USING CONCEPT LEARNING FOR KNOWLEDGE ACQUISITION

Ian H. Witten and Bruce A. MacDonald

Knowledge Sciences Institute, Department of Computer Science
The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4

ABSTRACT

Although experts have difficulty formulating their knowledge explicitly as rules, they find it easy to demonstrate their expertise in specific situations. Schemes for learning concepts from examples offer the potential for domain experts to interact directly with machines to transfer knowledge. Concept learning methods divide into similarity-based, hierarchical, function induction, and explanation-based knowledge-intensive techniques. These are described, classified according to input and output representations, and related to knowledge acquisition for expert systems. Systems discussed include candidate elimination, version space, ID3, PRISM, MARVIN, NODDY, BACON, COPER, and LEX-II. Teaching requirements are also analyzed.

The difficult process of knowledge acquisition (KA) has emerged as a key technology underpinning the creation of expert systems. Unfortunately there is no all-encompassing, unified theory of how knowledge is acquired, and probably never will be. Consequently, practical KA systems must strive to provide a variety of different services, hoping to cover the requirements for many diverse domains of expertise.

Although experts have trouble formulating their knowledge explicitly as rules, they find it easy to demonstrate their expertise in specific situations. This paper reviews the potential contribution of published schemes for learning concepts from examples to knowledge acquisition for expert systems. The next section identifies a number of roles which learning might play in KA systems. Following that we discuss the meaning of concept learning, classify methods for doing it, and describe alternative representations for what is learned. The next sections present several schemes for learning, structured according to the classification (see Table 1). Then we consider the role of the teacher, for in the present primitive state of the art, concept learning systems need someone who not only has an analytical understanding of the problem domain, but also is acquainted with some of the internal workings of the system itself. Finally we discuss how each scheme meets the requirements for knowledge acquisition, and suggest areas where more work is needed to render the results of concept learning research accessible to practical KA systems.

Only techniques based on clearly defined algorithms are described here. In general, this means ones that we know have been replicated, in essence if not in detail, by researchers other than their inventors or colleagues. This eliminates a number of successful systems which have not been defined publicly in enough detail to be reproduced by others.

THE ROLE OF LEARNING IN KNOWLEDGE ACQUISITION

What would it be like to use a KA system that was capable of learning? The target expert system's knowledge base will contain facts, inference rules, and possibly some representation of the system's goals. Accordingly, there are a number of rather different opportunities for learning.

Acquiring facts

Rote learning of facts is a mundane requirement expected of all KA systems. The expert states facts explicitly and the system must store, index, and retrieve them as appropriate. For example, the user might say "Here are facts about fungal infections. There are three main types: a, b, and c. ...".

More challenging is the acquisition of procedures. The target system may be expected to execute various action sequences to acquire or process data relevant to the case in hand. The expert may prefer to communicate procedures by entering examples of their execution, instead of formulating an imperative program which covers all cases. By generalizing action sequences, the system is able to perform automatically tasks such as reading data from an instrument, formatting it, and activating appropriate analysis routines.

Learning rules

If the expert can articulate the general rules that embody his expertise, the process of knowledge transfer reduces to rote learning. For example, the expert might enter "All fungal infections are treated with drug p, q, or r; the explanation being that these drugs all create an environment inhospitable to fungi". However, it has long been recognized that experts find it very hard to formulate their rules explicitly.

Rules may be learned from selected test cases, if the expert can give examples which make the "how to" information clear. For instance, "This is an example of how to select a treatment for fungal infections of types a and b: skin infection symptoms are x, y and z, so you prescribe p and q". The more explicitly the expert can indicate the pertinent conditions surrounding an example, the less generalization is required of the system. Existing elicitation techniques give the expert the tools to study examples himself and extract the important factors.

Rules might be learned from hosts of examples. Many expert systems are designed to replace or augment a manual operation. It may be possible to extract rules by processing past cases and analyzing them for similarities. Such examples may be preselected by an expert to cover all major possibilities with little duplication. Alternatively they may constitute a random sampling of cases.

Active learners ask questions. The system might analyze the knowledge it has acquired, seeking alternative interpretations of cases it has encountered. When alternatives are found, it asks for more information, perhaps by creating a test case which discriminates between them and presenting it to the expert for judgement.

Learning what to do

While an expert system's goals are often implicit, or stored in the form of unanalyzable text strings, real experts understand their goals and can reason about them in a wider context. Moreover, they know their limitations, when to refer a case to another expert or seek a second opinion, and so on. The expert may be able to teach the system metaknowledge about its own goals and capabilities; for example, "Your aim is to take inputs of the type skin infection and produce outputs of the type treatment and explanation".

Learning from users

Real experts continually improve their performance on the job, by analyzing new situations brought to them for advice or judgement, and learning from them (Hawkins, 1983). They exploit these cases to acquire new knowledge, to increase their understanding of what they already know, and to operationalize existing knowledge from theory into rules which can be recalled rapidly in future. They are capable of "sustained learning", the integration of substantial new knowledge into a large established knowledge base on a continual, ongoing basis.

CONCEPT LEARNING

Current knowledge acquisition techniques take care of routine housekeeping, permit rote learning of whatever can be communicated explicitly, and are able to elicit relatively simple rules based on the attributes of example cases (eg Boose & Bradshaw, 1987). But they do not permit the inference of rules which operate on more powerful representations (eg nested structures), nor can they learn from cases presented to them by non-expert users. Methods of concept learning may be able to overcome these limitations, although the present state of the art is primitive and offers ideas rather than well-developed algorithms to the KA toolbox. Concept learning systems take examples — which experts *can* provide — and create more general descriptions, often in the form of rules, which expert systems need.

What is meant by "concept learning"? Dictionary definitions of "concept" are remarkably vague, but have in common the abstract idea of a class of objects, particularly one derived from specific instances or occurrences. "Learning" also is a very broad term, and denotes the gaining of knowledge, skill, and understanding from instruction, experience, or reflection — in other words, "knowledge acquisition" by people. Notwithstanding the grand and sweeping nature of these ideas, the notion of concept learning as used in AI is much more constricted and specific. We will take it to denote *the acquisition of structural descriptions from examples of what is being described.*

Others have defined terms such as "generalization" (Schank *et al*, 1986), "inductive learning" (Michalski, 1983), and "inductive modeling" (Angluin & Smith, 1983) in almost identical ways. Not only are these all used to mean much the same thing, but what is learned is sometimes called a "generalization", a "description", a "concept", a "model", a "hypothesis". The field has not yet been successful in developing a satisfactory technical vocabulary; which term one favors seems purely a matter of taste.

Concept learning implies the acquisition of descriptions which make the structure of generalizations explicit. When a person learns a new concept, what she gains is knowledge that she can use in a rich variety of different ways: apply it to a particular case, reflect on it, tell it to a friend, criticize it, relate it to other ideas, and so on. For a person, learning is not simply a matter of acquiring a description, but involves taking something new and integrating it fully with existing thought processes. Existing programs which embody concept learning are not nearly so capable! However, they do acquire descriptions which are explicit in the sense that they can be communicated economically (for example, in the form of rules) and could plausibly support a variety of different kinds of reasoning. This orientation rules out, for example, connectionist models of learning, which embed knowledge in high-dimensional numerically-parametrized spaces, making learning into a process of weight adjustment. Since this does not result in concept descriptions that can be communicated economically or reasoned about, we will not discuss it further.

Mechanisms

There are almost as many paradigms for machine learning as there are systems. In fact, the variety of different, evocative, connotation-laden words used to describe simple mechanisms is one of the biggest problems in coming to grips with the field. Methods have been reported for learning by *analogy, being told, debugging, discovery, doing, examples, experimentation, exploration, imitation, instruction, observation, rote,* and *taking advice.* The apparent richness and variety of these approaches may give a misleading impression of a field teeming with fruitful techniques, with a selection of well-defined methods for tackling any given problem. Not so.

Here we distinguish the classes of concept learning mechanisms given below, and summarized in Table 1. While many taxonomies are possible, this one is intended to support distinctions that are particularly relevant to KA applications.

- *Parameter learning,* where the task is to optimize numerical parameters, often by detecting regularities in noisy situations. It assumes a particular description from the start, learning only the parameters of the model. Generally, large numbers of examples are used to allow incremental, hill-climbing style optimization and to overcome the effects of noise. We do not address parameter learning here.
- *Similarity-based learning,* where the space of concept descriptions is delineated in advance and searched for concepts which best characterize the structural similarities and/or differences between known examples.
- *Hierarchical learning,* where the concept description language is augmented as each concept is learned, so that new descriptions can build on old ones.
- *Function induction,* which constructs a function that accounts for a set of observed input-output tuples.
- *Knowledge-intensive learning,* which tackles the problem of relating new information to an existing body of knowledge. This area is as yet too undeveloped to attempt a taxonomy. Significant approaches are *explanation-based learning,* where a substantial chunk of information is learned by intensive analysis of just one example, and *learning by discovery,* where the system operates autonomously, performing

experiments to enhance its knowledge base.

Concept representation

Another important dimension of classification is how systems represent what they learn. Knowledge representation has always been a central topic in AI, and Barr & Feigenbaum (1982, pp.143-222) identify the representation schemes of *logic, procedural representations, semantic networks, production systems, direct (analogical) representations, semantic primitives,* and *frames and scripts.* It is worth pointing out that a data structure is not knowledge, any more than an encyclopaedia is. A representation of facts or rules only becomes knowledge when used by a program to behave in a knowledgeable way.

The requirement to acquire knowledge automatically from teacher or environment and integrate it with what is already known means that only the simplest representations are used by programs for machine learning. Separate representations are required for input examples and output concepts. Representations used for input include

- attribute vector (eg see Figure 1)
- combination of predicates (eg see Figure 2)
- nested term (eg see Figure 6)
- input-output tuples (eg see Figure 6);

and for concepts,

- vector of generalized attributes (eg see Figure 1)
- combination of predicates (eg see Figure 2)
- decision tree (eg see Figure 3)
- production rule (eg see Figures 3, 4)
- concept tree or semantic network (eg see Figure 5)
- recursive functional relationship (eg see Figure 6)
- nested numeric expression (eg see Figure 6)
- polynomial or other constrained numerical function (eg see Figure 6).

Table 1 shows the input and concept representations for the methods reviewed below.

Many learning methods apply to examples which can be expressed as vectors of attributes. The values an attribute can have may be *nominal, linear,* or *tree-structured* (Michalski, 1983). A *nominal* attribute is one whose values form a set with no further structure. A *linear* attribute is one whose values are totally ordered; ranges of values may be employed in descriptions. A *tree-structured* attribute is one whose values are ordered hierarchically. Only values associated with leaf nodes are observable in actual objects; descriptions, however, can employ internal node names where necessary. A nominal attribute is a special case of tree-structuring with no levels other than root and leaves; a linear attribute can be represented as a tree structure of intervals. Figure 1 shows an example of each kind, along with a sample object and a sample concept description which includes it.

Attribute vectors are not powerful enough to describe situations where each example comprises a "scene" containing several objects. The classic example is Winston's (1975) *arch* concept, defined as three blocks, two having the attributes required of columns and the third having those required of a lintel, with the columns set apart from each other and each supporting the lintel. Objects are characterized by their attributes. Moreover, pairwise (or more complex) relations may exist between objects. Such relations can be described by functions which, like attributes, may be nominal, linear, or tree-structured. For example, the result of a function *relative-position* which takes two objects as arguments could be a pair of linear values *x-distance, y-distance.* Alternatively it could be tree-structured, as illustrated in Figure 2. Objects and concepts in such domains are characterized by combinations of predicates.

Method	System	Input representation	Concept representation	Teacher
similarity based				
version space	(many)	attribute vector	vector of generalized attributes	supplies examples one at a time; results are insensitive to order, searching is not
	(many)	combination (invariably conjunction) of predicates	combination of predicates	
decision rules	ID3	attribute vector	decision tree	supplies examples all at once; examples may be noisy
	PRISM		production rules	
hierarchical	MARVIN	nested term	concept tree	supplies examples and classifies ones suggested by system
	ALVIN		semantic net	
function induction	MARVIN	input-output tuples	recursive functional relationship	
	NODDY		nested numeric expression	supplies examples one at a time
	BACON		product of linear and derivative operators	supplies table of experimental data
	COPER	input-output tuples + dimensions	polynomial of suitably transformed arguments	supplies initial data, performs experiments as suggested
knowledge-intensive				
explanation-based	LEX-II	conjunction of predicates	production rules	supplies examples one at a time
discovery-based	AM	domain simulation	frames containing rules	can suggest directions to explore

Table 1 Methods, systems, and data representation

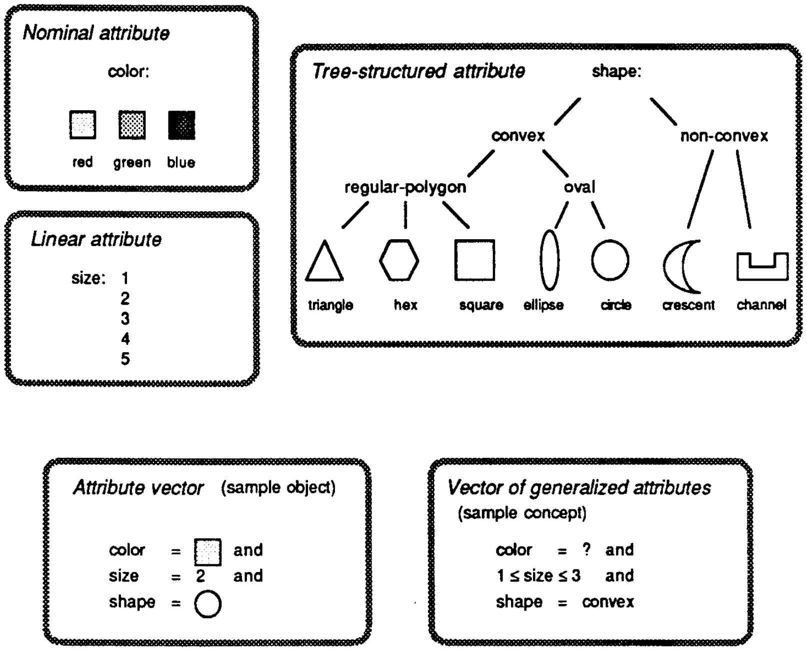


Figure 1 Attribute domain (adapted from Haussler, 1987b)

SIMILARITY-BASED LEARNING

Given a set of objects which represent examples and counterexamples of a concept, a similarity-based learner attempts to induce a generalized description that encompasses all the positive examples and none of the counterexamples. Interesting general issues include the conditions under which the procedure converges to a single description, whether the system can know that it has converged, whether the final concept may depend on the order of presentation of examples, and whether the training set is expected to be exhaustive (so that the concept description just discriminates between the examples presented) or representative (in which case some "generalization" is expected).

Candidate elimination and version spaces

A number of methods stem from the early experiments of Bruner *et al* (1956), who presented subjects with cards on which were drawn various figures differing in size, shape, color, etc, and indicated whether they were positive or negative examples of a concept. Attempts to inject some realism into this artificial task indicated that subjects behave differently when they can relate the various attributes to the real world. However, experimental studies continued to be preoccupied with information-processing aspects of the problem.

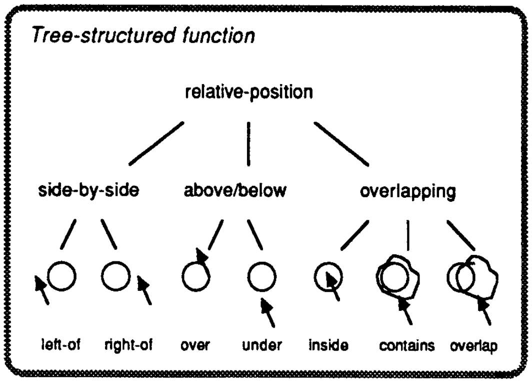
In situations where objects can be described by a finite vector of attributes, the inductive problem of creating an appropriate description can be transformed into a deductive one by circumscribing the language in which descriptions are expressed and searching for that description (or set of descriptions) which fits the examples given. The "version space" approach to concept learning (Mitchell, 1982) postulates a language in which objects are expressed, a language in which concept descriptions are expressed, a predicate which determines whether a particular object matches a particular description, and a partial ordering on descriptions (interpreted as generalization/specialization). Given a set of positive and negative examples of a target description, the problem is to find all descriptions that are consistent with the examples. This set is called the "version space".

In principle, the problem can be solved by enumerating candidate descriptions and striking out those which do not fit the examples that are presented. A positive example rules out all descriptions that it does not match, while a negative one eliminates those it does match. This is called the "candidate elimination" method. As each example is encountered, the set of remaining descriptions shrinks (or stays the same). If only one is left, it is the target description. If several are left, they may still be used to classify unknown objects. An unknown object which matches all remaining descriptions is included in the target; if it fails to match every one it is excluded by the target description. Only when it matches some descriptions but not others is there ambiguity; in this case if the classification of the object were known it would cause the version space to shrink.

Candidate elimination does not use the partial ordering on descriptions. The essential insight of the version space method is that the descriptions in the version space need not be stored explicitly, but can be inferred from its upper and lower boundaries in the partial order. The upper boundary comprises all members that are maximally general, the lower boundary all that are maximally specific. The boundaries can be updated incrementally as new examples are encountered. The upper boundary is initialized to comprise all maximally general descriptions (there may be several). In principle, the lower one could be initialized to comprise all maximally specific descriptions; since there may be a large number of these it is instead initialized to include the first example only (which must be positive).

Vectors of attributes. The candidate elimination and version space methods apply simply and directly when each object is described by an attribute vector, and the performance of version space algorithms in such domains has been studied extensively. It is well known that allowing disjunctions in the description language causes the version space to explode. Even with purely conjunctive concepts, however, if the number of attributes is large the size of one version space boundary can grow exponentially in the number of examples. This and other complexity results are discussed by Haussler (1987a).

Combinations of predicates. In structural domains, each example (or counter-example) comprises a "scene" containing several objects. Part of the problem in matching a scene with a structural description is determining an appropriate mapping between objects in the scene and those specified in the description. Figure 2 shows an example scene containing three particular objects — a, b and c. It also gives a description which refers to two variable objects x and y. Binding x to a and y to c shows that something matching the description is contained within the scene. But it is necessary to clarify how the description is to be interpreted. It might be



Combination of predicates
(sample object)

color(*a*) = red and
size(*a*) = 2 and
shape(*a*) = circle and
under(*a, b*) and
over(*a, c*) and
color(*b*) = green and
size(*b*) = 4 and
shape(*b*) = crescent and
over(*b, a*) and
over(*b, c*) and
color(*c*) = red and
size(*c*) = 3 and
shape(*c*) = square and
contains(*c, a*) and
under(*c, b*)

Combination of predicates
(sample concept)

shape(*x*) = circle and
1 ≤ size(*x*) ≤ 3 and
shape(*y*) = convex and
relative-position(*x, y*) = inside

Figure 2 Structure domain (adapted from Haussler, 1987b)

prefaced by a statement such as

there exist different x, y with ...

giving what Stepp (1987) calls "contains" semantics. This is distinguished from "is" semantics, where the scene must have those parts mentioned in the description and only *allowable* extras. Under "is" semantics, optional parts (eg vines, spiders, dust, on an arch) would be observed in positive examples, and as more were seen the concepts used to describe optional parts would become more and more general. Negative examples containing ornaments would prohibit adornments of that kind.

Haussler (1987b) has obtained several theoretical results which indicate that, even with the simpler "contains" semantics, extreme computational complexity can be involved in working with version spaces of structured objects. The problems of testing whether one description is more general than another, and whether a description contains a given scene, are both NP-complete if we allow arbitrarily many objects in scenes and arbitrarily many variables in descriptions. Furthermore, the boundaries of the version space can become arbitrarily large. Even when each scene has exactly two objects, no binary relations are defined between the objects, and each object has only two-valued attributes, the size of boundaries can grow exponentially in the number of examples if there are sufficiently many attributes. Moreover, under these conditions it is NP-complete to determine if there exists any concept consistent with a given set of examples. These pessimistic results may indicate why even though Winston's early machine learning system used structured objects, relatively few subsequent projects have worked with structured examples (Stepp, 1987).

Creating decision rules

Another way to tackle the problem of concept learning is to use the training set to construct a decision tree or collection of rules which discriminates positive from negative examples. In the case of a tree, the root specifies an attribute to be selected and tested first, then depending on its value subordinate nodes dictate tests on further attributes. The leaves are marked to show the classifications of the objects they represent. For two-class problems these classifications are simply "positive" and "negative", but it is easy to accommodate situations where more than two classes must be distinguished. In the case of production rules, the training set is used to construct a set of rules which can be interpreted by an expert system in standard forward- or backward-chaining manner. In both cases, it is normally assumed that all examples are available and can be processed together to construct the tree or rules.

Decision trees. Procedures for top-down induction of decision trees were developed many years ago to model human performance in concept learning (Hunt, 1962; Hunt *et al*, 1966). It was found that subjects focus on positive instances, and they are assumed to build up a decision tree in order to categorize instances. These early programs sought an attribute which distinguished positive from negative instances; if there were none, they selected a frequent characteristic of the positive instances and made this the first test in the decision tree. The set of instances that passed the test were treated as a subproblem in exactly the same way.

The ID3 algorithm (Quinlan, 1986a) is a popular modern incarnation of these ideas. It does not attempt to model human performance but rather aims to minimize the number of nodes in the tree. It uses an information-theoretic heuristic to determine which attribute should be tested at each node, looking at all members of the training set which reach that node and selecting the attribute that most reduces the entropy of the positive/negative decision. By the nature of the algorithm, correct performance is guaranteed on the training set. Any "generalization" achieved depends purely on the representation of objects and the choice of training set. By seeking the simplest decision tree, ID3 finds the most economical way to classify the examples given.

Because of its simplicity and elegance, the ID3 procedure has been replicated and studied extensively. Having performed well in the domain of chess end-games (where the problem is more to summarize a large body of decisions economically than to generalize), it was soon adopted for a number of commercial applications. When presented with noisy data, the basic ID3 algorithm constructs huge decision trees which reflect the detail of every example seen. The resulting uneconomical representation is not a suitable vehicle for generalization, and several research efforts are underway to develop improved algorithms which copy with noise (eg Quinlan, 1986b; Niblett & Bratko, 1986).

An example of operation of ID3 (from Cendrowska, in press) concerns an adult spectacle wearer who consults an optician with a view to purchasing contact lenses, bringing along her spectacle prescription. We assume that, from the optician's point of view, this is a three-category problem with four factors *a*, *b*, *c*, and *d* to take into consideration. The complete decision table is given in Figure 3, and ID3.produces the tree shown.

Production rules. For incorporation into expert systems, the decision tree is usually converted into production rules such as those of Figure 3. These rules are supposed to represent the professional judgement of the optician, which he is likely to state in forms such as

> This patient is not suitable for contact lens wear because her tear production rate is reduced
> (rule 1: a=1 → decide 3).
>
> This patient can only be fitted with hard contact lenses because she is astigmatic. As she is young and has a normal tear production rate, hard lenses are not contraindicated
> (rule 6: b=1 & c=2 & d=2 → decide 1 and rule 7: a=1 & b=2 & c=2 & d=2 → decide 1).

However, Cendrowska (in press) has noticed that even when a decision tree has the minimal number of nodes, there may be redundancy in the rules extracted from it. One manifestation of this is that there is no single rule which corresponds to the second example above; rules 6 and 7 between them cover the situation. Because of the existence of rule 6, rule 7 could be generalized to

> rule 7': a=1 & c=2 & d=2 → decide 1

without affecting the outcome. The additional clauses in the ID3 rules could cause an expert system to call for redundant tests to be made, in this case by demanding a value for *b* even when *a*, *c*, and *d* are known and together determine the result.

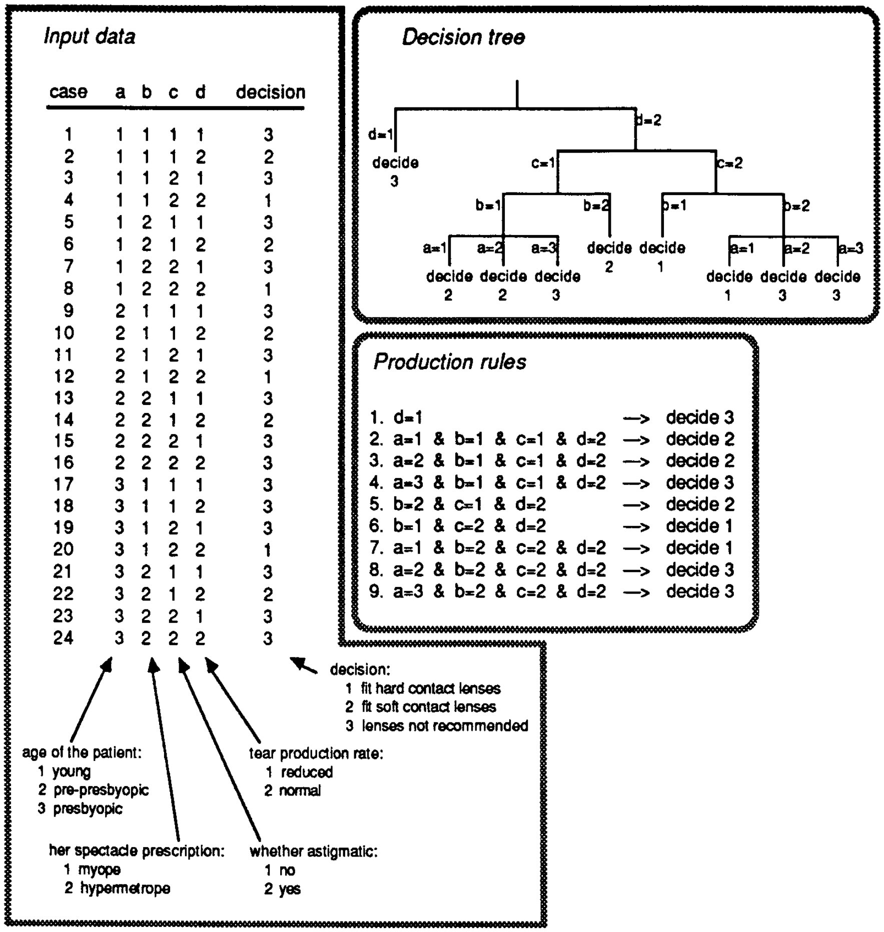
Rather than attempting to clean up the production rules generated from the ID3 decision tree, it is possible to dispense with the tree altogether and generate rules directly. The PRISM procedure (Cendrowska, in press) generates those shown in Figure 4 for the example problem. Although the number of rules is the same as for ID3, they are simpler and more general — six have had redundant terms removed. On a larger problem with a training set of 647 examples, PRISM found 15 rules containing a total of 48 terms where ID3 found 52 rules containing 337 terms. Both sets correctly classified all members of the training set.

Limitations of similarity-based learning

Similarity-based methods are the most mature and immediately applicable for the purpose of knowledge acquisition. However, it has often been pointed out (eg see Schank *et al*, 1986, for a recent discussion) that these methods hardly relate to what people do in real life.

First, they use the wrong sorts of concept, representing an object like "chair" or "table" as a certain configuration of sticks and boards. Many everyday concepts do not even involve a common function, let alone a common appearance (Wittgenstein, 1958). Second, they embed examples in an artificial scenario where objects are lifted out of context, isolated, cleaned up, and presented individually to the system. Third, they assume the existence of a teacher who decides when to create a new concept and when to modify an old one. In real life, the environment neither identifies nor labels instances. Fourth, systems tend to embody the implicit assumption that categories are defined by a collection of features that are individually necessary and collectively sufficient. Real instances have indefinite numbers of features, some explicit, some inferred.

Attempts have been made to go some way towards meeting such objections by designing new learning mechanisms which are embedded in richer knowledge-based systems. These are discussed below under "Knowledge-intensive learning".



Input data

case	a	b	c	d	decision
1	1	1	1	1	3
2	1	1	1	2	2
3	1	1	2	1	3
4	1	1	2	2	1
5	1	2	1	1	3
6	1	2	1	2	2
7	1	2	2	1	3
8	1	2	2	2	1
9	2	1	1	1	3
10	2	1	1	2	2
11	2	1	2	1	3
12	2	1	2	2	1
13	2	2	1	1	3
14	2	2	1	2	2
15	2	2	2	1	3
16	2	2	2	2	3
17	3	1	1	1	3
18	3	1	1	2	3
19	3	1	2	1	3
20	3	1	2	2	1
21	3	2	1	1	3
22	3	2	1	2	2
23	3	2	2	1	3
24	3	2	2	2	3

decision:
1 fit hard contact lenses
2 fit soft contact lenses
3 lenses not recommended

age of the patient:
1 young
2 pre-presbyopic
3 presbyopic

tear production rate:
1 reduced
2 normal

her spectacle prescription:
1 myope
2 hypermetrope

whether astigmatic:
1 no
2 yes

Decision tree

Production rules

1. d=1 —> decide 3
2. a=1 & b=1 & c=1 & d=2 —> decide 2
3. a=2 & b=1 & c=1 & d=2 —> decide 2
4. a=3 & b=1 & c=1 & d=2 —> decide 3
5. b=2 & c=1 & d=2 —> decide 2
6. b=1 & c=2 & d=2 —> decide 1
7. a=1 & b=2 & c=2 & d=2 —> decide 1
8. a=2 & b=2 & c=2 & d=2 —> decide 3
9. a=3 & b=2 & c=2 & d=2 —> decide 3

Figure 3 Decision tree and rules produced by ID3
(adapted from Cendrowska, in press)

Production rules

1. d=1 —> decide 3
2. a=1 & c=1 & d=2 —> decide 2
3. a=2 & c=1 & d=2 —> decide 2
4. a=3 & b=1 & c=1 —> decide 3
5. b=2 & c=1 & d=2 —> decide 2
6. b=1 & c=2 & d=2 —> decide 1
7. a=1 & c=2 & d=2 —> decide 1
8. a=2 & b=2 & c=2 —> decide 3
9. a=3 & b=2 & c=2 —> decide 3

Figure 4 Rules produced by PRISM
(adapted from Cendrowska, in press)

HIERARCHICAL LEARNING

It would be attractive if learning systems could build upon concepts already learned, and use them as components in newly-constructed descriptions. This might allow learning to be sustained over an extended period of time, instead of being done on a one-off basis. A mechanism for attempting this is reviewed below. However, it cannot support indefinitely sustained learning, for two reasons. First, the search space involved grows exponentially with the number of known concepts. Much more attention must be paid to the use of domain knowledge to curtail this growth before such systems become practical. Second, in order for learning to be truly sustained the teacher has to be able to correct misconceptions after they have occurred. Shapiro's (1983) techniques for debugging declarative programs may possibly provide a handle on this problem.

Examples which are presented to these systems may be atomic names rather than vectors of attributes or structures built from components. Concepts are stored in the form of a logic program. Each concept may be described by several clauses, which are interpreted as disjuncts. Each clause may have several terms which are interpreted as conjuncts. Terms are either primitives provided in the system, or other concepts; thus the structure is hierarchical.

Examples do not have to be atomic. They can have nested structure and, when used recursively, are capable of representing arbitrary list structures. If non-atomic examples are presented, they are decomposed to see if they can be expressed as compositions of other concepts. The process permits recursion, so that the concept of a number as a list of digits, as well as the procedure for appending such lists, can be taught by example.

Concept trees. A system called MARVIN learns hierarchical structures of concepts (Sammut & Banerji, 1983, 1986). Given an example, it searches for alternative ways to express it in terms of known concepts. Instead of awaiting further examples from the teacher to constrain its choice of expressions, however, it selects a tentative description, synthesizes a "crucial object" to determine if that description is a correct generalization, and asks whether it too is an example of the concept. If so, the tentative description is accepted as a valid generalization and attempts are made to generalize it further. If not, the tentative description is specialized to rule out the negative example and a new crucial example is synthesized. Thus a sequence of generalizations and specializations is made, each being tested by asking the teacher to classify a crucial example. When all possibilities for generalization have been exhausted, the description is stored and the teacher is asked for another example of the concept.

As an example, suppose MARVIN already has the concept *fish*, which includes *flying fish*, *cod* and *trout* as illustrated in the upper part of Figure 5. Upon learning that *trout* is a *swimmer*, it will hypothesize that all *fish* are *swimmers* and test this hypothesis by asking whether *cod* is a *swimmer*. Since it is, and no further generalizing hypotheses can be made, the system will record the fact that *fish* implies *swimmer* and await a new example from the teacher. Given *whale* as a new example, the concept is expanded by adding as a disjunct the fact that *whale* implies *swimmer*.

Semantic nets. The method implemented by MARVIN breaks down if the concepts are not structured and taught in the form of a tree. For example, the lower part of Figure 5 illustrates the *swimmer* concept in a richer, partially-ordered, knowledge base representing an animal kingdom. To create a strictly hierarchical structure it would be necessary to have used subclasses such as *swimming flightless bird* and *swimming aerial bird* when teaching the original *bird* concept. This is illustrated in the middle part of the Figure for a fragment of the animal kingdom. Notice that in the non-hierarchical representation on the right, each object can be represented by an attribute vector. Real-world knowledge bases are likely to fall between the extremes of hierarchical and completely flat representations.

It is desirable that new concepts (such as *swimmer*) can be added to general knowledge structures (like that illustrated at the bottom of Figure 5) without interfering in any way with previously learned concepts. This requires a more sophisticated method of selecting examples to be presented to the teacher. Given, for instance, that *man* is a *swimmer*, it is not immediately clear whether this is because it is a *mammal* and all *mammals* are *swimmers*, or because it is a *flyer* and all *flyers* are *swimmers*, or because it is both *mammal* and *flyer*, or purely by virtue of being a *man*. The question would be further complicated if there were superclasses of the *man* concept. Each hypothesis must be investigated. If a crucial object exists within the current tentative hypothesis but not within any competing ones, it is presented to the teacher. If such an object does not exist, the alternative hypotheses are investigated to discover the most general valid description (Krawchuk, 1987).

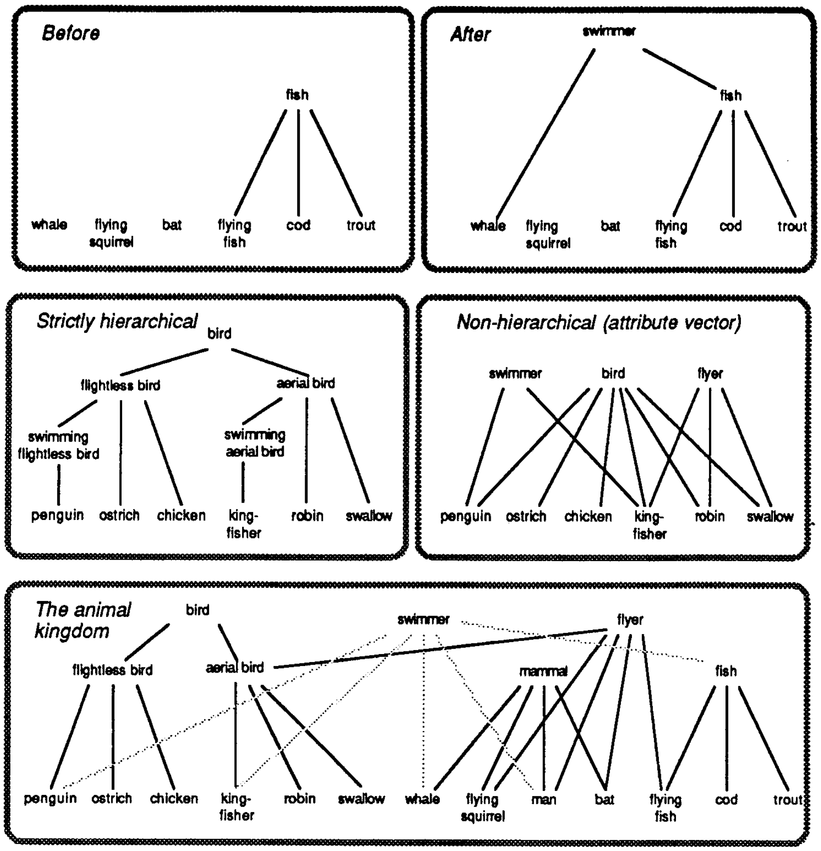
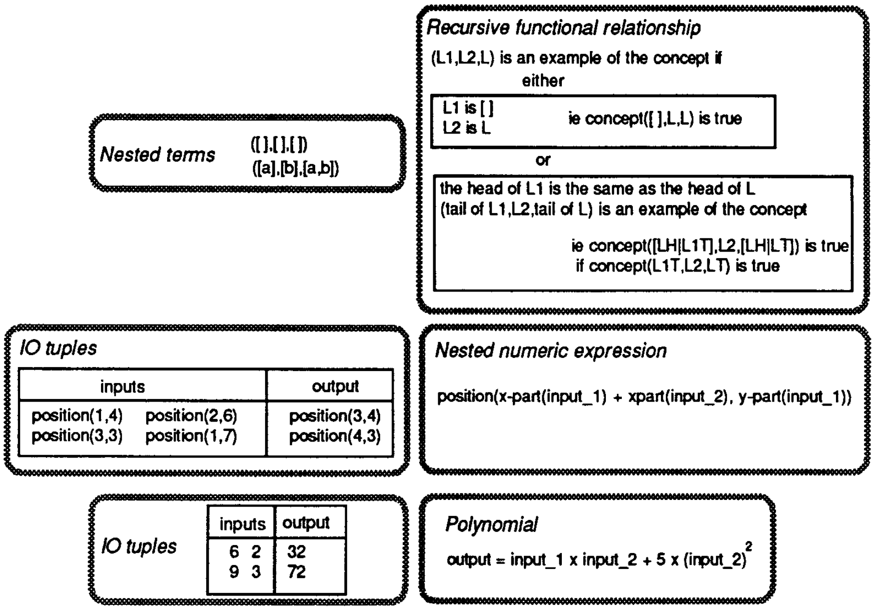


Figure 5 Teaching swimming in the animal kingdom



Recursive functional relationship

(L1,L2,L) is an example of the concept if

either

L1 is [ ]
L2 is L

ie concept([ ],L,L) is true

or

the head of L1 is the same as the head of L
(tail of L1,L2,tail of L) is an example of the concept

ie concept([LH|L1T],L2,[LH|LT]) is true
if concept(L1T,L2,LT) is true

Nested terms

([ ],[ ],[ ])
([a],[b],[a,b])

IO tuples

inputs		output
position(1,4)	position(2,6)	position(3,4)
position(3,3)	position(1,7)	position(4,3)

Nested numeric expression

position(x-part(input_1) + xpart(input_2), y-part(input_1))

IO tuples

inputs		output
6	2	32
9	3	72

Polynomial

output = input_1 x input_2 + 5 x (input_2)^2

Figure 6 Examples of function induction

FUNCTION INDUCTION

Given a number of input-output (IO) tuples, function induction creates a function which computes the output of each pair from its input. In general this is very difficult, although for numerical functions whose form is known there is a well-developed body of theory in numerical analysis to calculate parameters of the relationship. Most artificial intelligence work on function induction concerns structured functions which are at least partially non-numeric. It is hard to identify common approaches, except to say that a massive search space is involved and searching is relatively unguided, since there is little potential for heuristically evaluating competing functional expressions. A selection of systems are reviewed briefly below.

MARVIN can be used for function induction as well as for hierarchical learning. Examples of functions which have been learned include

- list manipulation: appending, finding the maximum, and so on;
- arithmetic on numbers represented as strings of binary digits;
- simple insertion sort.

The first part of Figure 6 shows examples of the "append" concept, and the resulting function description. MARVIN proceeds by producing successively better approximations of the concept, oscillating between over-generalization and under-generalization until the target is finally reached. Each approximation is "tested" by synthesizing a new object and asking the teacher if it is an example of the concept. MARVIN's greatest weakness is that it cannot invent new existentially quantified variables. This eliminates the possibility of learning many interesting procedures (including, for example, more efficient sorting algorithms such as quicksort).

Function induction is employed as a component of NODDY, a system for learning robot programs from traces of their execution (Andreae, 1984a, 1984b). When presented with a list of IO pairs gleaned from successive incarnations of the same robot procedure, and a list of known constants that can be used, it attempts to induce the IO function. The partially complete function is represented as an expression which contains a "gap". This expression is initialized to the identity function with the gap standing for the unknown function to be induced. A list of candidate operators is programmed in, and the gap is closed if a single operator is found which generates each output from the corresponding input. Moreover, if there is an operator which maps all IO pairs to the same constant, the gap can be closed by using that operator's inverse. Otherwise, a new set of expressions is constructed, each of which modify the gap by adding an operator to it. Along with each new expression, a new set of IO pairs is constructed by applying the new operator to the input values. After pruning duplicate expressions and expressions corresponding to inconsistent IO pairs, the algorithm repeats.

It proves surprisingly difficult to characterize the functions that this method can induce. They are of course limited to combinations of the operators provided. The algorithm is constrained to introduce no more than one new constant, to avoid searching the infinitude of functions like $f(x) = x+a+b$ where $a+b=c$. (Actually, functions may have several inputs, and a separate constant can be introduced for each input variable.) Although the search for functions is strongly constrained by typing variables and operator arguments, it is nevertheless still an incredibly expensive search, and NODDY attempts function induction only when other evidence suggests that a functional relation may exist, and simpler methods fail. The middle part of Figure 6 shows example IO tuples and the nested expression that NODDY generates.

One AI system which works on wholly numeric functions is BACON, which attempts to discover empirical scientific laws by inducing functions that account for observed data (Langley *et al*, 1983). It is given as input a table of values of dependent and independent variables, and looks for monotonic relations between them. If one is found, it is tested to see if it is linear. If not, the slope of the curve relating the two variables is calculated and added to the table as a new variable. The lower part of Figure 6 illustrates the input and concept descriptions. From appropriate tables of data, the system discovered various relations in physics (eg Snell's law, conservation of momentum, gravitation); adding a heuristic which finds common divisors allowed it to interpret correctly data on the results of various chemical reactions. More recent work on the system is aimed at making it able to deal with noisy data using a hill-climbing strategy (Langley *et al*, 1986).

A more structured approach to deducing physical laws is to use dimensional analysis in concert with observed data. Kokar's (1986) system COPER works with physical quantities, for instance 3 m, 5 kg/m^2, 7 s. It takes a set of numeric observations, along with the dimensions of the variables involved (eg m^{1}kg^{1}s^{-2} for force). It is also told which are dependent and which are independent variables. When given data representing the observations in appropriate experiments, the system uses dimensional analysis to tell whether the independent variables are complete (ie sufficient to determine the dependent variable's value uniquely). For example, given a set of data representing speed, time, and distance traveled by a falling body, the system can detect that a variable (g, the acceleration due to gravity) has been omitted. It then suggests experiments from which it can infer the dimensions of the missing variable (m/s^2).

Having determined the necessary arguments, COPER goes on to discover the exact functional relationship by searching the space of polynomial functions. Any function (fulfilling some formal conditions) can be approximated to arbitrary accuracy by a polynomial. However, alternative sets of arguments can be used for the functional relationship, and if the wrong set is chosen this may lead to an artificially complex form. For example, suppose area (m^2), time (s) and mass (kg) are the arguments. Even though complete they may result in an overly complex polynomial, for it may be necessary to approximate a square root to derive distance (m). Consequently after searching polynomials up to a certain complexity the system selects alternative bases and tries those instead. If none gives a satisfactory result (ie a good approximation), each base is reconsidered using more complex polynomials.

KNOWLEDGE-INTENSIVE LEARNING: EXPLANATION BASED

It is often argued that an explanation-driven process is essential for reasonable generalization. Correlations derived purely from empirical observations are much less convincing than a theory which explains them. Explanation-based generalization methods take a single example and deduce a general rule by relating it to an existing theory. They do not learn the theory. Indeed, since the general rule is already implicit in the theory, they learn nothing new. What they do is use examples to guide the operationalization of knowledge already implicitly known, so that it can henceforth be employed more efficiently. The knowledge operationalized pertains to the original example, and probably to many others.

The earliest example of explanation-based learning is LEX-II, which learns problem-solving heuristics in the domain of symbolic integration. It starts with a set of operators for performing integration, corresponding to a table of integrals and a set of techniques for integration (eg integrating by parts). With each operator are specified preconditions which indicate when it can validly be applied. Solving a problem consists of applying a series of operators to derive an equivalent expression containing no integrals. The heuristics that LEX learns indicate when it is useful to apply the operator to solve an integration problem. Given an example expression to integrate, LEX embarks on an extensive search to obtain an appropriate sequence of operators. It then traces the example back through the sequence, at each stage using the preconditions to generalize the expression as much as possible consistent with that operator still being applicable. The result is an integration sequence for a more general expression than the original example. Finally, the operators used are stored along with the generalized expression to which they apply, so that they can be quickly chosen if in future a matching expression is encountered.

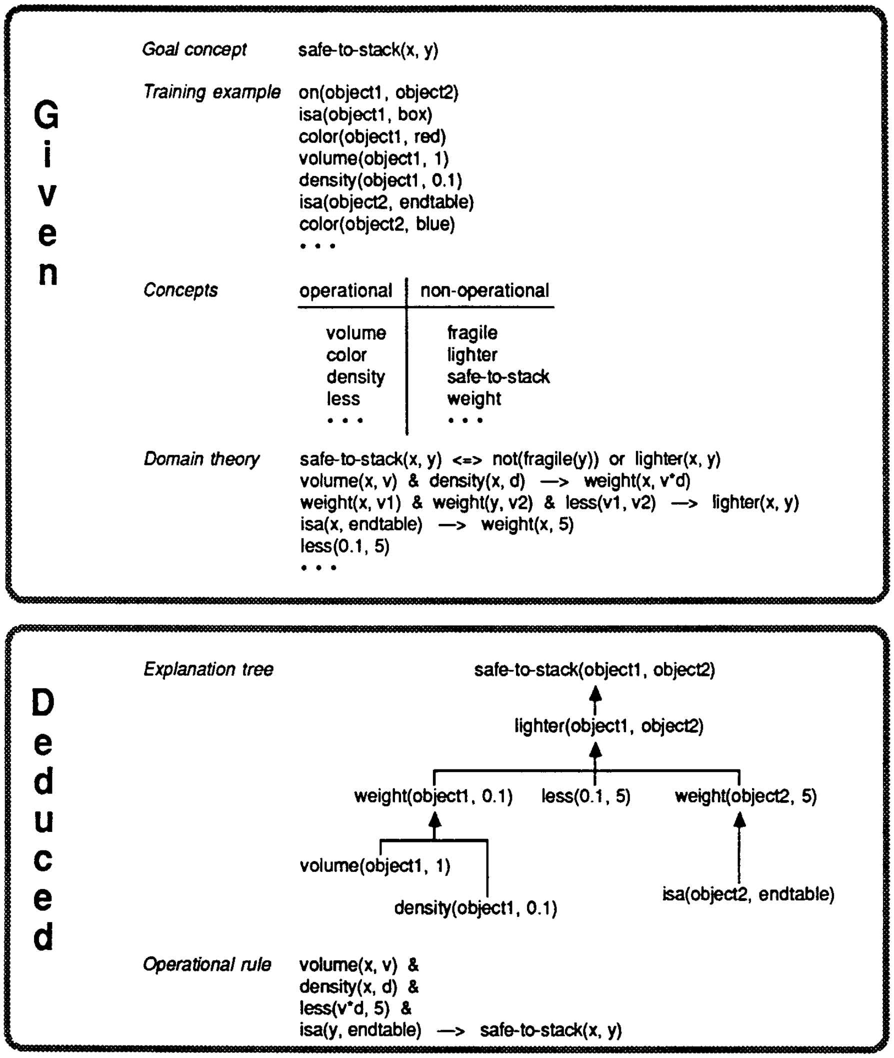
Figure 7 illustrates explanation-based learning in a blocks world (Mitchell *et al*, 1986). The goal concept is *safe-to-stack*, where

> it is safe to stack object x on object y if y is not *fragile* or x is *lighter* than y,

An example is given which says that it is safe to stack a particular box (whose color, density, volume are given as attributes) on a particular endtable. According to the domain theory, which is known in advance, endtables have a weight of 5 and can be deduced to be fragile. Moreover, the theory shows how to calculate weight from volume and volume, and how *lighter* is defined. The goal concept cannot be applied directly because *fragile* and *lighter* are not "operational" concepts; that is, they are not given directly but must be derived from the attributes of the objects through a proof procedure. Once a proof has been found for this particular example, however, the concepts involved can be generalized to produce the rule

> it is safe to stack object x on an *endtable* provided its *volume* times its *density* does not exceed 5.

Notice that the rule suppresses irrelevant details of the example (box, color) and identifies the relevant constraints on the other attributes (weight, density). It can be used to test rapidly whether objects are stackable



Goal concept safe-to-stack(x, y)

Training example
on(object1, object2)
isa(object1, box)
color(object1, red)
volume(object1, 1)
density(object1, 0.1)
isa(object2, endtable)
color(object2, blue)
• • •

Concepts

operational	non-operational
volume	fragile
color	lighter
density	safe-to-stack
less	weight
• • •	• • •

Domain theory
safe-to-stack(x, y) <=> not(fragile(y)) or lighter(x, y)
volume(x, v) & density(x, d) —> weight(x, v*d)
weight(x, v1) & weight(y, v2) & less(v1, v2) —> lighter(x, y)
isa(x, endtable) —> weight(x, 5)
less(0.1, 5)
• • •

Operational rule
volume(x, v) &
density(x, d) &
less(v*d, 5) &
isa(y, endtable) —> safe-to-stack(x, y)

Figure 7 Explanation-based generalization
(adapted from Mitchell et al, 1987)

on endtables.

Explanation-based generalization is a two-step process, first creating an explanation that distinguishes the relevant features of the examples from the irrelevant ones; and then analyzing it to determine the minimum constraints that are sufficient for the explanation to apply in general. Because the explanation is created for the given training example, it does not necessarily explain every possible example of the goal concept. The final concept is typically a specialization of the goal concept rather than a direct re-expression of it.

A number of extensions to the view of explanation-based learning presented here have been identified (DeJong and Mooney, 1986). First, the proof on which generalization is based need not be discovered by the unaided system; any hints which a teacher might be able to provide can be taken into account, giving the system the character of an apprentice. Second, concept learning often involves specialization as well as generalization, and the explanation-based approach can be used for this too. Third, in more complex domains it becomes necessary to consider how predicates might be generalized; otherwise spurious conditions could become embedded in the generalized rule. Finally, the method described operates only on attributes and should be extended to structured objects.

Explanation-based learning usually presumes the existence of a strong domain theory in which proofs can be constructed that show why a particular example is valid. Some experiments have been reported with a kind of explanation-based learning supported by a very weak theory (Pazzani, 1987a). The domain chosen is physical causality for everyday events, and rules include

> if an action on an object always precedes a state change for the object, then it may cause the state change;
>
> if two similar actions performed on an object by different actors have different results, the differing features of the actors may be responsible for the different results.

Such rules are used to construct "explanations" for example events; competing explanations are resolved by a preference ordering on the rules. The explanation is used to choose which features of the events should be held accountable for their classifications. Of course, subsequent examples may contradict the reasoning, in which case an alternative explanation is constructed using lower-priority rules.

KNOWLEDGE-INTENSIVE LEARNING: DISCOVERY BASED

Explanation-based learning only addresses part of the objections to similarity-based learning which were raised earlier. Isolated attempts have been made to meet what is perhaps the major shortcoming, that examples are lifted out of the world, cleaned up, and presented to the system by a teacher who makes all the important decisions about when to create new concepts. Still the most impressive are the AM and EURISKO systems of Lenat (Lenat, 1978, 1983; see in particular Lenat & Brown, 1984) which work on their own to discover interesting concepts in elementary mathematics and interesting heuristics. However, research in this area is quite immature and does not yet form a foundation for practical KA techniques.

TEACHING REQUIREMENTS

A concept learning system's practical utility obviously depends critically on its teaching requirements. Teaching differs from programming in that a teacher does not use a formal model of the learner whereas a programmer normally expects to know exactly how his instructions are going to be interpreted. In other words, a teacher adopts the *intentional stance* while a programmer adopts the *design stance* (Dennett, 1981). Ideally, therefore, to be a good teacher one need know nothing about the internal structure of the learner, nor about the representation it uses for concepts.

Moreover, experts who use concept learning for knowledge acquisition are unlikely to have a clear analytical understanding of the task domain — otherwise more explicit methods of knowledge transfer will probably prove more appropriate. In general, people find it difficult to translate their own expertise into explicit descriptions. Consequently concept learning systems should be able to work with the kind of examples that a teacher finds it natural to provide, ordered in a way that is natural to him.

The teaching requirements and conditions under which learning is to occur provide a useful dimension for classifying learning mechanisms. Anything that is learned must be communicated through the expert- or user-machine interface. For example:

- *How cooperative/skillful is the teacher?* This dimension ranges from no teacher at all (input coming raw from the environment) through a naive user, a domain expert, to a teacher who is prepared to program in a conventional programming language and thereby circumvent the need for learning. Must the teacher show all working, give "near-miss" examples, teach simple concepts before complex ones, and so on (Witten & MacDonald, 1987)?
- *How reliable are the examples?* The examples provided may be noisy or exact. Much work in concept learning assumes exact data so that structural regularities can be learned without the problems of statistical detection.
- *Is learning done incrementally or retrospectively?* It may be that the current description must be augmented as each example is encountered; alternatively it might be possible to process all examples together. Batch operation is infeasible for systems which attempt sustained learning. In the incremental case, is the final concept supposed to be independent of the order of examples?
- *How much knowledge must the system have already?* There is an oft-quoted aphorism that in order to learn something, you must nearly know it already. The amount of knowledge that a system already possesses about the problem domain critically affects the kind of things it can be expected to learn. Even similarity-based techniques embody substantial *a priori* knowledge in the form of languages in which examples and concepts are expressed.

Table 1 provides some indication of the teaching requirements for each method discussed.

Formal aspects

A basic theoretical distinction has been shown between presenting positive examples only, presenting both positive and negative examples, and allowing the learning system to choose examples itself and have them classified by an informant. In the first case, it is assumed that all examples are included eventually. In the second, it is assumed that every member of some universe which contains all possible descriptions is shown eventually; this allows an informant to be simulated by simply waiting until a selected example occurs in the presentation sequence. Under these conditions, Gold (1967) proved formally what one expects intuitively, namely that the second and third methods are more powerful than the first. They permit primitive recursive languages, which include the context sensitive, context free, and regular languages, to be learned eventually, whereas with positive presentations only, the inclusion of even one infinite language in the set of potential concepts can destroy learnability.

There is a curious theoretical result which relates to the sequence of examples presented. Gold (1967) showed that descriptions unlearnable from positive and negative examples *can* be learned under a special condition: the presentation sequence must be sufficiently regular. Moreover, positive examples alone suffice! For instance, descriptions from the class of recursively enumerable languages (which include the primitive recursive ones mentioned above, and more besides) are not normally identifiable even from positive and negative examples, nor indeed using a helpful informant who is prepared to classify arbitrary examples. But a suitable, highly contrived, presentation sequence of positive examples can render such descriptions learnable. In essence the learner can identify the algorithm used to select the examples, thereby identifying the set from which they are drawn. Here the importance of the teacher's selected example sequence is clearly reflected at the theoretical level. A good sequence of examples does more than speed the search, it makes learnable the otherwise unlearnable.

Felicity conditions

A skilled teacher will select illuminating examples himself and thereby simplify the learner's task. The benefits of carefully constructed examples were appreciated in the earliest research efforts in concept learning. Winston (1975) showed how "near misses" — constructs which differ in just one crucial respect from examples of a concept being taught — could radically reduce the search required for generalization. Confident that its teacher is selecting examples helpfully, a learning system can assume that any difference between an example being shown and its nascent concept is in fact a critical feature.

The notion of a sympathetic teacher has been formalized in terms of "felicity conditions", constraints imposed on or satisfied by a teacher that make learning better than from random examples (Van Lehn, 1983). One obvious condition is that the teacher should classify examples correctly, and not (intentionally or unintentionally) mislead the student. Another is that if the absence of something is important, the teacher should point it out explicitly. If in the classic example of the concept of "arch" it is necessary that a gap be left between the pillars, the gap should be explicitly labeled in examples. More generally, the teacher should show all work and avoid glossing over intermediate results. Examples should not by coincidence include features that might mislead the learner. Finally, the teacher should introduce one essentially new feature per lesson and not try to teach multiple differences at once — a similar condition to Winston's "near miss" approach.

Intermediate results, and other implicit parts of examples, are difficult for a system to learn since they necessarily involve a large search. The system must consider everything that *might* be absent, and the numerous relationships it may have with other parts of the description. Andreae's (1984a, 1984b) "justified generalization" addresses this problem by using the amount of justification associated with a conjecture to determine which of three different levels of search to undertake. When the evidence is sufficiently strong, it will expend considerable resources seeking implicit relationships between parts of a description.

Although it does not increase formal learning power, the possibility of a system constructing its own examples and having them classified by an informant has considerable potential to speed up learning and reduce dependence on the skill of the teacher. This potential can only be realized if the system is able to synthesize "crucial examples" which discriminate effectively between alternative hypotheses consistent with the information seen so far. This ability seems to be one of the chief distinguishing characteristics of people who are good learners. To take the generation of examples one step further, if there is an automatic way of classifying examples as positive or negative, a system can attempt to learn autonomously through the paradigm of "learning by discovery".

CONCLUSION

Many present applications of concept learning techniques to knowledge acquisition for expert systems fall into the area of learning classification rules. For example, Carter & Catlett (1987) describe how a training set of 350 applications for credit cards, each classified as accepted or rejected, can be processed by ID3 into a decision tree which classifies around 80% of test cases "correctly" (ie according to the judgement of a human expert). Since the input is an attribute vector and the situation is noisy, top-down decision methods such as ID3 and PRISM are well suited to the task.

The version space technique represents a bottom-up approach. Its advantages are firstly in producing a result that is independent of the order of presentation of examples, and secondly in dealing with structured domains that cannot be captured by simple attribute-value vectors. These are offset by the fact that the version space collapses if the input is noisy. Moreover, the amount of computation *does* depend on the order of presentation of examples, and is very high in structured domains.

Hierarchical learning systems offer considerable promise but again suffer from excessive computational complexity and have only been applied to toy domains. A major research effort is required to integrate them with other techniques and use domain knowledge to constrain the search for alternative hypotheses which account for the examples seen so far.

Methods for function induction are ripe for exploitation in expert systems. Existing systems illustrate how constrained search can find the simplest input-output relation that accounts for given input-output tuples, and some (eg NODDY's) are not limited to purely numerical relationships. COPER shows how dimensional analysis can serve to determine the relevant arguments to functions, and how alternative bases can be searched to find economical expression of polynomial functions. Whether the same idea can be applied to more generalized typing and dimensional constraints is at present an open question.

Explanation-based generalization seems an attractive component for any knowledge-based system. Expert systems can be viewed as knowledge-intensive programs as opposed to domain-independent general-purpose problem solvers. Much work is required to build and debug the knowledge base of an expert system. This may take the form of extracting and operationalizing appropriate knowledge from a general theory. If so, an alternative is to define the general knowledge needed for a problem-solver, but this is bound to be inefficient because it must search extensively for solutions. Explanation-based learning is able to operationalize just the knowledge needed to process typical examples (Pazzani, 1987b). This paradigm offers some promise for emulating the ability to learn from cases supplied by users, the hallmark of human expert behavior.

Finally, there are many opportunities for combining the various different learning methods. At the simplest, one can imagine a set of independent "learning experts", all receiving the same data, and reporting if and when they discover interesting structure. More interesting is the integration of different learning paradigms. Pazzani (1987a) describes an early experiment in which

- if an existing schema accounts for an event, it is indexed in memory by features which elaborate on that schema;
- otherwise, if existing schemata can be combined to explain it, a new schema is constructed using explanation-based learning;
- otherwise, if there are several similar examples, a new schema is created by similarity-based methods;
- otherwise, the event is simply added to memory.

Although this research is in a very early stage, it illustrates how different mechanisms can be brought into play depending on the knowledge available. Only in this way can we expect to build systems capable of sustained learning, continually expanding their knowledge base, drawing new lessons and insights from every example they encounter, the way real experts do.

ACKNOWLEDGEMENTS

We wish to thank Brian Gaines for suggesting that we write this paper and for constructive comments on an early draft. Financial support for this research has been provided by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Andreae, P.M. (1984a) "Constraint limited generalization: acquiring procedures from examples" *Proc American Association on Artificial Intelligence*, Austin, TX, August.

Andreae, P.M. (1984b) "Justified generalization: acquiring procedures from examples" PhD Thesis, Department of Electrical Engineering and Computer Science, MIT.

Angluin, D. and Smith, C.H. (1983) "Inductive inference: theory and methods" *Computing Surveys, 15* (3) 237-269, September.

Barr, A. and Feigenbaum, E.A. (Editors) (1982) *The handbook of artificial intelligence Volume II.* HeurisTech Press, Stanford, CA.

Boose, J. and Bradshaw, J.M. (1987) "Expertise transfer and complex problems" *Int J Man-Machine Studies, 26* (1) 3-28, January.

Bruner, J.S., Goodnow, J.J., and Austin, G.A. (1956) *A study of thinking.* Wiley, New York.

Carter, C. and Catlett, J. (1987) "Assessing credit card applications using machine learning" *IEEE Expert, 2* (3) 71-79, Fall.

Cendrowska, J. (in press) "PRISM: an algorithm for inducing modular rules" *Int J Man-Machine Studies.*

DeJong, G. and Mooney, R. (1986) "Explanation-based learning: an alternative view" *Machine Learning, 1* (2) 145-176.

Dennett, D.C. (1981) *Brainstorms.* Harvester Press, Brighton, Sussex.

Gold, E.M. (1967) "Language identification in the limit" *Information and Control, 10,* 447-474.

Haussler, D. (1987a) "Bias, version spaces, and Valiant's learning framework" *Proc 4th International Workshop on Machine Learning,* 324-336, Irvine, CA, June.

Haussler, D. (1987b) "Learning conjunctive concepts in structural domains" *Proc AAAI,* 466-470.

Hawkins, D. (1983) "An analysis of expert thinking" *Int J Man-Machine Studies, 18* (1) 1-47, January.

Hunt, E.B. (1962) *Concept learning: an information processing problem.* Wiley, New York.

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

Kokar, M.M. (1986) "Determining arguments of invariant functions" *Machine Learning, 1* (4) 403-422.

Krawchuk, B.J. (1987) "How to ask the right questions" Research Report, Department of Computer Science, University of Calgary.

Langley, P., Bradshaw, G.L., and Simon, H.A. (1983) "Rediscovering chemistry with the BACON system" in *Machine learning: an artificial intelligence approach,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 307-329. Tioga, Palo Alto, CA.

Langley, P., Zytkow, J.M., Simon, H.A., and Bradshaw, G.L. (1986) "Rediscovering chemistry with the BACON system" in *Machine learning Volume II,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 425-469. Morgan Kaufmann, Los Altos, CA.

Lenat, D.B. (1978) "The ubiquity of discovery" *Artificial Intelligence, 9,* 257-285.

Lenat, D.B. (1983) "EURISKO: a program that learns new heuristics and domain concepts" *Artificial Intelligence, 21,* 61-98.

Lenat, D.B. and Brown, J.S. (1984) "Why AM and EURISKO appear to work" *Artificial Intelligence, 23,* 269-294.

Michalski, R.S. (1983) "A theory and methodology of inductive learning" *Artificial Intelligence, 20* (3).

Mitchell, T.M. (1982) "Generalization as search" *Artificial Intelligence, 18,* 203-226.

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

Niblett, T. and Bratko, I. (1986) "Learning decision rules in noisy domains" in *Research and development in expert systems III,* edited by M.A.Bramer, pp 25-34. Cambridge University Press, Cambridge, England.

Pazzani, M.J. (1987a) "Inducing causal and social theories: a prerequisite for explanation-based learning" *Proc 4th International Workshop on Machine Learning,* 230-241, Irvine, CA, June.

Pazzani, M.J. (1987b) "Failure-driven learning of fault diagnosis heuristics" *IEEE Trans Systems, Man and Cybernetics, SMC-17* (3) 380-394, May/June.

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

Quinlan, J.R. (1986b) "The effect of noise on concept learning" in *Machine learning Volume 2,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 149-166. Morgan Kaufmann Inc, Los Altos, CA.

Sammut, C. and Banerji, R. (1983) "Hierarchical memories: an aid to concept learning" *Proc International Machine Learning Workshop,* 74-80, Allerton House, Monticello, IL, June 22-24.

Sammut, C. and Banerji, R. (1986) "Learning concepts by asking questions" in *Machine learning Volume 2,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 167-191. Morgan Kaufmann Inc, Los Altos, CA.

Schank, R.C., Collins, G.C. &A Hunter, L.E. (1986) "Transcending inductive category formation in learning" *Behavioral and Brain Sciences, 9* (4) 639-651, December.

Shapiro, E.Y. (1983) *Algorithmic program debugging.* MIT Press, Cambridge, MA.

Stepp, R.E. (1987) "Machine learning from structured objects" *Proc 4th International Workshop on Machine Learning,* 353-363, Irvine, CA, June.

VanLehn, K. (1983) "Felicity conditions for human skill acquisition: validating an AI-based theory" Research Report CIS-21, Xerox PARC, Palo Alto, CA, November.

Winston, P.H. (1975) "Learning structural descriptions from examples" in *The psychology of computer vision,* edited by P.H.Winston. McGraw Hill, New York, NY.

Witten, I.H. and MacDonald, B.A. (1987) "Concept learning: a practical tool for knowledge acquisition" *Proc Avignon 87 -- expert systems and their application,* 1535-1555, Avignon, France, May.

Wittgenstein, L. (1958) *Philosophical investigations.* Blackwell, Oxford, UK.