Machine Learning, 5, 267–298 (1990)
© 1990 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Empirical Learning as a Function of Concept Character

LARRY RENDELL RENDELL@CS.UIUC.EDU

HOWARD CHO HCHO@GONDOR.CS.PSU.EDU

Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, Illinois 61801 U.S.A.

Editor: Pat Langley

Abstract. Concept learning depends on data character. To discover how, some researchers have used theoretical analysis to relate the behavior of idealized learning algorithms to classes of concepts. Others have developed pragmatic measures that relate the behavior of empirical systems such as ID3 and PLS1 to the kinds of concepts encountered in practice. But before learning behavior can be predicted, concepts and data must be characterized. Data characteristics include their number, error, "size," and so forth. Although potential characteristics are numerous, they are constrained by the way one views concepts. Viewing concepts as ***functions over instance space*** leads to geometric characteristics such as concept size (the proportion of positive instances) and concentration (not too many "peaks"). Experiments show that some of these characteristics drastically affect the accuracy of concept learning. Sometimes data characteristics interact in non-intuitive ways; for example, noisy data may degrade accuracy differently depending on the size of the concept. Compared with effects of some data characteristics, the choice of learning algorithm appears less important: performance accuracy is degraded only slightly when the splitting criterion is replaced with random selection. Analyzing such observations suggests directions for concept learning research.

Keywords. Empirical concept learning, concepts as functions, experimental studies

1. Introduction

Data character affects concept learning. If a ***data characteristic*** is some property of the training instances input to a learning program, then accurate learning depends on characteristics such as the number of data (Haussler, 1986; Kearns, Pitt, and Valiant, 1987), the kind of data (Buchanan, Rissland, Rosenbloom, Ng, and Sullivan, 1987; Winston, 1975), and the amount of error in the data (Quinlan, 1986). Other data characteristics important to learning include the number of attributes, the size of the concept class, and the concept's concentration in instance space (i.e., the number of disjunctive "peaks;" Rendell, 1988, in press). Despite the extensive empirical and theoretical investigation into these relationships, however, many questions remain. Which data characteristics affect learning most? How do characteristics interact? Can data be characterized in ways that will improve concept learning? This paper explores various data characteristics and their effects on learning.

The way one analyzes data depends on how one views a concept. For our studies it has proven useful to view a concept as a function or surface over instance space (Rendell, 1986b; Rendell, Cho, and Seshu, 1989). This perspective can reveal hidden characteristics. For example, viewing a concept as a function leads one to wonder about concept shape and the effects of grossly irregular concepts (e.g., functions having many peaks).

Another reason to view concepts as functions is to overcome the problem of data collection. Thorough studies are difficult if they rely exclusively on "natural" data. On the other hand, if concepts are functions, we may ***characterize*** them to generate artificial data. Through systematic data generation, specific data characteristics can be finely controlled by manipulating the generating function. We designed and implemented a program for generating artificial data based on the characteristics we have described. Previous studies have used both natural concepts (e.g., Lavrac, Mozetic, and Kononenko, 1986) and artificial concepts (e.g., Langley, 1987; Valiant, 1984).

Although we have investigated several data characteristics, this paper focuses on two major ones: concept size and concept concentration. Concept ***size*** is the proportion of positive instances. Concept ***concentration*** characterizes the distribution of positive instances throughout the instance space: High concentration implies few concept regions; low concentration means dispersion across many disjunctive regions. As the experiments demonstrate, both of these characteristics have a profound effect on the accuracy of concept learning.

Other data characteristics influence learning to greater or lesser degrees. In our experiments we have varied some of them while observing their effects on the two major characteristics mentioned above. The minor characteristics reported in this paper are class error, attribute error, peak shape, and number of data. One of the general results of our study is that these data characteristics can interact in non-intuitive ways.

In our quantitative investigation we have used two systems: ID3/C4 (Quinlan, 1983) and PLS1 (Rnedell, 1983). These inductive systems represent a class of algorithms that specialize hypotheses using information or probabilistic splitting criteria; for the problems we address, these algorithms behave particularly well (Lavrac et al., 1986; O'Rorke, 1982; Rendell et al., 1989). We consider both ID3/C4 and PLS1 to compare the effects of design choices with the effects of data character. We also demonstrate the generality of the results through a set of experiments in which the splitting criterion is replaced with random selection. Surprisingly, performance accuracy is not significantly degraded, compared with the effects of some data characteristics.

Our characterization of data and the experimental results motivate certain extensions of empirical learning methods. For example, such methods behave poorly when concept concentration is low, which suggests that improving this important characteristic should be a driving principle of mechanized representation change.

This paper develops data characteristics from first principles. Beginning from the definition of concepts as functions, we review and reorganize various data characteristics relevant to our investigation (Section 2). Next we outline the design of the experiments and discuss our method for artificial data generation (Section 3). Then we present and analyze experimental results that focus on concept size, concept concentration, and splitting criteria (Section 4). Finally, we summarize by suggesting methods for improving concept learning, while proposing questions for additional study (Section 5).

2. Characterizing the data

Aspects of concept learning, such as accuracy and speed, depend on the learning algorithm and the problem domain. Here we are primarily interested in the latter: What are the effects of the many kinds of data that can be encountered? We consider this question in five parts. First, Section 2.1 reviews current notions of data character. To help clarify these data characteristics, Section 2.2 examines concept structure. Sections 2.3 and 2.4 continue this analysis by refining dimensions of "concept distribution" and "sampling distribution." Finally, Section 2.5 summarizes and organizes the many data characteristics to help rationalize the experiments.

2.1. A preliminary view of data variables

Some data characteristics are known to affect concept learning in specific ways; other data characteristics are less well understood. This section begins with some well-known effects and leads to open research problems.

Number of data in the learning sample. The number of training instances affects the speed of learning. Some systems are slowed only linearly as the sample size increases.[1] To reduce the learning time, the available data may be sampled, although the accuracy can suffer if the sample is too small. Whereas ID3 determines the sampling empirically (Quinlan, 1983), the required number of data for a given accuracy can sometimes be decided theoretically (Ehrenfeucht, Haussler, Kearns, and Valiant, 1988).

Number of attributes. The speed of learning is also affected by the number of attributes that describe an input datum. Some learning algorithms (such as ID3 and PLS1) are slowed only linearly. If the number of attributes is large, techniques of attribute selection may be used to save time without sacrificing accuracy (Devijver and Kittler, 1982; Draper and Smith, 1981; Samuel, 1963). Another way to improve the attributes is to replace some with more appropriate choices using a domain theory (Mitchell, Keller, and Kedar-Cabelli, 1986).

Scales of attributes. The scale of an attribute may be nominal (unordered), tree-structured (partially ordered), ordinal (totally rank ordered), or interval (integer or real). The scales toward the end of this list code more information than those at the beginning; such information may benefit learning. Techniques to manage various scales are given by Michalski (1983) and by Quinlan (1983). Scales are often interconvertible (Anderberg, 1973).

Error or noise. The effects of noisy data can be severe. In a detailed study, Quinlan (1986) artificially corrupted data input to ID3. He introduced varying degrees of corruption, sometimes to attribute values and sometimes to class membership values, to show how each degrades accuracy (also see Breiman, Friedman, Olshen and Stone, 1984; Rendell et al., 1989). Later in this paper we describe new results that relate error to attribute scales and other data dimensions.

Class distribution. Except for some interesting but limited theoretical results (Cover, 1965; Haussler, 1986; Ehrenfeucht et al., 1988), the effects of concept "distribution" are not well understood. By ***concept*** or ***class distribution***, we refer to the way the positive instances are arranged in instance space. To mention two extremes, the positive instances may form just a single, tightly packed group or cluster, or they may be widely scattered over instance space. One might suppose that tightly packed groups are easier to find and describe than concepts that are dispersed.[2] Section 4 reports results of experiments using varied class distributions. We also study how this data variable interacts with error and other data dimensions. To lay the groundwork for these results, Sections 2.2 and 2.3 refine the idea of concept distribution.

Sampling distribution. The distribution of a concept may or may not be reflected in the training examples. In other words, relative to the whole population of instances, the training sample may be random, or it may be biased in various ways (Hunt, Marin, and Stone, 1966; Simon and Lea, 1974). Because some instances are more useful for learning than others, the training data may be limited not only by random sampling, but also by "intelligent" sampling (Buchanan et al., 1987; Sleeman, 1981). Section 2.4 refines the notion of the ***sampling distribution.***

2.2. A closer look at concepts: From definition to characterization

By definition, a concept is a rule that describes a class (subset) of instances. If we represent an instance as a tuple of attributes, any particular concept has an associated instance space whose dimensionality is the number of attributes. As shown in Figure 1(a), an "all-or-none" concept is a binary-valued function over instance space. We allow a concept to be probabilistic, as in Figure 1(b), so the function becomes graded: It has values between 0 (certain class exclusion) and 1 (certain class membership). Thus a concept is a (binary or graded) ***function*** or ***surface over instance space.***

Any concept expressed using an n-tuple of attributes is a function over the instance space defined by those attributes.[3] The instance space of a concept using n attributes is n-dimensional; the structure of the space depends on attribute scales. The diagrams in Figure 1 show familiar interval scales which are ordered and permit a distance measure. If the attributes are real-valued, the concept may be a continuous function; if the attributes are integer, the concept may still be fairly "smooth" (it might vary gradually from one point to the next). Partially ordered and nominal scales also determine an instance space, albeit a less ordered space. Because ordering and smoothness make concepts easier to learn using empirical systems (Rendell, 1986), partially ordered and nominal scales are more problematic, although such variables may be converted to integer (Anderberg, 1973).

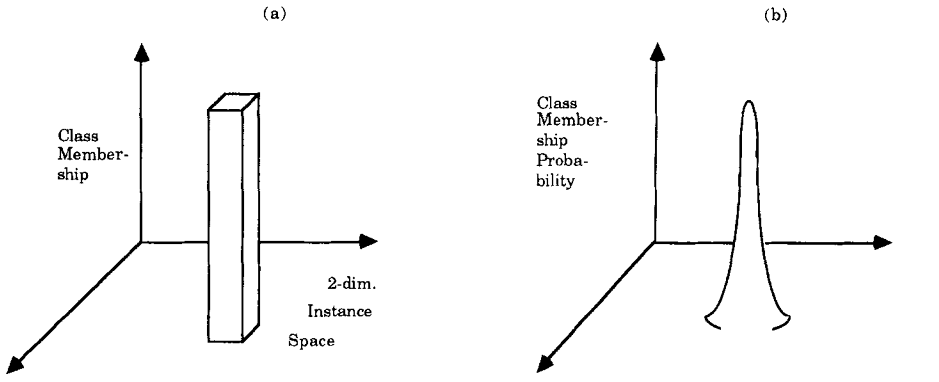

Figure 1. Binary-valued and graded concepts. In (a), the concept is binary or all-or-none; in (b), it is graded or probabilistic. Here the instance space is two-dimensional and has interval scales, showing clear differences in "concept conformation" (shape of individual peaks). In (a), the concept is uniform; in (b), it is normally distributed.

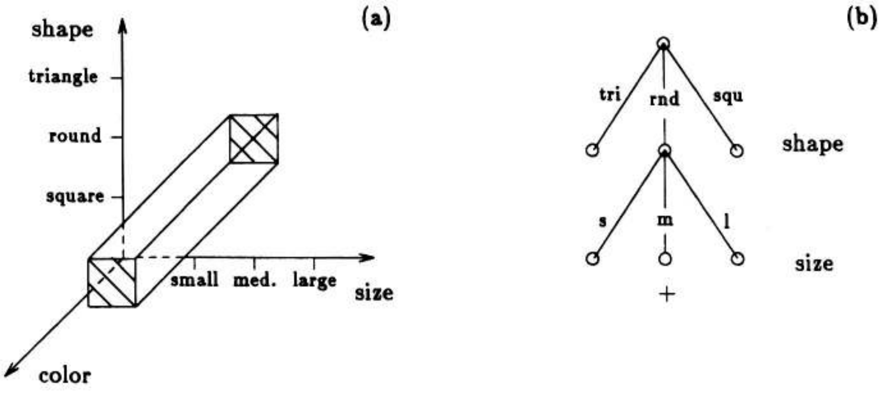

Figure 2. Relationship of a decision tree to the instance space form of a concept. The two diagrams are related. The instance space form (a) corresponds to the decision tree (b) although the latter orders attributes. (Whereas (a) suggests that shape and size are equally important, (b) shows that shape is more significant. Both (a) and (b) indicate that color is irrelevant.) Another difference between the two representations is that with usual interpresentations, one tends to think of (a) but not (b) as ordering attribute *values*.

Although one can represent a concept in various ways, all adequate representations are logically equivalent to a function over instance space. For example, the decision tree in Figure 2(b) describes the same class as the binary-valued function in Figure 2(a). We have introduced the idea of a concept as a function because this perspective helps to clarify data dimensions. For example, refining the class distribution dimension becomes the problem of ***characterizing the class membership function.*** The character of the function affects the difficulty of learning.

2.3. Refining the notion of class distribution

How does concept difficulty depend on the distribution of the class? For binary-valued concepts, one way to describe difficulty is in terms of the number of conjuncts and disjuncts in the concept (Haussler, 1986; Kearns et al., 1987). To permit graded concepts, however, we propose to break down the class distribution dimension into more refined data variables. Each refined dimension in Table 1 refers to some aspect of the concept as it appears over its instance space. We illustrate and discuss the first three dimensions below, relegating the fourth to a later discussion on future work.

2.3.1. Relative size of the concept or prevalence of positives

Figure 3 shows two extreme cases of instance space coverage, relative concept size, or prevalence of positive instances. Concept size is the positive volume (integral of the class membership function) divided by the volume of the whole space. The size of the concept may affect accuracy, especially when the data are noisy. Suppose that some attribute of a

Table 1. Class distribution dimensions and their hypothesized effects.

Data Variable or Dimension	Explanation or Typical Values	Specific Effects (hypothesis)
Relative Size of the Concept (prevalence of positives)	proportion of instance space covered by the concept	Minor effect on accuracy, unless there is noise
Concentration of Positives (number of peaks)	localization of the concept in instance space	Major effect on accuracy, speed, other performance considerations
Conformation of Positives (shape of individual peaks)	e.g., uniform, normal, or irregular	Some effect, as the shape becomes more spread out or irregular
Higher-Order Regularity (association of peaks)	random distribution of peaks, vs. some way to relate them	No effect on systems tested, but important in future systems

negative instance is corrupated. The effect should be greater in Figure 3(a) because the probabillity of the corrupted instance's landing in a positive region of the space is ½, whereas in Figure 3(b) this probability is only 1/100.

Figure 3(b) is typical of data found in some problems. For example, when we examined the output from PLS1 in medical domains (Clark and Niblett, 1989; Lavrac et al., 1986), we found small underlying concepts (just a small proportion of all medical patients suffer from any given disease). Many concepts learned in the literature are small, although the relative size can vary greatly.

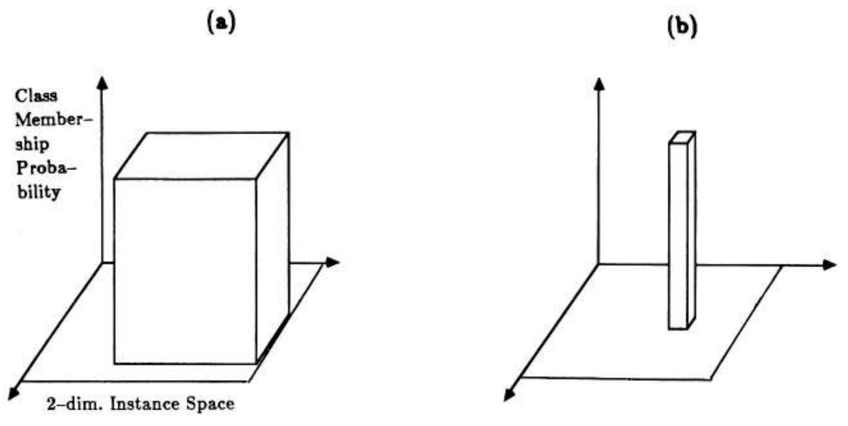

Figure 3. Variation of relative concept size (prevalence of positives) in instance space. Although the diagram shows two attributes, the space may have any number of dimensions. In (a), the concept covers half of the space. In (b), the concept covers just 1% of the space. Concept size can affect learning behavior in the presence of error (see text).

2.3.2. Concept concentration

Concepts may also vary in the ***concentration***. A coarse measure of concentration simply counts the number of distinct disjuncts or "peaks" in the concept. For an interval (integer- or real-valued) instance space S, a *peak* is a neighborhood[4] of S that has a class membership value greater than average. "Neighborhood" is a meaningful term only for ordered scales (integer, real, partially-ordered); for other scales (e.g., nominal) a peak is always a single point until some ordering is learned or imposed. Figure 4 contrasts two extremes of concept concentration in an integer space: one peak (a), and many peaks (b). Counting the number of peaks provides a simple measure of the degree of localization of a concept, which may cohere tightly or be spread out in instance space.

Concept concentration, or its opposite, concept ***dispersion***, can be related to problems found in the literature. Holte and Porter (1988) surveyed several machine learning articles and found that the reported concepts tend to have few disjuncts. Moreover, because machine learning algorithms such as ID3 and PLS1 align disjuncts with instance space axes, several disjuncts may indicate just one peak: An irregularly shaped peak misaligned with the axes requires many disjuncts to describe it. Small numbers of peaks is characteristic of representations used in many domains, including games (Samuel, 1963), puzzles (Rendell, 1983), and various pattern recognition problems (Tou and Gonzalez, 1974). Because of the attributes typically chosen for these problems, class membership functions are remarkably well-behaved. The function is often ***monotonic*** or ***singly peaked***.

In contrast, some problems (and alternative representations of these same problems) have low-level (primitive) attributes, which tend to produce dispersed concepts. Consider, for example, what the class-membership function would look like when the concept is the letter "A" and attributes are pixel gray-levels, or when the concept is "forced win" and attributes are chessboard contents. Class-membership functions having many peaks have been considered by Quinlan (1983, 1987b) and by Rendell (1985, 1988). As Section 4 shows, concept dispersion has a drastic effect on learning.

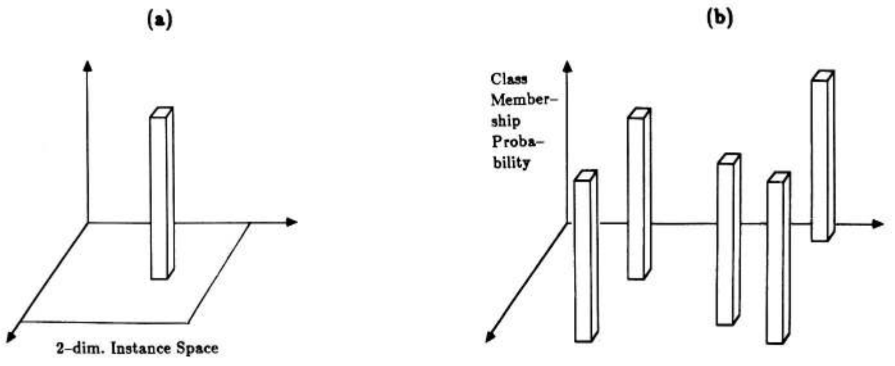

Figure 4. Variation of concept concentration. In (a), the concept is tightly grouped in a single region of instance space. In (b), the concept is spread out all over the space. Concept concentration can affect learning drastically.

2.3.3. Concept conformation or the shape of individual peaks

Figure 1 shows two different conformations of positive instances: uniform (a) and normal (b). Many natural cases are like Figure 1(b) (Tou and Gonzalez, 1974). Although the number of shapes is infinite, the effect of this data dimension on learning may be less important than the number of peaks. Later we test this hypothesis.

2.4. Refining the notion of sampling distribution

The simplest sampling characteristic is perhaps its size, but there are other aspects of sampling (Hogg and Craig, 1965). First, the sample may or may not be meaningfully ordered (Winston, 1975). Secondly, the distribution over instance space may or may not be reflected in the training examples; in other words, relative to the whole population of instances, the training sample may be random or it may be biased. Finally, the balance of positive versus negative examples may differ relative to the actual class distribution; in other words, the proportion of positive instances in the data can be larger or smaller than in the whole population. This may occur in practical applications; for example, if the class to be learned is some disease, one may have many recorded cases of the disease but few explicit records of health.[5]

2.5. A summary of data dimensions

Table 2 organizes the data dimensions into a coherent structure based on our view of concepts as surfaces over instance space. We have referred to coarse data variables (such as concept distribution) and refined data variables (such as concept concentration). The table lists many refined variables (Column 2) as specific cases of three coarser variables (Column 1).

This table also previews the experiments. Each was designed to measure quantitative relationships between particular aspects of learning and particular data variables. Our experiments emphasize certain variables, especially concept size and concept concentration (Column 3). In some cases the data variables may interact; their effect may depend on other dimensions. For example, we argued in Section 2.3.1. that the effect of error should depend on the size of the concept. Our experiments examine such relationships (Column 4).

Table 2. Data dimensions and their investigation.

Coarse Data Variable	Refined Data Variable	Investigated Here?	with Other Var's?*
Problem Dimensions	Attribute Scales	Somewhat	Shape, No. of Peaks
(Domain Character,	Number of Attributes	No	—
Representation,	Attribute Error	Yes	Concept Size
Data Reliability)	Class Error	Yes	Number of Peaks
Concept Dimensions	Size of Concept	Yes	Error
(Class Distribution)	Concentration (No. of Peaks)	Yes	Scales, Error
	Conformation (Shape of Peaks)	Yes	Number of Peaks
	Higher-Order Regularity	No	—
Sampling Dimensions	Number of Data	Somewhat	Number of Peaks
(Training Biases)	Ordering Characteristics	No, always random	—
	Sign Balance Relative to Class	Somewhat	—

*This column shows secondary data characteristics found to interact with the primary data characteristic listed in the corresponding row of the table.

3. Experimental design

This section describes our broad experimental conditions, algorithms, and measures. We select two inductive systems known to behave well, discuss dependent measures of their behavior, and outline our method for obtaining data to train and test the systems.

Our experiments used two algorithms because we wanted to test the effects of their design compared with the effects of data and concept character. One of our hypotheses was that although various specialization or splitting algorithms differ in several respects, their design has a minor effect compared with data and concept character. To verify this hypothesis and to investigate effects of data character in detail, we varied the learning algorithm, measured accuracy and speed, and generated a broad selection of data.

3.1. Learning algorithms

The algorithms we test are ID3 (Quinlan, 1983, 1986) and PLS1 (Rendell, 1983, 1986).[6] These learning systems are similar: Both input data as attribute vectors; both use probabilistic or information criteria to specialize hypotheses; and both represent hypotheses in a similar fashion. Although ID3 uses a decision tree whereas PLS1 uses hyperrectangles, the two representations are similar (see Figure 2 and the Appendix). These programs output a rule or concept to classify unknown instances into positive or negative classes (although ID3 can handle multiple classes).

Because these learning algorithms are of secondary concern here, we relegate their detailed description to the Appendix. However, we note a few important facts:

- ID3 and PLS1 represent a class of specialization algorithms[7] that use probabilistic or information criteria. For others in the same class, see Breiman et al. (1984), Clark and Niblett (1989), and Gams and Lavrac (1987).
- In a number of experiments whose conditions were similar to those we investigate here, this class of learning algorithms was found to be faster and more accurate than generalization algorithms (Lavrac et al., 1986; O'Rorke, 1982; Rendell et al., 1989).[8]
- Viewing algorithms such as ID3 and PLS1 as members of a single class of learning systems can improve our perspective on the comparative effects of system design versus data character. We can vary both algorithm design and data character to determine their relative roles.

In this paper we primarily vary data characteristics to determine their effects on measures of concept quality.

3.2. Dependent measures

To measure the effects of various data characteristics, we need evaluation criteria. Numerous measures have been used to assess learning systems (see Lavrac et al., 1986; Rendell et al, 1989). Here we use measures not so much to compare systems, but rather to evaluate the effects of data character. Two important criteria are concept accuracy and learning speed. Although conciseness may also be important for some purposes, we do not report this aspect.

Because of the difficulty of real-world problems, the exact concept is seldom obtained. Instead, the concept output by the learning system is an approximation. The quality of the approximation may be defined in terms of its classification error. In other words, given a sample of instances whose true classification is known, the estimated accuracy of a concept is the estimated probability of correct classification = (number of correct classifications)/(number of instances tested). This definition of accuracy is standard (Tou and Gonzalez, 1974; Valiant, 1984).[9]

Although accuracy was our primary concern, we also measured speed or efficiency. This factor can be significant because slow learning can mean impractical learning. In some cases learning can become infeasible. We measured quantitative effects of problem characteristics on learning speed.

3.3. Data generation

As Table 2 shows, learning depends on many data characteristics. In order to measure their complex effects, one needs a wide range of data. In this respect, however, natural domains are often uncontrolled. Nevertheless, any natural data from any domain must have an underlying class membership function which *is* the concept (Figure 1). Because a function can be generated to have any desired form, one can simulate natural data by using artificially generated data (cf. Kearns et al., 1987; Langley, 1987).

Our data generation program has several parameters the user may select. The parameters include the desired type and number of attributes, the type and proportion of noise, information about the character of the class membership function, and choices for its sampling. The parameters and operation of the data generation program are explained in the next section, which details the experiments.

4. System behavior as a function of data character

We observed the behaviors of ID3 (the version called C4), PLS1, and a variant of PLS1. The versions used of ID3 and PLS1 were written in C and Pascal, respectively. Both were run on a VAX/780 under UNIX. Although the designs of ID3/C4 and PLS1 differ somewhat (see the Appendix), we have found little difference in their average accuracy. Elsewhere (Rendell et al., 1989), we showed that ID3 and PLS1 behave similarly with respect to accuracy, speed, and conciseness. These specialization algorithms, however, behave differently from generalization algorithms.

Because ID3 and PLS1 behave so similarly and because some experiments took several hours to run, we often used only PLS1, instead of wasting resources in what would have amounted to duplicate runs. In other experiments we tested both systems to underline their similar behavior and to contrast effects of system design with effects of data character.

To exploit data for both training and testing, Breiman et al. (1984) discuss three methods. The most common is to partition the sample and reserve part of it for testing. We used our data generation program to create testing and training instances at the same time (one half the data for each purpose). In all cases, both sets of instances had identical properties, except that the instances used for testing were never corrupted. The rationale behind this exception is that testing should be objective, not flawed (cf. Quinlan, 1986). For ease of experimentation, the attributes of a given data set were all of the same type—either integer or nominal. Another parameter of data generation is the number of values in an attribute's range. The number chosen (32) was rather arbitrary, although in combination with the dimensionality of instance space (5), this gives a size of $32^5 = 3.4 \times 10^7$, which is large enough so that even large samples cover just a fraction of the space.

Table 3 summarizes these and other data characteristics for all our experiments. This table follows the format of Table 2, listing detailed data dimensions (Column 2) as refinements of coarse characteristics (Column 1). The other columns show data characteristics for four major series of experiments. Each series is designed to show effects on learning of some major variable upon which the study is focused. The major variables (shown above the figure numbers in Table 3) are two data characteristics: concept size and concept concentration, and also a factor of system design discussed later.

These factors were not the only ones we varied. The effects of other concept characteristics were also studied, including most of the data dimensions of Column 2. Characteristics that varied over a series of experiments are shown in the table using an asterisk (*). For other variables, we set data characteristics to mimic some "natural" cases. For example, the last row of Table 3, which shows the proportion of positive examples, was either the same as the class distribution (Figure 5), or even (50% positive and 50% negative; Figures 6 to 9).

We evaluated effects of data characteristics using the two criteria of Section 3 (concept accuracy and learning speed). The ordinate of all but one of the graphs is accuracy. The lower accuracy limit is 50%, because in the two-class case with even sign distribution, this is the expected accuracy for guessing. Standard deviations for accuracy were at least as good as 3% (often 1%); all the phenomena we report are statistically significant. We present and interpret the results in four separate sections.

4.1. Effects of concept size and noisy data

This first set of experiments varied the size of the concept and the amount of error or noise in the data. As a function of these variables, we measured concept accuracy.

Table 3. Data characteristics for experiments.

		Experiment Focus and Figure Number				
		Size	Concentration (Peaks)			Algor.
Coarse Data Variable	Refined Characteristic	5	6	7	8	9
Problem Dimensions	Attribute Scales	Int.	Int.	*	Int.	Int.
(Basic Representation	Number of Attributes	5	5	5	5	5
and Data Reliability)	Attribute Error	*	0	0	0	0
	Class Error	*	0	*	*	0
Concept Dimensions	Relative Size	*	1%	1%	1%	1%
(Class Distribution)	Number of Peaks	1	*	*	*	*
	Peak Shape	Unif	*	Unif	Unif	Unif
Sampling Dimensions	Number of Training Data	2000	2000	2000	*	2000
(Training Biases)	Proportion of Positives	class	even	even	even	even

4.1.1. Experimental conditions

As illustrated in Figure 3, the concept characteristic we call relative concept size is the proportion of instance space covered by positive instances. Figure 5 gives the percentage of testing instances correctly classified, as a function of concept size.

Accuracy is also a function of (a) class error (flipping the sign of the instance) and (b) attribute error (random substitution of attribute values). To introduce a desired amount of noise, the data generation program was instructed to corrupt the training data it produced (cf. Quinlan, 1986). For class error, the program corrupted the class membership of a desired proportion of instances. For attribute error, the program randomly selected a desired proportion of instances and randomly substituted new values to some of their attributes.[10] The settings of the other data dimensions are shown in Table 3.

4.1.2. Observation and discussion

The graphs in Figure 5 support earlier experimentation and they also express less familiar relationships. First, our results agree with Quinlan's (1986) findings which show that induction systems like ID3 and PLS1 can learn accurately despite noise. Figure 5 also agrees that class error (a) is more damaging than attribute error (b). One reason is that corrupting the class value not only destroys information but reverses it; in contrast, corrupting an attribute tends to leave enough information in the uncorrupted attributes for adequate learning. Another reason that attribute error is less damaging is that incorrect attribute values may sometimes produce correct examples.[11] In other words, class noise is worse than attribute noise because flipping the class value guarantees that an error is introduced, whereas distorting an attribute value may or may not introduce error.

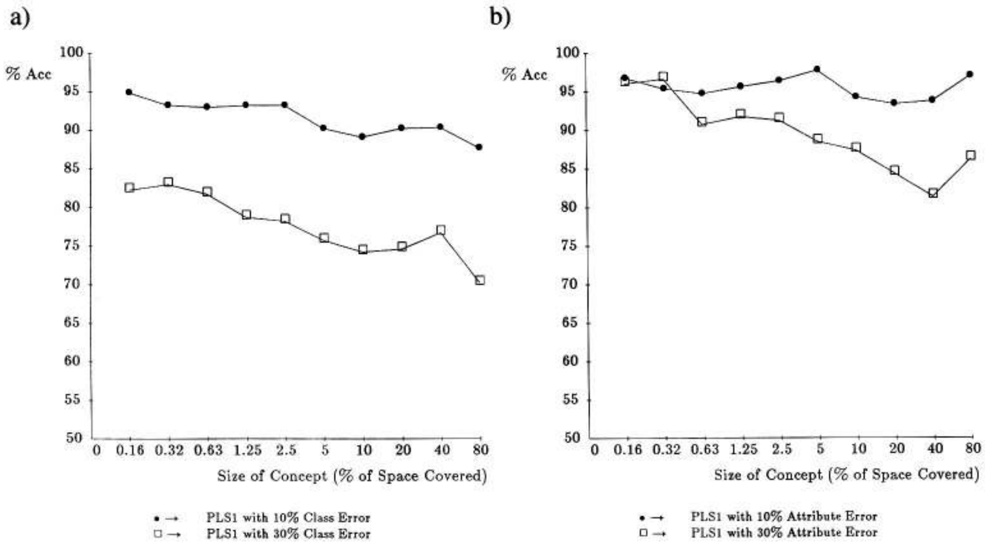

Figure 5. Variation of accuracy with concept size and noise. The abscissa is the proportion of positive instances covering instance space (logarithmic scale). In two separate experiments both class corruption (a) and attribute corruption (b) were also varied.

Another phenomenon may be less well known. Section 2.3.1 hypothesized that the effect of attribute error is greatest if the concept covers about 50% of the space. Figure 5(b) agrees: Classification accuracy is minimum at about 50% coverage, whereas attribute error has a lesser effect in either small or large concepts. According to the figure, even 30% attribute error degrades performance only slightly if the concept covers 1% or less of the space. But if the concept covers close to half the space, 30% attribute error degrades accuracy by up to 15% ***more***.

Why is the effect of attribute error largest when the concept covers half the space? Suppose that some attribute of a negative instance is corrupted. This will give an incorrect example only if the altered instance lands in a positive region of the space. Hence, the effect of corrupting a negative instance is greater if it has a high probability of landing in a positive region (i.e., if the concept is large). Conversely, the effect of corrupting a positive instance is great if the concept is small. And because both positive and negative instances may be corrupted, the greatest effect occurs when both conditions are maximal (subject to a tradeoff constraint).

We can state this argument more formally. Let the relative size of the concept be p ($0 \leq p \leq 1$). Hence, p is the probability that a random corrupted event x is positive. If attribute corruption is random, the probability that x will (incorrectly) land in a negative region of instance space is $1 - p$. Similarly, the probability that x is negative is $1 - p$, and the probability that it will (incorrectly) land in a positive region is p. In either case, the probability that x will become incorrect is $P = p(1 - p)$. By differentiating and setting the result to zero, we find that P has a maximum at $p = \frac{1}{2}$.

If attribute error is random, this phenomenon may be important in practice. Even if experiments on attribute error show a small effect, we still need to know concept sizes, both in the experimental domain and in any projected domains, before we can predict accuracy.

4.2. *Effects of concept concentration (and two other factors)*

In this section we observe concept accuracy as a function of concept concentration (number of peaks). We also consider the effect of concept shape and learning algorithm.

4.2.1. *Experimental conditions*

In instance spaces having interval scales, natural data tend to form one or more peaks. Instances of a class often cluster around the "center" of a peak. To simulate this structure, the data generation program selects random centers in instance space and generates instances around each center. Each center has a maximum class membership value of one; the probability of an instance's being in the (positive) class decreases with its distance from a center. The program allows the user to vary concept concentration by inputting the desired number of centers or peaks. Peak size may be varied, although peaks have uniform sizes within any given concept.

The data generation program also allows the user to select concept shape. Although the program assumes that the class is isotropic in instance space, the program allows various distributions, two of which are uniform (binary class membership) and normal (graded

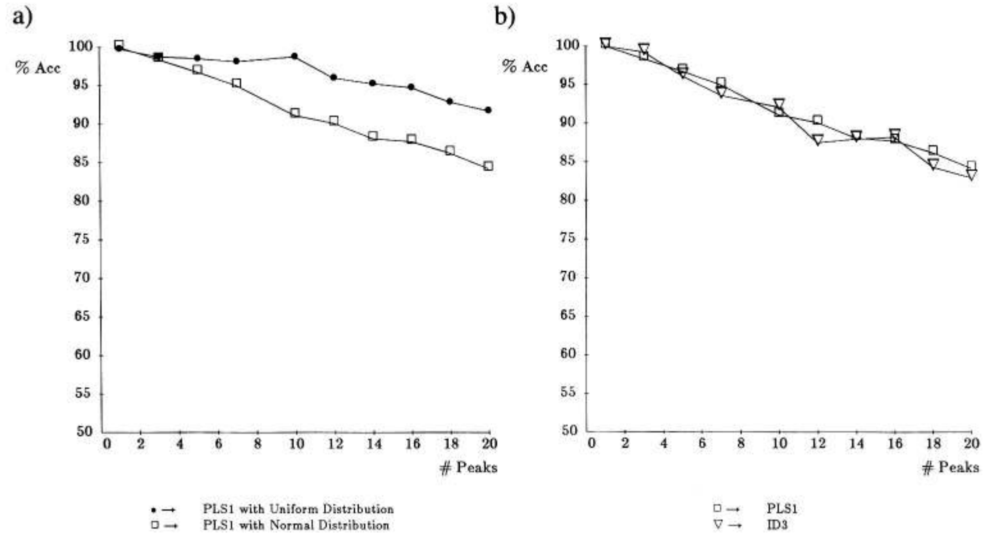

Figure 6. Variation of accuracy with concept concentration, measured as the number of peaks in the concept. The accuracy depends on the shape of the concept (a) but apparently not on the learning algorithm (b).

membership). For a uniform distribution, class membership changes abruptly (from 1 to 0 or 0 to 1) at the class boundary, as in Figure 1(a). For a normal distribution, class membership decreases gradually, as in Figure 1(b). (Rather than being an input parameter, the variance of the distribution was implicitly determined by concept size.) In the current version of the data generation program, peak shape is uniform within a given concept. For these experiments, no training error was introduced, although a normal distribution (or any graded concept) effectively embodies error.

Figure 6(a) shows accuracy as a function of concept concentration and concept shape, whereas Figure 6(b) gives accuracy as a function of concept concentration and learning algorithm. We observed both ID3 and PLS1 to compare their relative behaviors with effects of data characteristics.

4.2.2. *Observation and discussion*

Using PLS1 as the learning algorithm, Figure 6(a) shows the effect on accuracy as a function of concept concentration. For either shape (uniform or normal), a concept having only one peak is learned almost perfectly. But accuracy drops significantly as the number of peaks increases to twenty. A uniformly-distributed concept having twenty peaks (a) gives an accuracy of 92%; a normally-distributed concept (b) gives an accuracy of 84%.

Why does accuracy vary with these two choices of concept shape? The uniform distribution gives a fixed class membership value (either 0 or 1) over local regions of instance space. In contrast, the normal distribution gives only a probability of class membership. Although this probability u is converted to 0 or 1 by the data generation program, u merely weights the binary conversion (e.g., $u = .9$ means a 90% chance of an instance's being positive). Because regions of instance space can have these mixed class values, this is like the introduction of class error.[12]

Figure 6(b) shows the result of fixing concept shape (to be uniform) but varying the learning algorithm. This diagram shows that ID3 and PLS1 learn less accurately as the number of peaks in the concept increases from one to twenty. Although ID3 and PLS1 differ in significant ways (see the Appendix), the effect of these differences is indistinguishable, given the resolution of these experiments ($\pm 3\%$).

4.3. Further effects of concept concentration (and other data variables)

In this section we continue our study of concept concentration as we increase the number of peaks to larger values. We also vary class error and attribute scales.

4.3.1. Experimental conditions

To simulate the clustering of natural data around peaks, the data generation program allows selection of peak multiplicity and shape. But shape makes sense only if attribute scales are real or integer. To generate data having nominal scales, the program first assumes an integer scale to produce the usual distribution of positive instances around centers of peaks. Then the program randomly scrambles the attribute values, thus destroying the smooth structure of the integer ordering.

For both nominal and integer scales, we generated many data sets having highly varied concept concentration: The number of peaks was increased from one to 1000, in logarithmic intervals (1, 2, 4, 8, . . .). Using this logarithmic abscissa, Figure 7 shows the classification accuracy primarily as a function of the number of peaks in the concept. The other data characteristic we varied is the amount of class error. The number of training instances was fixed at 2000.

As in Figure 7, the major variable of Figure 8 is also the number of peaks, but here the number of training instances was 20 ***per peak***. A concept having one peak had only 20 instances, whereas a concept having 1000 peaks had 20,000 instances. This experiment was performed to see if massive amounts of data could overcome the detrimental effects of having to spread the instances over all the peaks. Further, if massive data can help accuracy, what is the cost in terms of learning time? Figure 8(a) shows accuracy and Figure 8(b) shows learning time.

4.3.2. Observation and discussion

Whether the attribute scales were nominal, as in Figure 7(a), or integer as in Figure 7(b), the accuracy dropped nearly to chance as the number of peaks approached the fixed number of training instances (2000). Even when the sample size was increased to keep pace with the number of peaks, as in Figure 8(a), the accuracy still suffered. Furthermore, the number of disjuncts required (not shown) to describe complex concepts increases with the number of peaks. Worse, the effect on learning time is drastic. To process the 20,000 data for 1000 peaks, PLS1 took 20,000 CPU seconds (six hours); by extrapolation, ID3/C4 would take several days.

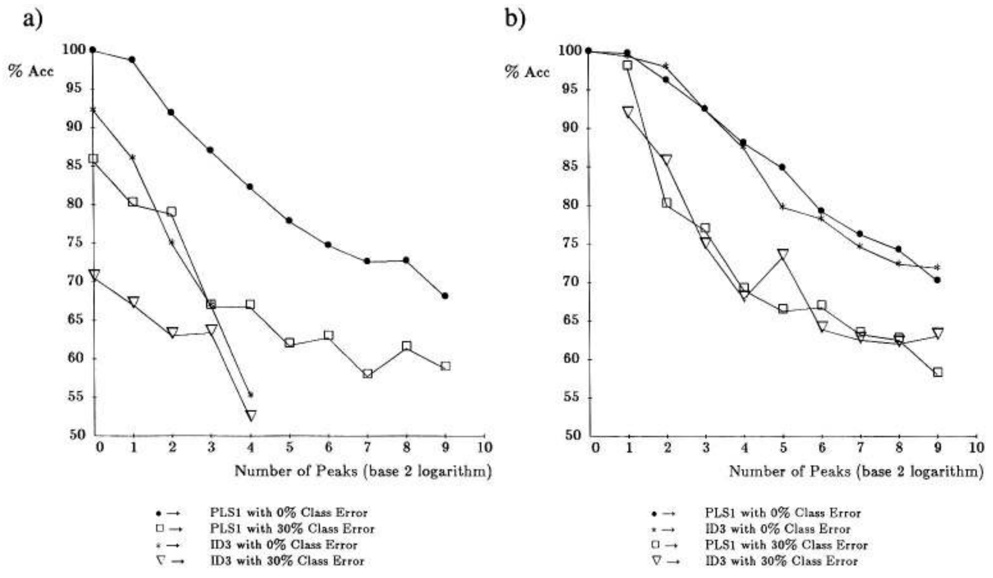

Figure 7. Variation of accuracy with diverse values of concept concentration. The abscissa is the logarithm of the number of peaks in the underlying concept. In (a) the attribute scales are nominal; in (b) the scales are integer. Both graphs compare two error rates and two learning algorithms.

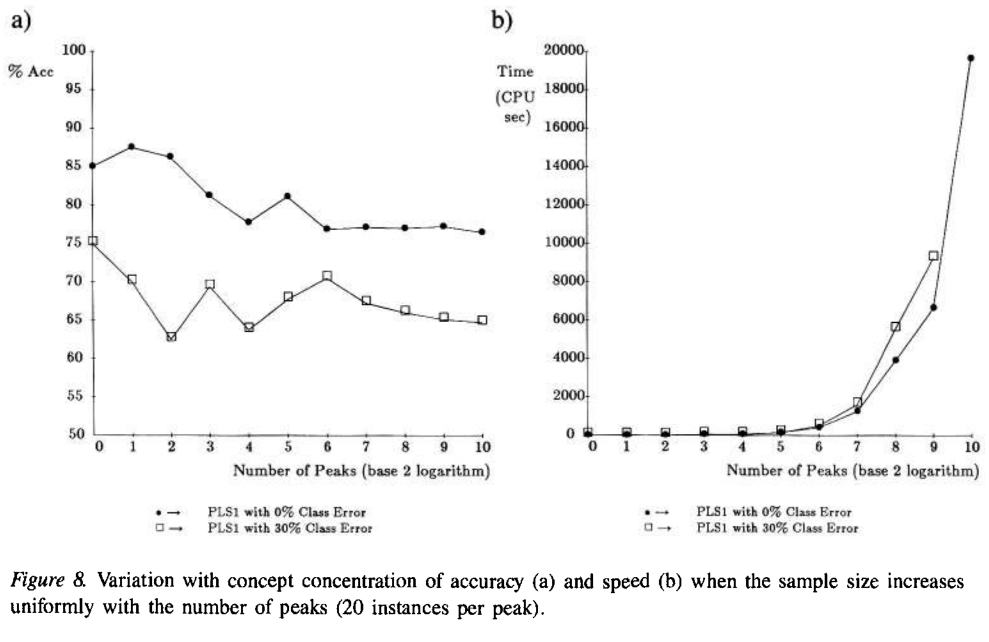

Figure 8. Variation with concept concentration of accuracy (a) and speed (b) when the sample size increases uniformly with the number of peaks (20 instances per peak).

Why do algorithms like ID3 and PLS1 behave so poorly when concepts are dispersed? Such systems provide no way to predict peaks. Although these induction systems are good for discovering the sizes and shapes of individual peaks in the class membership function, each peak must be learned ***separately***. Every peak requires at least one training example, if that peak is to be learned at all. And to shape a peak correctly, many instances may be needed, both from inside the peak (positive examples) and from outside the peak (negative examples). Moreover, as the number of peaks increases, so does the difficulty of finding them, because multiple peaks imply attribute interaction (high-order relationships), whereas typical systems rely on simpler correlations (Rendell, 1989).

Figure 8 suggests that the practical limit for our best empirical algorithms is perhaps between 1,000 and 100,000 peaks. A complexity analysis of PLS1 shows that, like ID3, its processing time t increases as the product of the number of instances N, the number of attributes n, and the number of disjuncts k (which varies with the number of peaks). Holding n constant, we would predict $t \propto Nk$. If we fit the model $t = c \times (Nk)^x$ to the values of Figure 8, we obtain $x = .57$ (this is less than the predicted value of $x = 1$ because of various heuristics in PLS1). Substituting $k = 100{,}000$ and $N = 20$ per peak (two million instances) into $t = (Nk)^{.57}$, we have $t =$ one CPU ***month***. Given the 10^{16} peaks that arise in real-world problems (Rendell, 1985), our current algorithms would be completely overwhelmed.

Problems whose initial descriptions give class membership functions having large numbers of peaks can be considered problems of representation. For example, even when restricted, a low-level representation of the protein folding problem results in many millions of peaks (Rendell, 1988, in press). The distinguishing feature of such domains is that a good representation is unknown or cannot easily be communicated.

Because the best known formulations of such problems tend to produce immense numbers of peaks in a primitive instance space, these problems require construction of new attributes that reduce the number of peaks in a transformed space. To automate construction, the original attributes must somehow be transformed so that peaks merge (Drastal and Raatz, 1989; Rendell, 1988, in press). This is discussed in Section 5.

4.4. The effect of one algorithm design factor

This final set of experiments has a different purpose. Rather than observe further effects of data character, we consider the effect of an aspect of algorithm design as a benchmark for comparison.

4.4.1. Experimental conditions

Figure 9 shows concept accuracy as a function of the number of peaks and also as a function of one aspect of algorithm design. This diagram is the same as Figure 6(b) except that one more curve has been added. The upper two curves copied from Figure 6(b) show the behavior of ID3 and PLS1. The third curve shows the effect on PLS1 when its splitting criterion was entirely randomized.

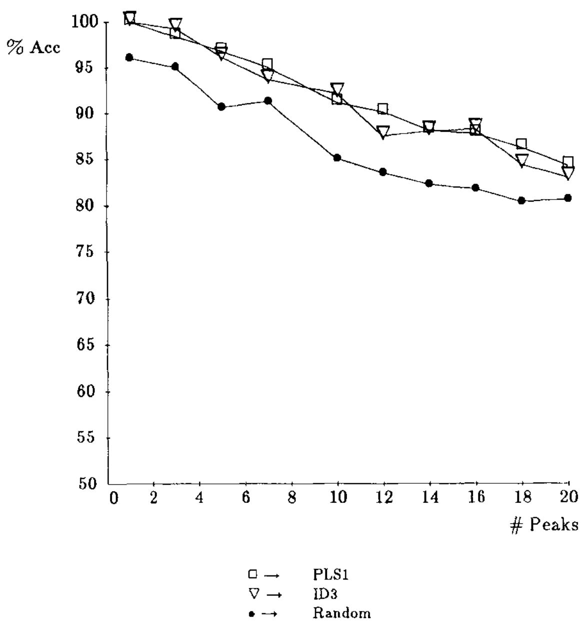

Figure 9. The comparative effect of a system design factor. The abscissa is the number of peaks in the underlying concept (linear scale). The splitting criterion used by ID3 and PLS1 is normally probabilistic or information theoretic (two top curves). The lowest curve shows PLS1 with its criterion altered so that splitting becomes random.

In the experiments represented by Figure 9, the peak shape was uniform and no error was introduced. We ran other experiments for various settings of peak shape (normal) and data corruption (class error rates of 10% and 30%); in these cases, randomizing the splitting had a similar effect.

4.4.2. Observation and discussion

Perhaps surprisingly, removing a learning algorithm's splitting criterion can degrade accuracy less than increasing the size of the concept in the presence of attribute error. In Section 4.1 we saw that depending on error, concept accuracy can drop from 95% to 80% as the size of the concept increases, as in Figure 5(b). Yet Figure 9 shows that removing the splitting criterion results in an average accuracy degradation of just 5%, despite the fact that the splitting criterion has been considered an important design factor for algorithms such as ID3 and PLS1.[13]

If we consider the 5% effect of the splitting criterion as a unit of influence, then the size of a concept has three times the influence. In other words, the splitting criterion, over the range of choices we investigated (random selection versus PLS1's probabilistic criterion versus ID3's information criterion) influences accuracy only a third as much as the size of the concept (over its whole range: 0% to 100%). This influence unit provides a benchmark for other data characteristics. Comparing it with Figure 6(a) from Section 4.2, we see that the shape of a concept's peaks can affect accuracy as much as the splitting criterion.

Figure 9 also compares the effect of the splitting criterion with the effect of concept concentration. The benefit of the "informed" splitting criterion used by ID3 or PLS1 is quickly outweighed if we increase the number of peaks. ID3 or PLS1 behave worse when learning a concept having 10 peaks (90% accuracy) than a naive algorithm does when learning a concept having one or two peaks (95% accuracy).

Why is the splitting criterion less important than concept concentration? The answer involves the fact that a dispersed concept is hard to learn for greedy algorithms like ID3 and PLS1. Increasing the number of peaks increases the degree of attribute interaction, resulting in higher-order relationships among the attributes. In contrast, greedy algorithms consider just one attribute at a time, which is tantamount to considering only one-dimensional projections of the class membership function over instance space. But one-dimensional projections become blurred when attribute relationships are high-order, so a one-dimensional splitting criterion becomes less useful. One response is to use lookahead in the decision tree or instance-space partition being learned, but this requires exponentially more time; moreover, extreme attribute interaction requires more data, slowing learning still further.

5. Further investigation

The results we have reported suggest two kinds of continued research. One is to exploit further the idea of "concept as function" to refine concept characteristics related to learning behavior. The other research direction is to improve learning algorithms based on findings to date. We discuss these two directions in two separate sections.

5.1. Improving studies of learning behavior

Table 2 is incomplete and oversimplified. It is incomplete with regard to sampling dimensions; for example, it ignores ideas such as "near misses." The table is also oversimplified with regard to concept concentration, for which the number of peaks is only a course measure. Below we detail one example each of incomplete characterization and oversimplified characterization.

5.1.1. Boundary concentration

Training data may be selected randomly or they may be biased. Figure 10 shows one kind of sample bias, where data near the boundaries of the class are favored. Such "boundary concentration" may speed learning by sharpening distinctions.

This data characteristic could be considered as intelligent instance selection. Winston's (1975) original idea of a "near miss" was that the instance is a member of the class except for the value of a single attribute. Buchanan et al. (1987) refined this notion, and showed

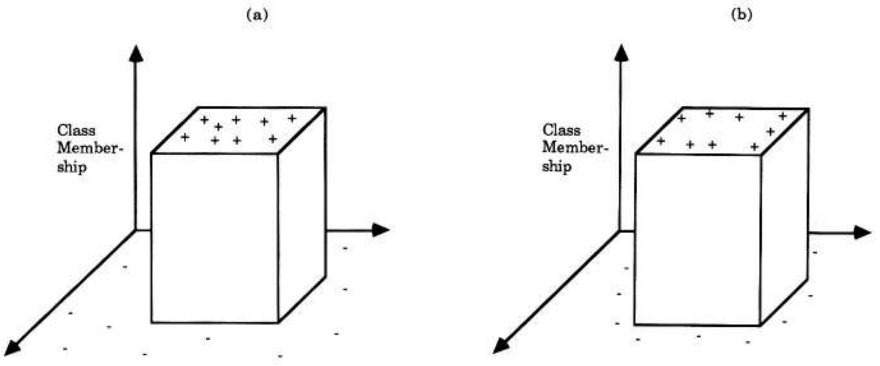

Figure 10. Boundary biasing of the training sample. In (a) the data are selected randomly, so they follow the whole population. In (b) the data are biased so that instances near the class boundaries are preferred.

that near misses and "near hits" are both helpful for learning. We suggest generalizing the notion still further, so that instead of near misses or near hits, we consider ***boundary concentration***, which is the selection of training examples from ***near the class boundary***. To implement this, our data generation program weights random selection to define the sampling distribution in terms of the class boundary. For example, in "quadratic" sampling, the distribution decreases with the square of the distance from the boundary.

Our preliminary experiments to find relationships between accuracy and boundary concentration have supported the results of Winston (1975) and Buchanan et al. (1987). Increasing the boundary concentration improved the accuracy. Quadratic (extreme) boundary concentration was better than linear (mild) concentration, which in turn was better than uniform distribution (no concentration). Because these results are tentative, we do not report details here.

How useful are such results? In our controlled experiments we could influence the sample at practically no cost, but to obtain a desired sample in practice, what resources are required? To answer such questions one must consider interactions of training with performance (e.g., Buchanan et al., 1987; Coles and Rendell, 1984; Mitchell, 1978; Sleeman, 1981). Although some of these studies show how the performance system can improve instance selection, finding boundary examples may be difficult, sometimes tantamount to locating the concept itself. Knowledge about the boundary is particularly complicated when the concept is dispersed in instance space, which brings us to another issue.

5.1.2. Concept dispersion

When we chose number of peaks p as a measure of concept dispersion in Section 2.3.2, we noted that this measure is limited: It does not distinguish peak sizes, shapes, or heights. Another measure, the Vapnik-Chervonenkis (VC) dimension[14] has been used to bound the number N of examples needed, as a function of the class of concepts to be learned. For example, for k-term DNF the best N known for any algorithm varies exponentially with the number of disjuncts k in the actual concept. More precisely, $N \propto n^k$, where n is the instance space dimensionality. The algorithm having this behavior changes representation to k-CNF (Ehrenfeucht et al., 1988; Pitt and Valiant, 1986).

We can relate this case to ours: Each term of a k-term DNF expression corresponds to a disjunct in the output of a program such as ID3 or PLS1. If we assume that each of p peaks requires roughly m disjuncts to describe, we need $k = m \times p$ terms. The theoretical result $N \propto n^k$ suggests a similar pattern for learning speed: $t \propto n^k$. In contrast, our experimental results for PLS1 appear more optimistic: In Section 4.3.2. we found that for fixed n, $t \propto (Nk)^{.57}$. We also saw that to maintain reasonable accuracy, we need to increase N at least linearly with k, which gives a rough estimate of $t \propto (ckk)^{.57} = C \times k^{1.2}$. For large k, however, this estimate is too optimistic; without change of representation the problem is known to be NP-hard (Pitt and Valiant, 1986). On the other hand, the theoretical upper bound may be too pessimistic if attribute construction is allowed.

By relating theoretical to experimental results we may profit in other ways. One issue is suitable measures for concept complexity. As a general measure, the number of peaks is limited, and so is the VC dimension. The VC dimension seems unsuited to practical problems because it is not easily measured and does not lead directly to algorithm improvement. Ideally, we would like a measure of concept complexity that is a good predictor of learning behavior, suitable for theoretical analysis, and directly related to practical problems.

The first two criteria are satisfied by the VC dimension. Are there other measures that would retain the analytical benefits but be more suited to practical work? For example, algorithms like ID3 have more trouble learning a concept (class membership function) as its variability increases, because then the instance space must be "chopped up" to a greater extent. This suggests a measure Δ = the sum (or integral) of the absolute value of the differences (or slopes) in the class membership function over the instance space. Hence a function having many changes of sign would have a higher Δ value than a monotonic function or a function having only one peak. The empirical results in this paper have been based on a simplification of Δ: The number of peaks in a concept is a coarse estimate of Δ because if all the peaks are similar, their number is proportional to Δ.

How can one take advantage of a measure such as Δ? It is straightforward to compute: As a by-product of standard empirical learning, one estimate of Δ sums the differences in the class membership values over adjacent regions (the Appendix describes how induction methods partition instance space into regions). Such an estimate is computationally inexpensive.[15] An estimate of Δ is valuable for attribute construction. This measure indicates the concept dispersion not only in the current instance space, but also in transformed instance spaces. Because a transformed instance space is the product of attribute construction, Δ can be used to evaluate the process. Such a measure may also be useful for theoretical analysis.

5.2. Changing representation to improve concept concentration

Whereas the previous section focused on measurement, this section considers algorithms. In particular, standard induction algorithms learn poorly when the concept is highly dispersed. For such a concept one needs to transform instance space so that the concept becomes more concentrated. Concentration improvement can be viewed as the purpose of hidden layers (the analogues of new terms) in research on neural networks, whose inputs are only low-level (Barron and Barron, 1988). In contrast, research involving symbolic techniques also applies domain knowledge, which could be considered "higher-level" information. The use of knowledge to transform representations for induction (and the use of induction to improve knowledge) is the theme of the following discussion.

5.2.1. Directly converting knowledge to representational bias

If increased concept concentration speeds learning and improves its accuracy, one might ***transform*** instance space so as to diminish peaks. One simple transformation involves only attribute scales: Nominal attributes can be converted to integer. PLS1 does this by trying all possible orderings of a nominal attribute's values, and choosing the "best" one. The best ordering is the one that gives the least variation in class membership (giving a monotonic increase) across the sequence of tentatively ordered attribute values, temporarily ignoring other attributes to project the membership function onto the attribute in question.[16] Finding the best ordering is probably the reason for PLS1's good behavior in Figure 7(a).

More generally, decreasing the number of peaks is difficult, though Table 2 specified one aspect of concept characterization that we have not used yet: higher-order regularity (see Figure 11). Although this characteristic is irrelevant for the systems we have considered, it can play an important role in new attribute construction. Instance spaces can be transformed into more abstract and useful spaces by exploiting higher-order regularity to relate peaks. Because diminishing peaks improves the behavior of inductive systems (Sections 4.2 and 4.3), peak merging becomes a purpose of attribute construction (Drastal and Raatz, 1989; Rendell, 1985, 1988, in press).

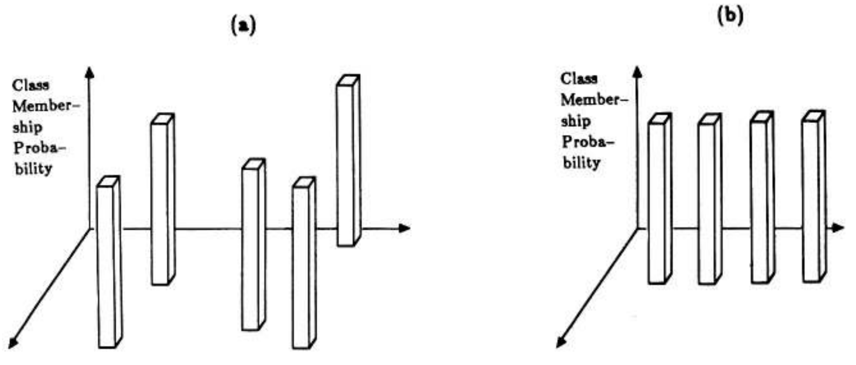
(a)
(b)
Class Member-ship Proba-bility
Class Member-ship Proba-bility

Figure 11. Higher-order concept regularity. In (a) the concept seems to be random, whereas in (b) it is more regular. A rule describing such regularities may merge and predict peaks.

Merging peaks involves "domain invariance." Domain invariance may be expressed using rules that ***merge*** observed peaks and ***predict*** unseen peaks. For example, many games and spatial problems exhibit symmetries—invariance of class membership under translation, rotation, reflection, and so forth. More specifically, suppose that the primitive instance space has peaks at regular intervals of length a, as in Figure 11. If the regularity appears along the x axis, a rule that expresses this invariance under translation is $u(x) = u(x \bmod a)$ (where u represents class membership), or more simply, $x \rightarrow x \bmod a$ (for an analogous example in a real problem, see Rendell, 1985). Systems that input such domain invariants allow peaks to be grouped together (Rendell, 1988, in press). Some of the peaks may be unrepresented by observed data, yet related to observed peaks through the knowledge.

Constructing new attributes by specifying (partial) knowledge has been shown to improve accuracy (Drastal, Raatz, and Munier, 1989; Matheus and Rendell, 1989). In fact, construction of attributes even without prior knowledge can improve accuracy (Pagallo, 1989; Pagallo and Haussler, 1988). As yet, however, such phenomena are relatively unexplored. We should explore the tradeoffs among (1) the amount of knowledge given to a construction system, (2) the resources consumed by the system, and (3) the improvement in learning behavior. Rendell (1986b) reported preliminary studies of this sort.

5.2.2. Inferring appropriate biases from domain theory

From the perspective of inductive learning, we might say that the purpose of domain knowledge is to speed the search for better attributes (those that diminish peaks in the transformed instance space). The role of knowledge is to constrain attribute construction to those attributes that merge and predict peaks (Rendell, 1988, in press). Unfortunately, the available knowledge is not always in a form that would merge peaks directly (it is not operational). Nevertheless, this idea provides what may be an important link between "similarity-based" empirical learning and "explanation-based" analytic learning. In one development, Drastal and Raatz (1989) have explored a system that merges peaks through inference from domain knowledge. Their scheme also addresses some problems of incomplete domain theories.

With the complication of inference, however, our research problems are compounded. Whereas the simpler type of knowledge of Section 5.2.1 directly expresses relationships among peaks, knowledge obtained through inference may be irrelevant for peak merging. In other words, many correct deductions are useless for the purpose of finding better attributes for induction. Hence, interplay between the deductive and inductive components becomes important, and one issue involves intelligent control of such components.

6. Summary

This paper has characterized and investigated the extensive role of data character as a determiner of system behavior in empirical concept learning. Careful study of data characteristics help to clarify their effects. Although we have examined just a cross section of realistic data, our experiments have shown quantitative relationships among data characteristics and learning behavior. Our experiments have supported some earlier results and have led to some new findings.

- As Quinlan (1986) showed, although class error (and to a lesser extent, attribute error) degrades learning, systems like ID3 and PLS1 can still learn fairly accurately. Nevertheless, the degree of degradation may depend on other data characteristics, such as the size of the concept (Section 4.1). With attribute error, a concept size of 50% leads to the poorest accuracy.
- Concept shape and especially concept concentration have significant effects. Increasing the number of peaks from one to twently degrades accuracy by 15% (Section 4.2). With large number of peaks, even compensation using proportionately more examples does not prevent degradation.
- In terms of both accuracy and time, learning systems such as ID3 and PLS1 become incapacitated when the peaks number in the thousands (Section 4.3). Yet important real-world problems exhibit many millions of peaks. Immense numbers of peaks occur when good representations are unavailable because humans lack understanding (e.g., protein folding) or cannot communicate the understanding (e.g., recognition and generation of sounds).
- Although the probabilistic or information theoretic splitting criteria used by algorithms such as ID3 and PLS1 have been considered important, their effects on accuracy are small compared with some concept characteristics. Even random splitting degrades accuracy by only five percent (Section 4.4).

Our basic approach in this paper has been to view concepts as functions. This characterization has yielded some general insights:

- Considering a concept as a ***function*** or ***surface over instance space*** leads to a characterization of concept complexity (shape, size, and concentration) that relates both to real-world problems and to learning behavior.
- Characterizing aspects of the concept or function allows artificial data generation, which is a valuable tool for exploring learning behavior by allowing controlled experimentation (cf. Langley, 1987).
- Because concept characteristics sometimes interact in unsuspected ways (Section 4.1), one must exercise caution when generalizing system behavior from one problem to another.
- One particularly important characteristic is concept concentration (Section 2.3). Experiments reported in the literature often use concepts that are highly compact (e.g., having just *one* peak), which are easy to learn.
- More "primitive," low-level attributes tend to produce class membership functions having many peaks. Because current learning algorithms are limited by the number of peaks they can handle, one purpose of constructive induction should be to transform the instance space so as to merge (and predict) peaks (Drastal and Raatz, 1989; Rendell, 1988, in press).

Data characteristics determine the behavior of learning algorithms. Empirical study of relationships between data characeristics and learning behavior can help discover which aspects of algorithms need augmentation. Moreover, appropriate characterization of data may suggest means for algorithm improvement.

Acknowledgments

Michael Hall, Pat Langley, Chris Matheus, Raj Seshu, and anonymous referees offered incisive comments on earlier drafts. We would like to thank Ross Quinlan for a copy of ID3/C4. Powell Benedict began work on the data generation program. This research was supported in part by grant IRI 8822031 from the National Science Foundation.

Appendix

To measure the effect of data we used the algorithms ID3 (Quinlan, 1983, 1986) and PLS1 (Rendell, 1983, 1986a),[17] which we describe here. These two learning systems represent a class of splitting algorithms that use probabilistic or information criteria, which have many versions and variations (e.g., Breiman et al., 1984; Clark and Niblett, 1989; Gams and Lavrac, 1987; Rendell et al., 1988). The reason for using this class of algorithms is that they behave well for the conditions investigated here; for empirical comparisons see Lavrac et al. (1986), O'Rorke (1982), and Rendell et al. (1988).

Both ID3 and PLS1 input data as attribute vectors. Both systems use probabilistic or information criteria to specialize hypotheses, and both represent hypotheses in a similar fashion. These programs output a rule or concept to classify unknown instances into one of two classes (positive or negative). Some variants, such as ASSISTANT (Gams and Lavrac, 1987) can learn more than one concept at a time.

Let us first consider the basic algorithms, and then discuss some of the variants. In these systems the entire space of possible instances begins as a single general description. The space is then split into two or more parts—those that have a greater likelihood of containing positive instances, and those that have a greater likelihood of containing negative instances. The splitting continues (using one attribute for each split) until some stopping criterion is satisfied. At each step, the attribute is chosen according to some probabilistic or information-theoretic standard.

As an example of the operation of ID3 or PLS1, if (small, triangle, red) were given as a negative instance of "toy ball," and (large, round, red) were a positive instance, then the instance space would be split in either the size or the shape dimension (see Figure 2). Since the algorithms normally consider one dimension at a time, instance space is partitioned into orthogonal hyperrectangles. The partition is refined until the differences in class membership probabilities no longer warrant any more splitting.

The ID3 algorithm

First we describe a basic version of Quinlan's (1979) original program, which continues splitting until the class membership probabilities are zero or one. This system represents a concept or rule as a decision tree. Each interior node of the tree is associated with one attribute; each arc leaving a node represents a value of an attribute (see Figure 2). The leaf nodes, which represent conjunctions of attribute values, are labeled as positive (+) or negative (−).

The tree-building algorithm begins with some training set of instances and compares the information measures of each attribute for the set. The attribute having the largest score becomes the label of the root node. For each attribute value, an arc is drawn from the root and labeled with this value. This process is then repeated at every intermediate node, until each node of the resulting partition is "pure" (i.e., has instances that are all "+" or all "−"). The basic learning algorithm is shown in Table 4.

Table 4. Two algorithms for partitioning instance space.

```
Let $S$ be a set of $n$ training instances.

Procedure ID3 ($S$)

While there are any exceptions in the decision tree,
    Select an impure node $r$.
    For each attribute $x$,
        Compute the value of the dissimilarity (information) criterion $d(S, x)$;
    For the attribute $x$ having the highest value of $d$,
        Extend the tree from node $r$ by drawing arcs labeled with the values of $x$.
        Return the resulting tree.

Procedure PLS1 ($S$)

Initialize the region set $R$ as the minimal region outlining $S$.
While regions exist whose splitting remains unexplored,
    Select from $R$ an untried region $r$.
    Let $d_{best}$ be 0 (initialize the stopping criterion).
    Choose a trial set of hyperplanes evenly-spaced and oriented with the axes
        (the trial set can be exhaustive or spaced out).
    For each trial hyperplane $H$,
        Use $H$ tentatively to split $r$ into two subregions $r_1$ and $r_2$.
        Compute the dissimilarity $d(H)$.
        If $d(H) > d_{best}$,
            Let $d_{best}$ be $d(H)$ and
            Record the information required to reproduce the split;
    If $d_{best}$ is greater than 0,
        Make the best split permanent.
        Add the new regions $r_1$ and $r_2$ to the set $R$.
Return the resulting region set $R$ (which is a piecewise-constant estimate
    of the class-membership function).
```

ID3 is quite fast.[18] Contributing to the speed of ID3 during classification and learning is the information criterion d, which tends to pick the "best" or most dissimilar splits, economizing data and minimizing attribute testing. For example, although there are a total of three attributes in Figure 2, a sufficient decision rule is a tree just two levels deep. Such a tree results from using the information criterion d, which can be defined by the expression:

$$d(x) = - \sum_{j=1}^{V} P_j \sum_{i=1}^{r} \frac{P_{ji}}{P_j} \log_2 \left[\frac{P_{ji}}{P_i} \right],$$

where P_j is the prior probability that attribute x has the j^{th} value, P_i is the prior probability that an example belongs to the i^{th} class, and P_{ji} is the probability that an example with the j^{th} value of x belongs to the i^{th} class. The values actually used are probability estimates from the training sample S.

The basic algorithm has been extended and modified in various ways. For example, when the number of instances is very large, a random subset or "window" helps to speed learning in some cases (Quinlan, 1983). The basic algorithm is given an initial window, and an overseeing algorithm retains exceptions to the window. The system then adds these exceptions to a new random sample to form a new window. This procedure is repeated until there are no exceptions to the rule. In the version of ID3 that we test in this paper (C4), this entire process is repeated ten times, and the tree with the smallest number of nodes is selected.

Other improvements of ID3 extend its applicability. Although the original program was designed for nominal attributes only, recent derivatives allow ordinal scales (Quinlan, 1986). Another extension permits the classification of instances into probability classes instead of pure binary classes (Quinlan, 1987a). Still another modification incorporates a measure of confidence in the splitting criterion d for dealing with uncertain data, based on the chi-square statistic (see Quinlan, 1986). These last two additions make ID3 similar to PLS1.

The PLS1 algorithm

The ***probabilistic learning system*** PLS1 consists of several components (Rendell, 1983, 1986a), one of which partitions instance space like ID3. Here we apply the name "PLS1" just to that module.[19] The algorithm accepts instances of known class membership, and based on their frequency, divides the instance space into mutually exclusive regions or "probability classes." As shown in Figure 12, this representation is a piecewise constant approximation (c) of the general (graded) class membership function (b). For classification, (c) can be converted to a Boolean (logic) representation (a) or to a decision tree that orders attributes like ID3.

Like ID3, PLS1 also uses specialization. It represents input instances as points in a k-dimensional space and creates orthogonal hyperrectangles by inserting boundaries parallel to instance space axes. Each hyperrectangle r is annotated with two values: (1) the probability u of finding a positive example within r,[20] and (2) an error measure e of u. These

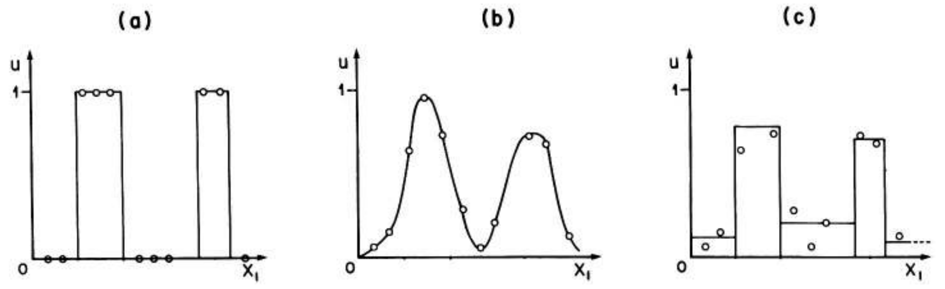

Figure 12. Alternative concept representations over a one-dimensional instance space. (In general this space is multidimensional.) In (a), the concept is assumed to be binary-valued; in (b), it is graded or probabilistic; in (c), it is approximated for easy learning.

annotated hyperrectangles (r, u, e), called ***regions***, are like nodes of a decision tree annotated with probability and error measures (see Figure 2 and compare Quinlan, 1987a). PLS1 allows disjunctive concepts in a manner similar to later verions of ID3.

Also like ID3, PLS1 chooses boundaries based on an informed splitting criterion. If u_1 and u_2 are the two class membership probabilities for a tentative region dichotomy, with e_1 and e_2 their error factors, then the dissimilarity is

$$d = |\log u_1 - \log u_2| - t \times \log(e_1 e_2).$$

The constant $t \geq 0$ expresses the degree of confidence. One source of error is the finite sample size n of training instances: the error factor e varies as $1 + 1 / \sqrt{n}$, where n is the sample size of a region. Larger values of d mean more assured dissimilarity. The dissimilarity measure d is also a stopping criterion whose effect can be compared to that of the chi-square statistic in some versions of ID3 (Quinlan, 1986). Table 4 summarizes the PLS1 algorithm, which is much like ID3 in terms of expected speed.[21]

Abbott (1987), Rendell (1986b), and Rendell et al. (1989) have compared PLS1, ID3, and related algorithms. If we compare ID3/C4 and PLS1 in light of Figure 2, we see that the two algorithms are similar, although they differ in some striking ways. For example, the splitting criteria are different. Also C4 prunes its decision tree to eliminate noise, whereas PLS1 faces that problem by splitting only if the statistical significance is high (and never prunes). These differences suggest that running ID3/C4 and PLS1 on the same data may improve our perspective about the effects of system design versus data character. Section 4 reports the results of such runs.

Notes

1. Complexity analysis of algorithms such as ID3 and PLS1 shows that their speed depends linearly on the product of the number of data, the number of attributes, and the number of disjuncts in the output concept (Quinlan, 1983; Rendell, 1983).

2. Our approach favors concepts expressed in disjunctive normal form (DNF). We picture an instance space with conjunctions representing local regions, and disjunctions collecting sets of regions. In contrast with conjunctive normal form (CNF), we could say that DNF gives a "spatial" or "human" representational bias (Lakoff and Johnson, 1980).
3. An important issue is change of representation involving new attributes. A new representation would result in a different function over a different description space (see Rendell, 1988, in press).
4. "Neighborhood" is a standard term in calculus and measure theory. The neighborhood of a point comprises the points close to it, given some notion of distance.
5. Uneven distributions of positive versus negative examples can also occur in learning evaluation functions. For example, PLS1 learned an evaluation function H to solve new problems (Rendell, 1983). Since H tended to improve search, it naturally favored the production of positive examples, which were then used in another (more focused) round of learning. The training sample became biased toward positive instances of the concept.
6. PLS1 is a complex system (Rendell, 1983), part of which corresponds to ID3; here we call just that part PLS1.
7. Rendell et al. (1989) speak of ***specialization*** algorithms as those that start with the whole instance space and proceed to partition it into more refined pieces (decision tree nodes or instance space regions). In contrast, ***generalization*** algorithms start with positive examples as seeds and extend them to other positive examples. If one views induction as clustering (of class membership probabilities), specialization algorithms split whereas generalization algorithms agglomerate (Anderberg, 1973).
8. Specialization or splitting algorithms are faster because they do less computation. They simply chop up the instance space, leaving a partition whenever they stop. In contrast, when generalization algorithms extend positive examples (seeds), they need to avoid negative examples, which is expensive. After extension of several seeds, additional computation ensures a partition (non-redundancy). Comparing well-known generalization algorithms with their counterpart specialization algorithms results in remarkable differences, as shown and discussed by Rendell et al. (1989).
9. Variations on the basic accuracy measure include different weightings for a "false positive" (incorrectly classifying a negative instance as a positive one) versus a "false negative" (incorrectly classifying a positive instance as a negative one). These distinctions can be important in critical applications such as disease diagnosis.
10. An error figure of x% in Figure 5(b) means attribute values were corrupted as follows: First, a random sample of x% of the instances was selected. Then, for each of these instances, a random number of attributes was chosen, each of whose values was replaced with a random value within the allowed range (which might have been the correct value).
11. Clark (personal communication) has formally related attribute error to class error.
12. Except for the fact that most experiments have involved binary concepts, there is no compelling reason to convert graded data values to binary. In other words, the instances could have graded class membership, but we have not investigated this case.
13. Note that this assessment concerns accuracy only. Since this paper was originally submitted, Mingers (1989) has shown a related result. With respect to conciseness, our experiments do not assess the splitting criterion, which reduces the size of the concept expression (Mingers). Conciseness can speed learning, especially in incremental situations.
14. The VC dimension of a class of concepts $\{H\}$ in hypothesis space X is the largest d such that there exists a set $S \subseteq X$ of size d, so that for all $T \subseteq S$, some concept in $\{H\}$ describes T consistently and completely.
15. Using an even cheaper estimate of concept dispersion, our programs are currently computing the sum of absolute differences between the mean class membership and the membership value of each region.
16. Becaue PLS1 orders attribute values one attribute at a time, the cost increases only linearly with the number of attributes. Unfortunately the effectiveness of this simple technique decreases as the number of peaks increases, because multiple peaks interfere with the one-dimensional method used by the program. The ordering algorithm is also limited in another way: Trying all possible orderings of an attribute's values becomes expensive as the number of values increases, because permutations increase factorially.
17. PLS1 is actually a set of algorithms, one of which was said to "cluster," because it clustered uniform values of probabilities. To simplify, we simply call the clusterer *PLS1*.
18. The outermost loop is repeated once for each of the m nodes in the final tree. The next loop is repeated for each of the k attributes. Finally, the time for the computation of criterion d varies linearly with the number of instances n. Hence the overall time complexity is $O(kmn)$.

19. The full version of PLS1 is designed for incremental or dynamic learning (Rendell, 1983). It performs various operations, including updating and refinement of probability estimates.
20. The true probability u is approximated by count ratios of the number of positive instances in a hyperrectangle to the total number of instances in that hyperrectangle.
21. PLS1 has a time complexity of $O(kmn)$, where k is the number of attributes, m is the final number of regions, and n is the number of instances.

References

Abbott, A.L. (1987). Cohesion methods in inductive learning. *Computational Intelligence, 3,* 267–282.

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

Barron, A.R., and Barron, R.L. (1988). Statistical learning networks: a unifying view. *Proceedings of the 20th Interface Symposium on Statistics and Computing* (pp. 192–203). Reston, VA: American Statistics Association.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth.

Buchanan, B.G., Rissland, E.L., Rosenbloom, P.S., Ng, H.T., and Sullivan, J. (1987). *The role of intelligent instance selection in learning systems: The near miss.* Unpublished manuscript, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA.

Clark, P., and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3,* 261–283.

Coles, D., and Rendell, L.A. (1984). Some issues in training learning systems and an autonomous design. *Proceedings of the Fifth Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 99–102). Toronto, Canada: Canadian Information Processing Society.

Cover, T. (1965). Geometrical and statistical properties of systems of linear equations with applications to pattern recognition. *IEEE Transactions on Electronic Computing, 14,* 326–334.

Devijver, P.A., and Kittler, J. (1982). *Pattern recognition: A statistical approach.* Englewood Cliffs, CA: Prentice Hall.

Dietterich, T.G., London, B., Clarkson, K., and Dromey, G. (1982). Learning and inductive inference. In P.R. Cohen and E.A. Feigenbaum (Eds.), *The handbook of artificial intelligence.* Los Altos: Kaufmann.

Draper, N.R., and Smith, H. (1981). *Applied regression analysis.* New York: Wiley.

Drastal, G., and Raatz, S. (1989). *Empirical results on learning in an abstraction space* (Technical Report DCS-TR-248). New Brunswick, NJ: Rutgers University, Department of Computer Science.

Drastal, G., Raatz, S., and Meunier, R. (1989). Induction in an abstraction space: A form of constructive induction. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 708–712). Detroit, MI: Morgan Kaufmann.

Ehrenfeucht, A., Haussler, D., Kearns, M., and Valiant, L.A. (1988). A general lower bound on the number of examples needed for learning. *Proceedings of the Workshop on Computational Learning Theory* (pp. 139–154). Boston, MA: Morgan Kaufmann.

Gams, M., and Lavrac, N. (1987). Review of five empirical learning systems within a proposed schemata. In I. Bratko and N. Lavrac (Eds.), *Progress in machine learning: Proceedings of the Second European Working Session on Learning.* Wilmslow, England: Sigma Press.

Haussler, D. (1986). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence, 36,* 177–221.

Holte, R.C., Acker, L.E., and Porter, B.W. (1988). Concept learning and the problem of small disjuncts. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 813–818). Detroit, MI: Morgan Kaufmann.

Hogg, R.V., and Craig, A.T., (1965). *Introduction to mathematical statistics.* New York: Macmillan.

Hunt, E.B., Marin, J., and Stone, P.J. (1966). *Experiments in induction.* New York: Academic Press.

Kearns, M., Li, M., Pitt, L., and Valiant, L.G. (1987). Recent results on boolean concept learning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 337–352). Irvine, CA: Morgan Kaufmann.

Lakoff, G., and Johnson, M. (1980). *Metaphors we live by.* Chicago: University of Chicago Press.

Langley, P.A. (1987). A general theory of discrimination learning. In D. Klahr, P. Langley, and R. Neches (Eds.), *Production system models of learning and development.* Cambridge, MA: MIT Press.

Lavrac, N., Mozetic, I., and Kononenko, I. (1986). *An experimental comparison of two learning programs in three medical domains.* Unpublished manuscript, Computer Science Department, University of Illinois, Urbana, IL.

Matheus, C.J., and Rendell, L.A. (1989). Constructive induction on decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 645–650). Detroit, MI: Morgan Kaufmann.

Michalski, R.S. (1983). A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann.

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

Mitchell, T.M. (1978) *Version spaces: An approach to concept learning.* Doctoral Dissertation, Department of Electrical Engineering, Stanford University, Stanford, CA.

Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1,* 47–80.

O'Rorke, P. (1982). *A comparative study of inductive learning systems AQ11P and ID3 using a chess endgame test problem.* (Technical Report No. UIUCDCS-F-82-899.) Urbana, IL: University of Illinois, Department of Computer Science.

Pagallo, G. (1989). Learning DNF by decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 639–644). Detroit, MI: Morgan Kaufmann.

Pagallo, G., and Haussler, D. (1988). *Feature discovery in empirical learning.* (Technical Report No. UCSC-CRL-88-08.) Santa Cruz, CA: University of California, Computer Research Laboratory.

Pitt, L., and Valiant, L. (1986). *Computational limitations on learning from examples* (Technical Report TR-05-86). Cambridge, MA: Harvard University, Aiken Computation Laboratory.

Quinlan, J.R. (1979). Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert systems in the microelectronic age.* Edinburgh, Scotland: Edinburgh University Press.

Quinlan, J.R. (1983). Learning efficient classification procedures and their application to chess end games. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann.

Quinlan, J.R. (1986). The effect of noise on concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Quinlan, J.R. (1987a). Decision trees as probabilistic classifiers. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 31–37). Irvine, CA: Morgan Kaufmann.

Quinlan, J.R. (1987b). Simplifying decision trees. *International Journal of Man-Machine Studies, 27,* 221–234.

Rendell, L.A. (1983). A new basis for state-space learning systems and a successful implementation. *Artificial Intelligence, 20,* 369–392.

Rendell, L.A. (1985). Substantial constructive induction using layered information compression: Tractable feature formation in search. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 650–658). Los Angeles, CA: Morgan Kaufmann.

Rendell, L.A. (1986a). Induction, of and by probability. In L.N. Kanal and J. Lemmer (Eds.), *Uncertainty in artificial intelligence.* Amsterdam: Elsevier Science Publishers.

Rendell, L.A. (1986b). A general framework for induction and a study of selective induction. *Machine Learning, 1,* 177–226.

Rendell, L.A. (1988). Learning hard concepts. *Proceedings of the Third European Working Session on Learning* (pp. 177–200). London: Pitman.

Rendell, L.A. (1989). Comparing systems and analyzing functions to improve constructive induction. *Proceedings of the Fifth International Machine Learning Workshop* (pp. 461–464). Ithaca, NY: Morgan Kaufmann.

Rendell, L.A. (in press). Learning hard concepts: Framework and rationale. *Computational Intelligence.*

Rendell, L.A., Cho, H.H. and Seshu, R. (1989). Improving the design of similarity-based rule-learning systems. *International Journal of Expert Systems, 2,* 97–133.

Samuel, A.L. (1963). Some studies in machine learning using the game of checkers. In E.A. Feigenbaum and J. Feldman (Eds.), *Computers and thought.* New York: McGraw-Hill.

Simon, H.A., and Lea, G. (1974). Problem solving and rule induction: A unified view. In L. Gregg (Ed.), *Knowledge and cognition.* Potomac: Erlbaum.

Sleeman, D.H. (1981). A rule-based task generation system. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 882–887). Vancouver, Canada: Morgan Kaufmann.

Tou, J.T., and Gonzalez, R.C. (1974). *Pattern recognition principles*. Reading, Massachusetts: Addison-Wesley.

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

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