2" and "chosen3" functions.

The Mixed-STL data generator:

```
makeA1(name)
char name[];
{
    float    bellrand();
    char      *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
        bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
        bellrand(10.0, 2.0), bellrand(100.0, 5.0), bellrand(60.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
        pickaval(4), pickaval(5), pickaval(2),
        chosen1(11), chosen2(11), chosen3(11));
    printf(" %s\n", name);
} /* end of makeA1 */

makeA2(name)
```

```
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
        bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
        bellrand(20.0, 2.0), bellrand(150.0, 5.0), bellrand(50.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
       pickaval(4), pickaval(5), pickaval(2),
       chosen1(12), chosen2(12), chosen3(12));
    printf(" %s\n", name);
} /* end of makeA2 */

makeA3(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
       bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
       bellrand(30.0, 2.0), bellrand(70.0, 5.0), bellrand(40.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
       pickaval(4), pickaval(5), pickaval(2),
       chosen1(13), chosen2(13), chosen3(13));
    printf(" %s\n", name);
} /* end of makeA3 */

makeB1(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
       bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
       bellrand(40.0, 2.0), bellrand(60.0, 5.0), bellrand(30.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
       pickaval(4), pickaval(5), pickaval(2),
       chosen1(21), chosen2(21), chosen3(21));
    printf(" %s\n", name);
} /* end of makeB1 */

makeB2(name)
```

```
char name[];
{
    float    bellrand();
    char      *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
      bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
      bellrand(50.0, 2.0), bellrand(200.0, 7.0), bellrand(20.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
      pickaval(4), pickaval(5), pickaval(2),
      chosen1(22), chosen2(22), chosen3(22));
    printf(" %s\n", name);
} /* end of makeB2 */

makeB3(name)
char name[];
{
    float    bellrand();
    char      *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
      bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
      bellrand(60.0, 2.0), bellrand(110.0, 5.0), bellrand(10.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
      pickaval(4), pickaval(5), pickaval(2),
      chosen1(23), chosen2(23), chosen3(23));
    printf(" %s\n", name);
} /* end of makeB3 */

/*---------------------------------------------------------------------*/
char *pickaval(n)
int n;
{
    switch (rand(n)) {
      case 1: return ("bull");
      case 2: return ("hogwash");
      case 3: return ("fiddle");
      case 4: return ("nuts");
      case 5: return ("foobar");
      case 6: return ("bananas"); }
}

/*---------------------------------------------------------------------*/
char *chosen1(class)
```

```
int class;
{
    switch (class) {
       case 11: case 12: case 13: return ("red");
       case 21: case 22: case 23: return ("green"); }
}

char *chosen2(class)
int class;
{
    switch (class) {
       case 11: case 12:
       case 13: if (rand(100) < 75) return ("red"); return ("black");
       case 21: case 22:
       case 23: if (rand(100) < 85) return ("green"); return ("red"); }
}

char *chosen3(class)
int class;
{
    switch (class) {
       case 11: if (rand(100) < 90) return ("burp"); return("squeal");
       case 12: if (rand(100) < 80) return ("burp"); return("belch");
       case 13: if (rand(100) < 75) return ("burp"); return("squeal");
       case 21: if (rand(100) < 70) return ("squeal"); return("burp");
       case 22: if (rand(100) < 65) return ("squeal"); return("moan");
       case 23: if (rand(100) < 85) return ("squeal"); return("burp"); }
}
/*----------------------------------------------------------------------*/
```

The Mixed-NTL data generator (This is also used in Chapter 6, Section 1.3):

```
makeA1(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
            bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
            bellrand(10.0, 2.0), bellrand(100.0, 5.0), bellrand(60.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
            pickaval(4), pickaval(5), pickaval(2),
            chosen1(11), chosen2(11), chosen3(11));
    printf(" %s\n", name);
} /* end of makeA1 */

makeA2(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
            bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
            bellrand(20.0, 2.0), bellrand(150.0, 5.0), bellrand(50.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
            pickaval(4), pickaval(5), pickaval(2),
            chosen1(12), chosen2(12), chosen3(12));
    printf(" %s\n", name);
} /* end of makeA2 */

makeA3(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
            bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
            bellrand(30.0, 2.0), bellrand(70.0, 5.0), bellrand(40.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
            pickaval(4), pickaval(5), pickaval(2),
            chosen1(13), chosen2(13), chosen3(13));
    printf(" %s\n", name);
} /* end of makeA3 */

makeB1(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
            bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
            bellrand(10.0, 2.0), bellrand(100.0, 5.0), bellrand(60.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
            pickaval(4), pickaval(5), pickaval(2),
            chosen1(21), chosen2(21), chosen3(21));
    printf(" %s\n", name);
} /* end of makeB1 */

makeB2(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
            bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
            bellrand(20.0, 2.0), bellrand(150.0, 5.0), bellrand(50.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
            pickaval(4), pickaval(5), pickaval(2),
            chosen1(22), chosen2(22), chosen3(22));
    printf(" %s\n", name);
} /* end of makeB2 */

makeB3(name)
char name[];
{
    float    bellrand();
    char       *pickaval(), *chosen1(), *chosen2(), *chosen3();
    printf("%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
            bellrand(30.0, 10.0), bellrand(15.0, 5.0), bellrand(25.0, 2.0),
            bellrand(30.0, 2.0), bellrand(70.0, 5.0), bellrand(40.0, 2.0));
    printf(" %7s %7s %7s %7s %7s %7s",
            pickaval(4), pickaval(5), pickaval(2),
            chosen1(23), chosen2(23), chosen3(23));
    printf(" %s\n", name);
} /* end of makeB3 */

/*----------------------------------------------------------------------*/
char *pickaval(n)
int n;
{
    switch (rand(n)) {
       case 1: return ("bull");
       case 2: return ("hogwash");
       case 3: return ("fiddle");
       case 4: return ("nuts");
       case 5: return ("foobar");
       case 6: return ("bananas"); }
```

```
}

/*-----------------------------------------------------------------------*/
char *chosen1(class)
int class;
{

  switch (class) {
  case 11: return ("red");
  case 21: return ("green");
  case 12: return ("white");
  case 22: return ("black");
  case 13: return ("blue");
  case 23: return ("yellow"); }
}

char *chosen2(class)
int class;
{
  switch (class) {
  case 11: if (rand(100) < 75) return ("red");
  case 21: if (rand(100) < 90) return ("green");
  case 12: if (rand(100) < 70) return ("white");
  case 22: if (rand(100) < 90) return ("black");
  case 13: if (rand(100) < 60) return ("blue");
  case 23: if (rand(100) < 85) return ("yellow");
  default: return ("garbage"); }
}

char *chosen3(class)
int class;
{
  switch (class) {
  case 11: if (rand(100) < 90) return ("burb");
  case 21: if (rand(100) < 65) return ("belch");
  case 12: if (rand(100) < 75) return ("fart");
  case 22: if (rand(100) < 90) return ("groan");
  case 13: if (rand(100) < 80) return ("moan");
  case 23: if (rand(100) < 85) return ("squeal");
  default: return ("scream"); }
}
/*-----------------------------------------------------------------------*/
```

5. The generator for testing overfitting

In Section 1.2 of Chapter 5, I describe a domain for testing overfitting and the recognition criterion. The class structure for this experiment is a complete binary tree of depth two: two main classes each with 2 subclasses. However, the generator simply chooses among the four subclasses with equal probability. For this generator, the first two and the last two subclasses should form top-level classes. As can be seen, there are six numeric attributes per instance.

```
/*----------------------------------------------------------------------*/
make-class1(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f ",
          bellrand(100.0, 4.0), bellrand(120.0, 4.0), bellrand(120.0, 4.0));
    printf("%5.2f %5.2f %5.2f ",
          bellrand(100.0, 4.0), bellrand(120.0, 4.0), bellrand(120.0, 4.0));
    printf(" %s\n", name);
} /* end of mtab1 */

make-class2(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f ",
          bellrand(100.0, 4.0), bellrand(140.0, 4.0), bellrand(140.0, 4.0));
    printf("%5.2f %5.2f %5.2f ",
          bellrand(100.0, 4.0), bellrand(120.0, 4.0), bellrand(120.0, 4.0));
    printf(" %s\n", name);
} /* end of mtab1 */

make-class3(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f ",
          bellrand(100.0, 4.0), bellrand(160.0, 4.0), bellrand(160.0, 4.0));
    printf("%5.2f %5.2f %5.2f ",
          bellrand(100.0, 4.0), bellrand(180.0, 4.0), bellrand(180.0, 4.0));
    printf(" %s\n", name);
} /* end of mtab1 */

make-class4(name)
```

```
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f ",
         bellrand(100.0, 4.0), bellrand(180.0, 4.0), bellrand(180.0, 4.0));
    printf("%5.2f %5.2f %5.2f ",
         bellrand(100.0, 4.0), bellrand(180.0, 4.0), bellrand(180.0, 4.0));
    printf(" %s\n", name);
} /* end of mtab4 */
/*----------------------------------------------------------------------*/
```

6. The generator for the gain ratio function

In Section 2.3 of Chapter 5, I use an artificial domain to compare the system's ability with two different evaluation functions: gain ratio against category utility. Although I describe this database in terms of flower colors and flower species, this is by analogy only; the actual domain is completely artificial, with eight numeric attributes per instance.

However, there are two very different types of attributes. The first four attributes are the 'color' attributes, those with lower magnitude values that Classit$_{CU}$ uses at the top level of its hierarchy. The second four attributes are the species attributes, those with higher magnitude values that Classit$_{GR}$ uses first. This two sets of attributes suggest conflicting class hierarchies, as seen in Figure 31.

As usual, the generator simply has a set of eight sub-classes, and chooses among them with equal probability.

```
/*----------------------------------------------------------------------*/
makeA1(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
       bellrand(3.0,0.6),bellrand(2.0,0.6),bellrand(3.0,0.6),bellrand(2.0,0.6),
       bellrand(150.0, 10.0), bellrand(650.0, 10.0),
       bellrand(150.0, 10.0), bellrand(650.0, 10.0));
    printf(" %s\n", name);
} /* end of makeA1 */

makeA2(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
       bellrand(3.0,0.6),bellrand(2.0,0.6),bellrand(3.0,0.6),bellrand(2.0,0.6),
        bellrand(400.0, 10.0), bellrand(900.0, 10.0),
        bellrand(400.0, 10.0), bellrand(900.0, 10.0));
     printf(" %s\n", name);
} /* end of mtab1 */

makeB1(name)
char name[];
{
     float    bellrand();
     printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
        bellrand(5.0,0.6),bellrand(4.0,0.6),bellrand(5.0,0.6),bellrand(4.0,0.6),
        bellrand(180.0, 10.0), bellrand(680.0, 10.0),
        bellrand(180.0, 10.0), bellrand(680.0, 10.0));
     printf(" %s\n", name);
} /* end of mtab1 */

makeB2(name)
char name[];
{
     float    bellrand();
     printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
        bellrand(5.0,0.6),bellrand(4.0,0.6),bellrand(5.0,0.6),bellrand(4.0,0.6),
        bellrand(430.0, 10.0), bellrand(930.0, 10.0),
        bellrand(430.0, 10.0), bellrand(930.0, 10.0));
     printf(" %s\n", name);
} /* end of mtab1 */

makeX1(name)
char name[];
{
     float    bellrand();
     printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
        bellrand(16.0,0.6),bellrand(15.0,0.6),bellrand(16.0,0.6),bellrand(15.0,0.6),
        bellrand(150.0, 10.0), bellrand(650.0, 10.0),
        bellrand(150.0, 10.0), bellrand(650.0, 10.0));
     printf(" %s\n", name);
} /* end of mtab1 */

makeX2(name)
char name[];
{
     float    bellrand();
```

```
    printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
       bellrand(16.0,0.6),bellrand(15.0,0.6),bellrand(16.0,0.6),bellrand(15.0,0.6),
       bellrand(400.0, 10.0), bellrand(900.0, 10.0),
       bellrand(400.0, 10.0), bellrand(900.0, 10.0));
    printf(" %s\n", name);
} /* end of mtab2 */

makeY1(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
       bellrand(14.0,0.6),bellrand(13.0,0.6),bellrand(14.0,0.6),bellrand(13.0,0.6),
       bellrand(180.0, 10.0), bellrand(680.0, 10.0),
       bellrand(180.0, 10.0), bellrand(680.0, 10.0));
    printf(" %s\n", name);
} /* end of mtab1 */

makeY2(name)
char name[];
{
    float    bellrand();
    printf("%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f ",
       bellrand(14.0,0.6),bellrand(13.0,0.6),bellrand(14.0,0.6),bellrand(13.0,0.6),
       bellrand(430.0, 10.0), bellrand(930.0, 10.0),
       bellrand(430.0, 10.0), bellrand(930.0, 10.0));
    printf(" %s\n", name);
} /* end of mtab3 */
/*----------------------------------------------------------------------*/
```

7. *The generator for comparing algorithms*

In Section 3 of Chapter 5, I compare CLASSIT against an agglomerative algorithm and against an iterative optimization method. The artificial domain I use for these experiments includes 4 classes, 2 of which (class1 and class3) are further divided into subclasses. Attributes #3 through #5 are irrelevant, and attributes #7 through #9 indicate the sub-class divisions. Attribute #9 is used for the prediction task.

```
/*----------------------------------------------------------------------*/
make-class1(name)
char name[];
{
    float    bellrand();
    int         rand();
    printf("%6.2f %6.2f %6.2f ",
      bellrand(100.0, 5.0), bellrand(120.0, 5.0), bellrand(120.0, 5.0));
    printf("%6.2f %6.2f %6.2f ",
      bellrand(50.0, 10.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    switch (rand(2))        case 1: printf("%6.2f %6.2f %6.2f ",
               bellrand(180.0,5.0), bellrand(180.0,5.0), bellrand(180.0,5.0));
            printf(" %s1\n", name); break;
       case 2: printf("%6.2f %6.2f %6.2f ",
               bellrand(120.0,5.0), bellrand(120.0,5.0), bellrand(120.0,5.0));
            printf(" %s2\n", name); break; }
} /* end of make-class1 */

make-class2(name)
char name[];
{
    float    bellrand();
    printf("%6.2f %6.2f %6.2f ",
      bellrand(100.0, 5.0), bellrand(140.0, 5.0), bellrand(140.0, 5.0));
    printf("%6.2f %6.2f %6.2f ",
      bellrand(50.0, 10.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
      bellrand(140.0, 5.0), bellrand(140.0, 5.0), bellrand(140.0, 5.0));
    printf(" %s0\n", name);
} /* end of make-class2 */

make-class3(name)
char name[];
{
    float    bellrand();
    int         rand();
    printf("%6.2f %6.2f %6.2f ",
      bellrand(100.0, 5.0), bellrand(160.0, 5.0), bellrand(160.0, 5.0));
    printf("%6.2f %6.2f %6.2f ",
      bellrand(50.0, 10.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    switch (rand(2)) {
        case 1: printf("%6.2f %6.2f %6.2f ",
               bellrand(160.0,5.0), bellrand(160.0,5.0), bellrand(160.0,5.0));
            printf(" %s1\n", name); break;
        case 2: printf("%6.2f %6.2f %6.2f ",
               bellrand(200.0,5.0), bellrand(200.0,5.0), bellrand(200.0,5.0));
            printf(" %s2\n", name); break; }
```

```
} /* end of make-class3 */

make-class4(name)
char name[];
{
    float    bellrand();
    printf("%6.2f %6.2f %6.2f ",
       bellrand(100.0, 5.0), bellrand(180.0, 5.0), bellrand(180.0, 5.0));
    printf("%6.2f %6.2f %6.2f ",
       bellrand(50.0, 10.0), bellrand(90.0, 10.0), bellrand(120.0, 20.0));
    printf("%6.2f %6.2f %6.2f ",
       bellrand(100.0, 5.0), bellrand(100.0, 5.0), bellrand(100.0, 5.0));
    printf(" %s1\n", name);
} /* end of make-class4 */
/*-------------------------------------------------------------------------*/
```