1

A Review of Machine Learning Methods

Miroslav Kubat
Ivan Bratko
Ryszard Michalski

1.1 Introduction

The field of machine learning was conceived nearly four decades ago with the bold objective to develop computational methods that would implement various forms of learning, in particular mechanisms capable of inducing knowledge from examples or data. As software development has become one of the main bottlenecks of today's computer technology, the idea of introducing knowledge into computers by way of examples seems particularly attractive and appealing to common sense. Such a form of knowledge induction is specially desirable in problems that lack algorithmic solutions, are ill-defined, or only informally stated. Medical or technical diagnosis, visual concept recognition, engineering design, material behavior, chess playing, or detection of interesting reqularities in large data sets are examples of such problems.

One of the vital inventions of artificial intelligence research is the idea that formally intractable problems can be solved by extending the traditional scheme

program = *algorithm* + *data*
to the more elaborate

Machine Learning and Data Mining: Methods and Applications,
Edited by R.S. Michalski, I. Bratko, and M. Kubat
© 1996 John Wiley & Sons Ltd

program = algorithm + data + domain knowledge

Applying domain knowledge, encoded in suitable data structures, is fundamental for solving problems of this kind. Anyone who has taken a course on artificial intelligence knows the power of production rules, frames, semantic networks, and uncertainty propagation in expert systems. Machine learning systems, too, profit from this idea.

Nevertheless, the use of knowledge only shifts the bottleneck from the programmer to the knowledge engineer, who has to elicit it from an expert and encode it into the system. The process of knowledge acquisition and encoding is, in any real-world application, far from being easy. For example, specialists in computer chess know that the brute-force approach has led to more powerful programs than the artificial intelligence methods because the knowledge needed to make the program beat a grandmaster is very difficult to formulate; grandmasters use their experience intuitively and are, in most cases, unable to convey it to an artificial intelligence system in the form of production rules or other representational systems. Chess textbooks are full of abstract concepts such as initiative, cooperation of pieces, weak pawn structures, good bishops, and well-protected king; and chess players need years of experience to develop their understanding. Such concepts usually lack precise definition and encoding them in a computer is very difficult.

Thus a tempting idea springs to mind: employ a learning system that will acquire such higher-level concepts and/or problem-solving strategies through examples in a way analogical to human learning.

Most research in machine learning has been devoted to developing effective methods to address this problem. Although progress has been slow, many significant results have been achieved. The main argument of the opponents of the field is now: if, in expert systems, programming has been replaced by knowledge-encoding, then in machine learning knowledge encoding is supposed to be replaced by induction from examples; however, available learning systems are not powerful enough to succeed in realistic domains.

The objective of this book is to demonstrate that in many practical domains the application of machine learning leads to useful practical results. We demonstrate on presented case studies that the field has reached the state of development in which existing techniques and systems can be used for the solution of many real-world problems.

Two groups of researchers need to be put together if the application of machine learning is to bear fruits: those that are acquainted with existing machine learning methods and those with expertise in the given application domain to provide training dat. This book aims at attracting attention of potential users from disciplines outside computer science. If it can arise their interest and make them consider an application of machine-learning to problems in their fields that resist traditional approaches, it will be worth the authors' effort.

To initiate non-specialists to the field of machine learning, this chapter surveys methods that are necessary to develop a general understanding of the field and have been used in the case studies presented in this volume. Section 1.2 discusses the meaning of a concept and its use as a basic unit of knowledge. It then briefly elaborates on issues related to the problem of concept representation in computer memory. Section 1.3 formulates the generic machine learning task as a search through the space of representations. Upon these basic considerations, Section 1.4 discusses two fundamental approaches to concept learning, thus preparing the ground for Section 1.5, which outlines more sophisticated methods based on a subset of first-order logic.

To deepen the reader's understanding of the general way of thinking of machine learning researchers, Section 1.6 describes some approaches to discovery. Section 1.7 briefly reviews two methods to save effort in building concept descriptions in rich representation languages—analogy and the use of examples themselves as knowledge representtion.

The next two sections, 1.8 and 1.9, briefly cover techniques that are usually not included in traditional symbolic machine learning texts but whose knowledge is, nevertheless, indispensable for any specialist in this domain: neural networks, genetic algorithms, and various hybrid systems. The objective of this introduction is to arm the reader with the basic knowledge necessary for forthcoming chapters, rather than to provide a comprehensive coverage of the discipline. For a more detailed coverage, the reader is referred to various books devoted to this field such as Michalski, Carbonell, and Mitchell, (1983, 1986), Kodratoff and Michalski (1990), Michalski and Tecuci (1994), Langley (1996), or Mitchell (1996).

1.2 The Machine Learning Task

The general framework for machine learning is depicted in Figure 1.1. The learning system aims at determining a description of a given concept from a set of concept examples provided by the teacher and from the background knowledge.

Concept examples can be positive (e.g., a dog, when teaching the concept of a mammal) or negative (e.g., a scorpion). Background knowledge contains the information about the language used to describe the examples and concepts. For instance, it can include possible values of variables (attributes) and their hierarchies, predicates, auxiliary syntactic rules, subjective preferences, and the like. The learning algorithm then builds on the type of examples, on the size and relevance of the background knowledge, on the representational issues, on the presumed nature of the concept to be acquired, and on the designer's experience.

An important requirement is that the learning system should be able to deal with imperfections of the data. Examples will often contain a certain amount of *noise*—errors in the descriptions or in the classifications. For instance, a classification error will be if 'scorpion' is classified by the absent-minded teacher as 'mammal'. Moreover, examples can be incomplete in the sense that some attribute values are missing. Also the background knowledge need not necessarily be perfect.

Learning algorithms can be generally classified into one of two major cate-

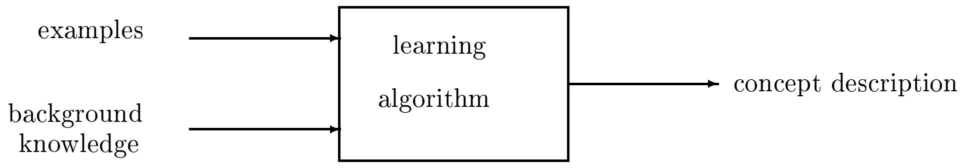


Figure 1.1 Machine Learning Task

gories: "black-box" methods, such as neural networks or mathematical statistics, and knowledge-oriented methods The *black-box* approaches develop their own concept representation that is to be used for concept recognition purposes. However, this internal description cannot be easily interpreted by the user and provides neither insight nor explanation of the recognition process. Black-box methods typically involve numeric calculations of coefficients, distances, or weights.

On the other hand, knowledge-oriented methods aim at creating symbolic knowledge structures that satisfy the principle of comprehensiblity (Michalski, 1983). Michie (1988) formulated the distinction between black-box and knowledge-oriented concept learning systems in the terms of three criteria. These criteria—weak, strong and ultrastrong—differ in their aspirations regarding the comprehensibility of learned descriptions:

1. *Weak criterion*: The system uses sample data to generate an updated basis for improved performance on subsequent data.
2. *Strong criterion*: Weak criterion is satisfied. Moreover, the system can communicate its internal updates in explicit symbolic form.
3. *Ultrastrong criterion*: Weak and strong criteria are satisfied. Moreover, the system can communicate its internal updates in an *operationally effective* symbolic form.

Any approach to learning, including artificial neural networks and statistical methods, satisfies the weak criterion. Methods of machine learning that have been inspired by artificial intelligence research have been particularly concerned with the strong criterion. The last, ultrastrong, criterion requires that the user not only understands the induced description, but can also *use* this description without being aided by a computer. That is, the user can execute all the corresponding computations required to apply the induced descriptions in his or her mind.

This chapter, and this book in general, is mainly concerned with the *knowledge-oriented* algorithms capable of developing descriptions understandable to the user. Most of these methods are based on manipulating symbolic structures. Before proceeding further, let us explore the cognitive perspective and the representation issues related to the crucial notion of this approach to machine learning: concept.

1.2.1 Cognitive Perspective

The notion of a *concept* is as vital to machine learning as is chemical compound to chemistry, force field to physics, number to mathematics, and knowledge to artificial intelligence. Throughout this book, we will understand a concept as an abstraction standing for a set of objects sharing some properties that differentiate them from other concepts.

'Bird,' 'tiger,' 'vertebrate,' 'aminoacid,' 'car,' 'rainy day,' 'mathematics,' 'prime number,' 'leukemia,' 'galaxy,' 'despotic ruler,' or 'fertile land' are concepts. Note that their boundaries are not always clear. While no serious problems are posed by 'bird' or 'prime number,' any attempt to define precisely what is meant by a 'despotic ruler' will turn out to be rather tricky because this concept is subjective and context-dependent. Other concepts, such as 'mathematics' or 'galaxy,' have fuzzy boundaries. Even when concepts can be defined precisely (e.g., leukemia), a correct classification of an object (e.g., a patient) based on the available data may present a difficult problem.

In statistics, the notion of a *cluster* is often used. Its meaning is related but different. By a cluster, statisticians usually mean a group of objects that are relatively close to each other according to a chosen numerical distance (which does not need to be Euclidean). A group of students sitting close to each other in the lecture hall represent a cluster. On the other hand, 'students of machine learning' is a concept defined by characteristic features such as the shared interest in the indicated discipline.

In real world, concepts are never isolated. Groups of related concepts can often be organized into a generalization hierarchy represented by a tree or graph (Figure 1.2). At a given level of the hierarchy, concepts are usually disjoint, but sometimes they can overlap; the difference between them may be small or large. In a generalization hierarchy, a concept can be exemplified not only by the objects at the bottom level but also by subconcepts at any level below the concept in question. For example, the concept 'bird' is an example of a 'vertebrate' and so is 'eagle.'

Three important notions that are germane to the mutual relations among concept deserve brief exposition: basic-level effect, typicality, and contextual dependency.

Psychological findings indicate that in an ordered hierarchy of concepts (e.g., a branch in Figure 1.2), one level can be understood as *basic*. This means that the concept on this level shares with its subconcepts a large number of features that can be described in sensorially recognizable terms.

For illustration, consider the sequence

$$\text{eagle} \rightarrow \text{bird} \rightarrow \text{animal} \rightarrow \text{living being}$$

Here, the basic-level concept is 'bird' because its subconcepts ('eagle,' 'blackbird,' 'ostrich,' etc.) share features that can be detected by sensors (e.g., wings, feathers, beak). Note that the subconcepts of 'animal'—e.g. reptiles, birds, or mammals—do not share such features and, therefore, the level of 'animal' is not basic. The same goes for 'living being.'

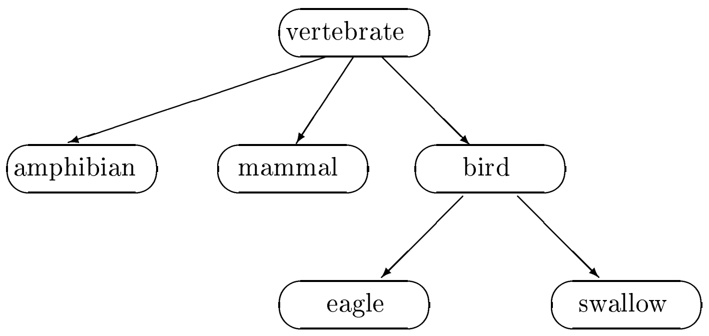


Figure 1.2 Generalization hierarchy

The basic-level concepts can be found in many concept hierarchies. After some thought, the reader will agree that in

BMW → car → transportation means

such a concept is 'car.'

The fact that basic-level concepts are described by features that can be readily identified makes them easy to learn for humans. The concepts on lower levels can then be understood as specializations of the basic-level concepts (e.g., 'birds that can sing') whereas concepts on higher levels are often defined as groups of basic-level concepts sharing some important feature.

The second useful aspect says how *typical* an instance is for a given concept. In learning, the typicality of instances plays a crucial role—to exemplify 'bird' by pinguin, ostrich, and goose will hardly lead to good learner's understanding of the concept. In psychological literature, two ways to measure typicality have been suggested: by the number of features shared with other subconcepts and by the number of features inherited from superconcepts (the greater the number of inherited features that can be found in the instance, the more typical the instance is).

The third aspect to be considered is *context dependency.* When speaking about students, the speaker can have different concepts in mind: students from the given university, students of computer science, or students from the high school in the neighborhood. Each of them has different connotations, as far as their knowledge, age, and interests are concerned. Obviously, real-world concepts are learnable only in an appropriate context.

Readers interested in more details related to psychological and cognitive aspects of concept acquisition, remembering, and recalling will find a profound analysis in Klimesch (1988).

1.2.2 Representational Issues

The first question to be posed whenever a task is to be solved by a computer is how to translate the problem into computational terms. In machine learning, this means how to represent concepts, examples, and the background knowledge. To describe concepts and examples, *representation languages* are used. In the sequel, some languages encountered in machine learning are outlined. In the ascending order of complexity and expressive power, they include zero-order logic, attribute-value logic, Horn clauses, and second-order logic. To avoid unnecessary mathematical complexity, we just give intuitional explanations of these languages.

From now on, we will say that a description *covers* an example if it is true for (or satisfied by) the example. Thus the description `has_four_legs` covers a lion, but does not cover a goose.

Zero Order Logic: Propositional Calculus

In zero-order logic, also called propositional calculus, examples and concepts are described by conjunctions of Boolean *constants* that stand for the individual features (attribute values). In mathematical terms, this type of description can look something like:

$$\texttt{c} \Leftarrow \texttt{x} \wedge \texttt{y} \wedge \texttt{z}$$

which reads: an object is an instance of the concept `c` whenever the conditions `x, y`, and `z` hold simultaneously.

For illustration, consider the following naïve description of a potential husband for Jane:

$$\texttt{can\_marry\_jane} \Leftarrow \texttt{male} \wedge \texttt{grown\_up} \wedge \texttt{single}$$

Other connectives include negation and disjunction.

The zero-order logic is capable of describing only simple concepts and the reader will find it difficult to capture in this way complex concepts encountered in daily life. In other words, the zero-order logic has low *descriptive power.* The low descriptive power excludes widespread application of zero-order logic in machine learning and can only be used to illustrate simple algorithms.

Attributional Logic

Formally, attributional logic is roughly equivalent to zero-order logic but employes a much richer and flexible notation. The basic idea is to characterize examples and concepts by values of some predefined set of *attributes*, such as color or height. The

Table 1.1 Positive and negative examples of the concept *big* $\vee$ *(medium* $\wedge$ *expensive)*

Object	*Make*	*Size*	*Price*	Classification
*car*1	European	big	affordable	$\oplus$
*car*2	Japanese	big	affordable	$\oplus$
*car*3	European	medium	affordable	$\ominus$
*car*4	European	small	affordable	$\ominus$
*car*5	European	medium	expensive	$\oplus$
*car*6	Japanese	medium	affordable	$\ominus$
*car*7	Japanese	medium	expensive	$\oplus$
*car*8	European	big	expensive	$\oplus$

improvement over the zero-order logic (where concepts are characterized by conjunctions of constants) is that the attributes are *variables* that can take on various values. For instance, the value of the attribute 'color' can be 'green,' 'red,' 'blue,' 'blue or green,' or '*' which stands for *any* color (the 'or' linking two or more attribute values is called *internal disjunction.*

Examples are often presented in a table where each row represents an example and each column stands for an attribute. Thus Table 1.1 contains positive ($\oplus$) and negative ($\ominus$) examples of a car attractive for a young enterpreneur.

Boolean, numeric, symbolic, or mixed-valued attributes can be considered, and the scope of their values is often constrained by background knowledge. The legal values can often be ordered or partially ordered. Intuitively, ordered values are those that can be expressed by integers, for instance, the length and height as measured in some properly chosen units. Partially ordered values are those that form a hierarchy. For illustration, consider the possible values of the variable 'animal' in Figure 1.2. Note that 'eagle' is more specific than 'bird' but is unrelated to 'amphibian.'

As a description language, attributional logic is significantly more practical than zero-order logic, although in a strict mathematical sense they have equivalent expressiveness. For this reason, attribute-value logic has received considerable attention from machine-learning researchers and provides the basis for such well-known algorithms as TDIDT (Quinlan, 1986) or AQ (Michalski, 1983a). A formal basis for such a description language was defined in *variable-valued logic* (Michalski, 1973a).

First Order Predicate Logic: Horn Clauses

First order logic provides a formal framework for describing and reasoning about objects, their parts, and relations among the objects and/or the parts. An importang subset of first order logic are Horn clauses. A Horn clause consists of a head and a body, as illustrated by the following definition of a grandparent:

```
grandparent(X,Y) :- parent(X,Z), parent(Z,Y)
```

which says that the person X is a grandparent of the person Y if a person Z can be found such that X is a parent of Z, and Z is a parent of Y. The part to the left from ':-' is called the *head* and the part to the right from ':-' is called the *body* of the clause. The commas stand for conjunctions and X, Y, and Z are universally quantified variables.

The words 'grandparent' and 'parent' are called *predicates* and the variables in the parentheses are called *arguments*. The number of arguments can be, in general, arbitrary but is fixed for a given predicate. If all predicates have precisely one argument, the language reduces into the attribute-value logic. If all predicates have precisely zero arguments, the language reduces into zero-order logic.

Horn clauses constitute an advanced representation language that facilitates very complex descriptions. They form the basis of the programming language Prolog and are used, for instance, in the learning system FOIL (Quinlan, 1990).

Second Order Logic

Second order logic builds on the idea that the predicate names themselves can be considered as variables. Thus, for instance, the schema

$$p(X,Y) \text{ :- } q(X,XW) \wedge q(Y,YW) \wedge r(XW,YW)$$

can be instantiated to

```
brothers(X,Y) :- son(X,XW) ∧ son(Y,YW) ∧ equal(XW,YW)
```

by means of the substitution

$\Theta = \{p = \text{brothers},\ q = \text{son},\ r = \text{equal}\}$

Another possible instantiation is

```
lighter(X,Y) :- weight(X,XW) ∧ weight(Y,YW) ∧ less(XW,YW)
```

with the substitution

$\Theta = \{p = \text{lighter},\ q = \text{weight},\ r = \text{less}\}$

So the skeleton of the clause remains untouched and only the predicate names can vary. The rationale behind this idea is that groups of concepts often share the same structure of admissible descriptions. The *second-order schemata* are used to store the most successful of such structures to assist the search for the concept. However, this representation language is rather complex and is rarely used. For an exception, see the program CIA, described in de Raedt (1992).

Explicitly Constrained Languages

Representation languages based on logic are sometimes so rich and flexible that their use for machine learning is computationally intractable. Therefore, a common practice is to introduce various constraints, such as limited number of predicates in the clause, limited number of predicate arguments, or excluded recursive definitions.

Limited occurence of variables in a clause means that the number of variables in the body of the clause is not allowed to exceed a predefined threshold. For instance, only those variables can appear in the body that have already appeared in the head of the clause. Alternatively, precisely one variable not appearing in the head can be allowed to appear in the body. A number of similar constraints of this kind can easily be introduced.

Another restriction can exclude *functions* from predicate arguments. This can become a severe limitation because, in general, an argument need not necessarily be a simple variable but also a calculation, complex algebraic or logic expression, or a n-ary function. The presence of functions significantly increases the space of all possible descriptions.

Finally, an important restriction can exclude *recursive descriptions*. Expressed in first-order logic, the power of recursive descriptions is often demonstrated by the definition of the predicate `ancestor`:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

which reads: X is an ancestor of Y either if X is parent of Y or if a person Z can be found such that X is parent of Z and Z is ancestor of Y.

Although recursively described concepts are sometimes unavoidable, they tend to complicate the learner's task and to be difficult to comprehend. Hence, recursions are sometimes explicitly forbidden by the language definition (Michalski, 1980).

Alternative Representations

Theoretically, also some extra-logical representational schemes are possible candidates for concept characterization. Among them, Minsky's *frames* (Minsky, 1975) have been very popular in the artificial intelligence community. Abstract mathematical structures such as *grammars* or *finite automata*, too, can be recommended because they possess structural properties that have been deeply explored by mathematicians and are useful for some applications.

However, these representations received relatively limited attention in machine learning. The reader should only bear in mind that even though logical schemes currently prevail in the relevant literature, other options can be considered in future research.

1.3 Search through the Space of Generalizations

Suppose that a representation language has been selected and that the learner wants to learn a concept from set of positive and negative examples. Even if the descriptions are based on attribute-value logic, the space of all discernible concepts is surprisingly large. Ten attributes with five possible values for each of them amount to $5^{10} = 9,765,625$ possible vectors. Any subset of such vectors can correspond to a concept, which means that $2^{9,765,625}$ concepts can be defined over these attributes. In more complex languages, this number grows even faster, even though background knowledge can limit the size of the representation space.

To cope with the problem of computational tractability, the learner mostly combines two powerful techniques: induction and heuristic search. The discussion of the relevant techniques, together with the analysis of the fundamental reasoning principles employed by the learner, is the task of the subsequent subsections.

1.3.1 Inductive Essence of Learning

Imagine that an extraterrestrial scientist lands in the vicinity of your town with the intention to study terrestrial living forms. The scientist has some preliminary linguistic knowledge which, however, needs polishing. This is why he contacts you as one of the informed aborigines with the question what is 'bird.'

To start with, you point to a 'blackbird' as a positive example of the concept. Nevertheless, simple memorizing of all features of blackbirds will hardly be sufficient to recognize other birds as instances of the same category. Obviously, a generalization of this example is necessary. But how strong generalization? To establish its limits, the ET will need a negative example, something which is *not* a bird. Knowing that, you suggest a 'dog.' Apparently, one of the differences between dogs and blackbirds is that dogs do not possess wings. (Suppose that at the beginning, the ET is interested in easily recognizable discriminators.)

To check whether all creatures with wings are birds, the ET will ask whether 'flies' also belong to the same category. Obviously they do not, which is a good indication that the possession of wings is a too general feature for proper discrimination between positive and negative instances. A specialization of this description is necessary. This can be accomplished by way of adding one more feature from an example to the current description. A noticable feature of the blackbird—absent in dogs and flies—is the yellow beak. ET might then assume that a 'crow' is not a bird because it does not have the yellow beak.

The fact that this feature is found only in some of the birds signals that this time the description has become too specific and, again, a proper generalization should be considered. Thus the scientist drops the requirement of the yellow color of the beak and concludes simply that birds are winged creatures with beaks. This description facilitates the recognition of a 'sparrow' as a positive example of a 'bird.'

This simple story illustrates an elementary machine-learning strategy. Let us recapitulate the procedure with a more scientific vocabulary. After the first example (blackbird), the idea of birds is very specialized. ET only knows the *most specific* description of this single example and nothing more. This description can become the first member of the set of the most specific descriptions, denoted by S. Any generalization of the most specific description is possible at this point. Only after the arrival of first negative example (dog), the learner is able to impose a limit on the generalization, thus obtaining a set of the *most general* descriptions (denoted by G) that correctly cover the positive but not the negative example. From the set G, the scientist selects 'has-wings' as the most appealing.

The next negative example (fly) reveals that the description is *overgeneralized*: wings do not represent a sufficient discriminator between blackbird (positive) and fly (negative). Therefore, the set G has to be specialized. On the other hand, the next positive example (crow) enriches the set S with another most specific description, which calls for another generalization, accomplished by replacing the description 'has-wings' with 'has-wings' and 'has-yellow-beak.'

To summarize, generalization is applied to the set S whenever a new positive example arrives. Conversely, a negative example can necessitate the specialization of the set G. This principle underlies a family of techniques—called Version Space algorithms—that build on the idea of gradual reduction of the space of current *versions* of the concept description. The method was invented and published by Mitchell (1982); for a more recent treatise with extensive bibliography see Hirsh (1990). An earlier method that viewed concept learning as a series of generalizations and specializations of a single hypothesis (rather than two sets of hypotheses) was presented in a well-known paper by Winston (1970).

For our needs, the details of the algorithm and of its many derivatives are not so important. The above story was meant to demonstrate the fact that concept learning can be conceived as a search through the space of descriptions, the essencial search operators being *generalization* and *specialization*.

A good deal of this chapter will deal, directly or indirectly, with this kind of operators. For instance, a description in Horn clauses can be generalized by turning a constant into variable or by dropping a condition. Hence the clause:

```
p(X,Y) :- q(X, 2), r(Y, 2).
```

can be generalized into

```
p(X,Y) :- q(X,Z), r(Y,Z).
```

or into

```
p(X,Y) :- q(X,2).
```

Specialization can be understood as the complementary operation. In this sense, a Horn clause can be specialized by turning a variable into constant, or by adding a literal to the clause.

Proper selection of the search operators is (besides the choice of the representation language), the critical task of the designer of a learning program. One of the first attempts to systemize generalization operators was made by Michalski (1983).

1.3.2 Exhaustive Search

A widespread framework for concept learning is *search* through the space of descriptions permitted by the learner's representation language. The merit of this philosophy is obvious: search techniques have been deeply investigated by artificial ntelligence researchers and are widely understood. The following terms are requisite in the definition of a search technique:

In general, a search process explores *states* in a search space according to the folowing:

- *Initial state* is the starting position of the search. In machine learning, initial states often correspond to the the most specific concept descriptions, i.e., to the positive examples themselves;
- *Termination criterion* is the objective to be arrived at. States that satisfy the termination criterion are referred to as *final states.* In machine learning, the termination criterion can require that the description cover all positive and no negative examples;
- *Search operators* advance the search from one state to another. In machine learning, these operators are mostly generalizations and/or specializations of concept descriptions;
- *Search strategy* determines under what conditions and to which state an operator is to be applied.

The two fundamental strategies of systematic search are the depth-first search and breadth-first search. These can be easily explained if we visualize the space of all possible states as an oriented graph whose nodes represent the individual states and edges represent the search operators.

In the *depth-first search* an operator is applied to the initial state S_1, arriving at a new state S_2. If S_2 is not recognized as the final state, then, again, an operator is applied to S_2, thus arriving at a new state S_3. If no new state can be reached in this way and the final state has not yet been found, the system *backtracks* (returns) to the previous state and applies some other operator. If this is not possible, the system backtracks further, until a state is found that allows the application of some of the operators. If no such state can be found, the search terminates. The principle is depicted in Figure 1.3. The numbers in the rectangles indicate the order in which the states are visited.

Breadth-first search constitutes the complementary approach. First, all operators are applied, one by one, to the initial state. The resulting states are then tested. If some of them is recognized as the final state, the algorithm stops. Otherwise, the operators

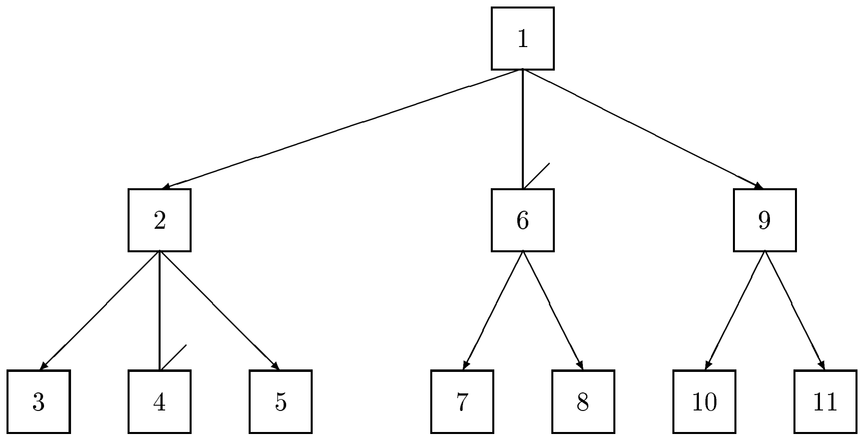


Figure 1.3 Depth-first search

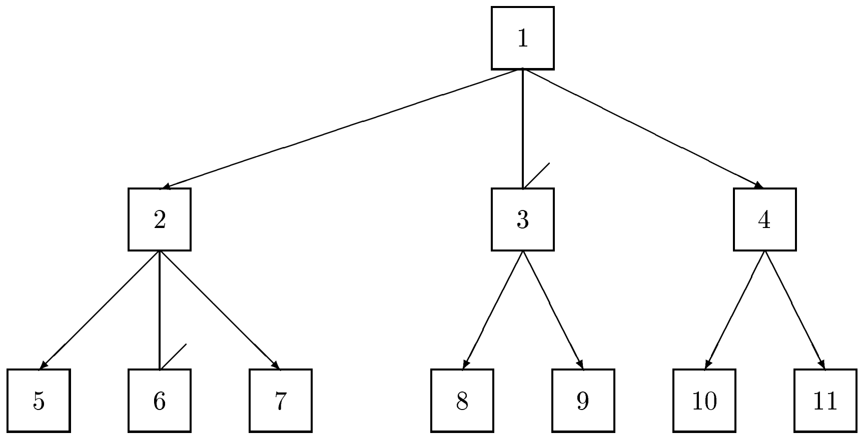


Figure 1.4 Breadth-first search

are applied to all subsequent states, then again to the subsequent states, and so on, until the termination criterion is satisfied. The principle is depicted in Figure 1.4.

Note that unlike the depth-first search, the breadth-first search assumes no backtracking, which is a slight simplification of the task. On the other hand, the searcher must store many intermediate states, which can render the system much too expensive.

1.3.3 Heuristic Search

The fundamental search techniques are not very efficient in large search spaces where *heuristics* guiding the search must be considered. The task of the heuristics is to decide which of the available operators will lead the search to the closest proximity of the final state. This requires an evaluation function to assess the value of each of the reached states. Assume, for the moment, that the evaluation function is given.

Best-First Search Algorithm.

1. Let the initial state be referred to as the *best* state and let the set of *current states* consist of this single state;
2. If the best state satisfies the given termination criterion, then stop (the best state is the solution of the search);
3. Apply all applicable operators to the best state, thus creating a set of new states that are added to the set of current states;
4. Evaluate all current states. Decide which is the best state and go to 2.

This algorithm differs from the breadth-first algorithm in that it always extends only the most promising state, thus hopefully speeding up the search. The price is the danger of falling to a local maximum of the evaluation function.

The algorithm is illustrated by the following example.

Example.

The learner tries to derive the concept description from the set of eight positive and negative examples described by the attribute values as shown in Table 1.2. '$\oplus$' stands for 'positive example' and '$\ominus$' stands for 'negative example.' Assume two operators: 'specialize the current description by adding a conjunction' and 'generalize the current description by adding a disjunction.'

Let the initial state be 'any description.' The application of the specialization operator (generalization has no sense, here) will produce the following descriptions: $at1 = a, at1 = b, at2 = x, at2 = y, at2 = z, at3 = m$, and $at3 = n$. Among them, $at2 = x$ and $at3 = m$ do not cover any $\ominus$ and will probably achieve the highest value of a reasonable evaluation function. Suppose that, for some reason, $at2 = x$ is preferred and will thus become the *best* description.

Since some $\oplus$'s in the table are now not covered, the learner will try to apply the search operators to the best description. Applying the specialization operator only

Table 1.2 Positive and negative examples for concept learning

example	$at1$	$at2$	$at3$	classif.
$e1$	a	x	n	$\oplus$
$e2$	b	x	n	$\oplus$
$e3$	a	y	n	$\ominus$
$e4$	a	z	n	$\ominus$
$e5$	a	y	m	$\oplus$
$e6$	b	y	n	$\ominus$
$e7$	b	y	m	$\oplus$
$e8$	a	x	m	$\oplus$

worsens the situation by reducing the number of covered $\oplus$'s. However, by generalizing the description into $at2 = x \vee at2 = y$ the number of covered $\oplus$'s increased. Suppose that the evaluation function confirms this description as better than $at2 = x$.

The new description covers all $\oplus$'s but, on the other hand, it covers also two $\ominus$'s. Thus, in the next step, the description is specialized into $at2 = x \vee [(at2 = y) \wedge X]$, where X stands for any of the following conjuncts: $at1 = a, at1 = b, at3 = m$, and $at3 = n$.

Among the new states, the best one is $at2 = x \vee [(at2 = y) \wedge (at3 = m)]$. As it covers all $\oplus$'s and no $\ominus$'s, the search terminates. $\square$

The best-first search may require excessive memory because it stores all the generated states. A more economical approach is the *beam search* that only retains N best states at any time.

Beam-Search Algorithm

1. Let the initial state be the *best* state;
2. If the best state satisfies some termination criterion, stop;
3. If the number of current states is larger than N, keep only the N best states and delete all others;
4. Apply the search operators to the best state, and add the newly created states to the set of current states;
5. Evaluate all states and go to 2.

A popular instantiation of the beam-search algorithm is defined by $N = 1$ and is sometimes called *hill-climbing* algorithm. The name is meant to emphasize the resemblance to hill climbers striving to find the shortest trajectory to the peak and picking always the steepest path.

Readers interested in more detailed information about search techniques are referred to artificial-intelligence literature, for instance Charniak and McDermott (1985) or Bratko (1990).

1.4 Classic Methods of Learning

Having explained the principles of generalization and specialization operators as well as some basic search techniques, we can proceed two essential learning principles, namely the divide-and-conquer learning and the AQ-philosophy. Both of them are crucial for the understanding of more advanced methods.

1.4.1 Divide-and-Conquer Learning

The essence of this method is very simple: the entire set of examples is split into subsets that are more easy to handle. In attributional logic, the partitioning is carried out along attribute values so that all examples in a subset share the same value of the given attribute.

In Table 1.2, the attribute $at1$ splits the set of eight examples into the subset defined by $at1 = a$ and the subset defined by $at1 = b$. Similarly, $at3$ imposes an alternative split into two subsets, one defined by $at3 = m$ and the other defined by $at3 = n$. Finally, $at2$ imposes decomposition into three subsets defined by $at2 = x$, $at2 = y$, and $at2 = z$, respectively.

This principle underlies the popular algorithm for induction of decision trees (see Breiman et al., 1984, and Quinlan, 1986), known under the acronym TDIDT (Top-Down Induction of Decision Trees) or ID3. With a properly defined evaluation function, the TDIDT-algorithm described below will derive the decision tree in Figure 1.5 from the examples in Table 1.2. The reader is encouraged to check that the concept description in the tree is really consistent with the table. For instance, the example $e1$ has $at2 = x$, which sends it downward along the leftmost branch, only to end up in the box labeled with $\oplus$. The example $e3$ has $at2 = y$ and will be passed down the middle branch, arriving at the test on attribute $at3$; having the value $at3 = n$, the example follows the right-hand branch, ending up at the box labeled with $\ominus$.

Note that the tree can be rewritten into the following logical expressions:

$$\begin{aligned}(\mathit{class} = \oplus) &\leftarrow (at2 = x) \vee [(at2 = y) \wedge (at3 = m)]\\ (\mathit{class} = \ominus) &\leftarrow (at2 = z) \vee [(at2 = y) \wedge (at3 = n)]\end{aligned}$$

Any future example will be classified according to these two formulae or the decision tree. The classification of examples that do not satisfy either of these rules (for instance, if the example has $at2 = w$ which is a value unseen during training) can be based on the distance between the example description and the rules. Alternatively, an 'I-don't-know' answer can be issued.

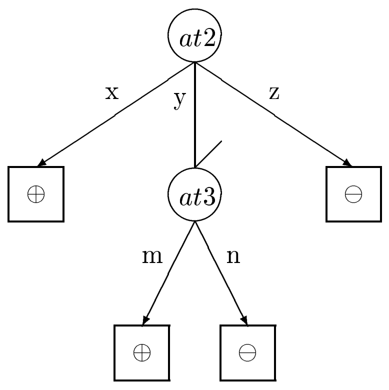


Figure 1.5 A decision tree

TDIDT Algorithm

$S \ldots$ the set of examples

1. Find the 'best' attribute *at*;
2. Split the set S into the subsets $S_1, S_2, \ldots$, so that all examples in the subset S_i have $at = v_i$. Each subset constitutes a node in the decision tree;
3. For each S_i: if all examples in S_i belong to the same class ($\oplus$ or $\ominus$), then create a leaf of the decision tree and label it with this class label. Otherwise, perform the same procedure (go to 1) with $S = S_i$.

The algorithm terminates when all subsets are labeled or when no further attributes splitting the unlabeled sets are available (in this case, some leaves of the tree will cover examples of both classes).

How to Find the 'Best' Attribute?

Assume two classes ($\oplus$ and $\ominus$) of examples described by attribute values. The task is to find the best attribute for step 1 in the previous algorithm. A plausible criterion is based on the number of $\oplus$'s and $\ominus$'s in each of the subsets generated by the different attribute values.

After some thought, the reader will agree that we need a function satisfying the following requirements:

1. The function reaches its maximum when all subsets are homogeneous, i.e., all examples in S_i are $\oplus$ or all examples in S_i are $\ominus$. In this case, the information about the attribute value is sufficient to decide whether the example is positive or negative;
2. The function reaches its miminum when 50% of the examples in each of the subsets are positive and 50% are negative;

3. The function should be steep when close to the extremes (100% positives and 0% negatives or vice versa) and flat when in the 50%—50% region.

Mathematicians know that information is maximized when another important quantity, *entropy*, is minimized. Entropy determines the extent of randomness, 'unstructuredness' and chaos in the data. In our context, the entropy of the subset S_i can be calculated by means of the following formula:

$$H(S_i) = -p_i^+ \log p_i^+ - p_i^- \log p_i^-$$

where p_i^+ is the probability that a randomly taken example in S_i is $\oplus$ and can be estimated by the relative frequency $p_i^+ = \frac{n_i^+}{n_i^+ + n_i^-}$; similarly, p_i^- is the probability that a randomly taken example in S_i is $\ominus$ and can be estimated by $p_i^- = \frac{n_i^-}{n_i^+ + n_i^-}$. Here, n_i^+ is the number of $\oplus$'s in S_i and n_i^- is the number of $\ominus$'s in S_i.

Let the values of attribute *at* split the set S of examples into the subsets $S_i, i = 1, \ldots K$. Then the entropy of the system of subsets S_i is:

$$H(S, at) = \Sigma_{i=1}^K P(S_i) \cdot H(S_i)$$

where $H(S_i)$ is the entropy of the subset S_i; $P(S_i)$ is the probability of an example belonging to S_i and can be estimated by the relative size of the subset S_i in S: $P(S_i) = \frac{|S_i|}{|S|}$.

The information gain achieved by the partitioning along *at* is measured by the entailed decrease in entropy:

$$I(S, at) = H(S) - H(S, at)$$

where $H(S)$ is the apriori entropy of S (before the splitting) and $H(S, at)$ is the entropy of the system of subsets generated by the values of *at*.

Let us demostrate the use of these formulae by building a decision tree from the examples in Table 1.2.

Since there are 5 positives and 3 negatives among the 8 examples in S, the apriori entropy of the system S is:

$$\begin{aligned} H(S) &= -p^+ \log p^+ - p^- \log p^- \\ &= -(5/8)\log(5/8) - (3/8)\log(3/8) \\ &= 0.954 bits \end{aligned}$$

Note that this entropy is close to its maximum ($0.954 \doteq 1$) because the number of $\oplus$'s is about the same as the number of $\ominus$'s. If the number of $\oplus$'s were much larger than that of $\ominus$'s (or vice versa), than we would have a high chance of a correct guess of the class by simply assuming that it is always $\oplus$. This would correspond to small entropy.

What will be the entropy of the different partitions as generated by the individual attributes? For instance, attribute $at2$ can acquire three different values, x, y, and z. For each of them, we obtain the entropies of the related subsets S_x, S_y, and S_z:

$$\begin{array}{rrcl} at2: & H(S_y) & = & -(2/4)\log(2/4) - (2/4)\log(2/4) = 1\,bit \\ & H(S_x) & = & -1 \cdot \log 1 - 0 \cdot \log 0 = 0\,bit \\ & H(S_z) & = & -0 \cdot \log 0 - 1 \cdot \log 1 = 0\,bit \end{array}$$

and determine the overall entropy as their weighted sum:

$$H(S, at2) = (3/8) \cdot 0 + (1/8) \cdot 0 + (4/8) \cdot 1 = 0.5\,bit$$

Calculating similarly the entropies for the remaining attributes, we will arrive at the following information gains:

$$\begin{array}{rcl} I(S, at2) & = & H(S) - H(S, at2) = 0.954 - 0.500 = 0.454\,bits \\ I(S, at1) & = & H(S) - H(S, at1) = 0.954 - 0.951 = 0.003\,bits \\ I(S, at3) & = & H(S) - H(S, at3) = 0.954 - 0.607 = 0.347\,bits \end{array}$$

Evidently, $at2$ yields the highest information gain (0.454 bits) and thus ought to be selected as the root of the tree. This is how the tree from Figure 1.5 was created.

The use of entropy is just one of many possibilities. Several alternative attribute-selection criteria have been suggested. Some can be found in Breimann et al. (1984) and in Mingers (1989a).

Probability Estimation

Estimating the probabilities p_i^+ and p_i^- by relative frequencies is far from being ideal because the estimates are reliable only by sufficiently large sets of examples. However, in the process of a decision-tree generation, the number of examples quickly decreases with each subsequent splitting. Assume that only two examples are left and that both of them have the class of $\oplus$. Then, a naïve learner will conclude that the probability of the class $\oplus$ is 100%, which may not be true.

For this reason, improved methods to estimate probabilities have been put forward. For instance, the m-estimate (Cestnik 1990) calculates the probabilities by the following formula:

$$p_\oplus = \frac{n_\oplus + m p_a}{N + m}$$

where $n_\oplus$ is the number of $\oplus$'s, N is the total number of examples in the subset ($N = n_\oplus + n_\ominus$), p_a is the *prior* probability of $\oplus$, and m is the parameter of the estimate. In the case of much noise in the examples m should be set high, and for low noise m is set low. In any case, the user or the expert who understands the domain and the level of noise should recommend a corresponding setting for m and p_a.

A special case of the m-estimate is the Laplace law of succession (or simply Laplace estimate) which is for two classes, $\oplus$ and $\ominus$ given by:

Table 1.3 Probability (in percentage) of heads in tossing a coin

toss No.	1	2	3	4	5
outcome	heads	heads	tails	heads	tails
relative frequency	100	100	67	75	60
Laplace estimate	67	75	60	67	57

$$p_{\oplus} = \frac{n_{\oplus} + 1}{N + 2}$$

The appropriateness of this formula is illustrated by the simple experiment in Table 1.3. Suppose you are tossing a coin, each time arriving at one of the two possible outcomes, heads and tails. The table gives the probabilities of heads (in percent) obtained from relative frequencies and from the Laplace estimate. Clearly, the Laplace estimate gives more realistic results.

Pruning the Trees

A few pitfalls can put the utility of a decision tree in question. One of them is *overfitting*. A tree branch (ended with a class label) might have been created from examples that are noisy in the sense that the attribute values or class labels are erroneous. Obviously, this branch, or rather some of its decision tests, will be misleading. Second, if the number of attributes is large, the tree may contain tests on random features that are actually irrelevant for correct classifications. Thus for instance, the color of a car does not matter if we are interested in the principle of the engine. In spite of that, the attribute can appear in the trees whenever many cars with, say, combustion-based engine happen to be red. Finally, very large trees are hard to interpret and the user will perceive them as a black box representation.

For all these reasons, it may be beneficial to prune the resulting tree by the method indicated in Figure 1.6.

In principle, two approaches to pruning are possible: on-line pruning and post-pruning. The essence of *on-line pruning* is to stop the tree growing when the information gain caused by the partitioning of the example set falls below a certain threshold. *Post-pruning* methods prune out some of the branches after the tree has been completed.

A popular approach to pruning, known as *minimal-error* pruning, was designed by Niblett and Bratko (1986). This technique aims at pruning the tree to such an extent that the overall expected classification error on new examples is minimised. For this purpose, the classification error is estimated for each node in the tree. In the leaves, the error is estimated using one of the methods for estimating the probability that a new object falling into this leaf will be misclassified. Suppose that N is the number of examples that end up in the leaf, and e is the number of these examples that

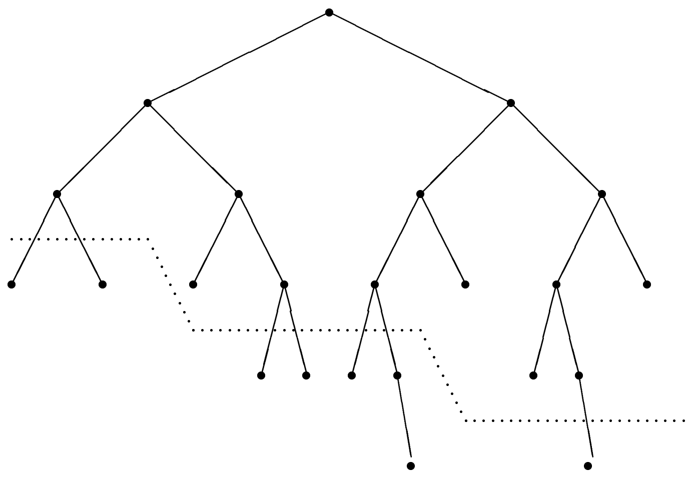


Figure 1.6 Pruning decision trees

are misclassified at this leaf. Niblett and Bratko (1986) used the Laplace estimate $(e+1)/(N+k)$ (where k is the number of all the classes) to estimate the expected error. Cestnik and Bratko (1991) showed that using the m-estimate instead of Laplace estimate gives better results. For a non-leaf node of the decision tree, its classification error is estimated as the weighted sum of the classification errors of the node's subtrees. The weights are calculated as relative frequencies of the examples passing from the node into the corresponding subtrees. This non-leaf error estimate is called backed-up error. Now the classification error in a non-leaf node is also estimated for the case that its subtrees were pruned out and the non-leaf would thus become a leaf. If this error estimate is lower than the backed-up error, the subtrees will be pruned out. This process of pruning subtrees starts at the bottom levels of the tree and propagates upwards as long as the backed-up errors are higher than the "static estimates".

Several alternative approaches to pruning are reviewed by Mingers (1989b) and Esposito et al. (1993).

Other Motivations for Tree Simplification

Figure 1.7 illustrates another type of tree simplification, this time with the objective to carry out a kind of *constructive induction* (Michalski, 1983a). The learning system strives to create new attributes as logical expressions over the attributes provided by the teacher. Constructive induction can be profitable in situations where a subtree is replicated in more than one position in the tree—see Figures 1.8 and 1.9.

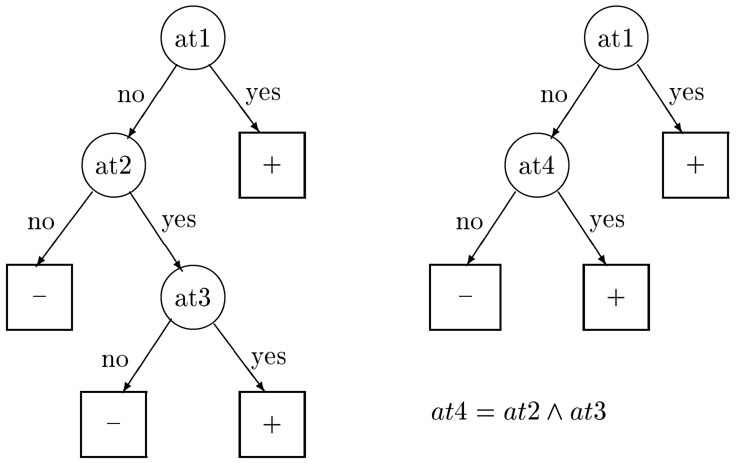


Figure 1.7 Constructive induction in decision trees. A new attribute, *at4*, is constructed.

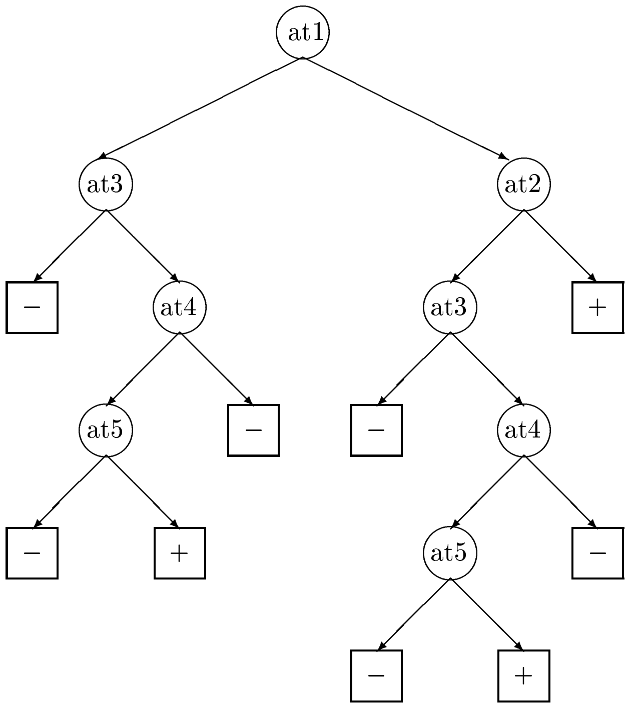


Figure 1.8 The replication problem in decision trees

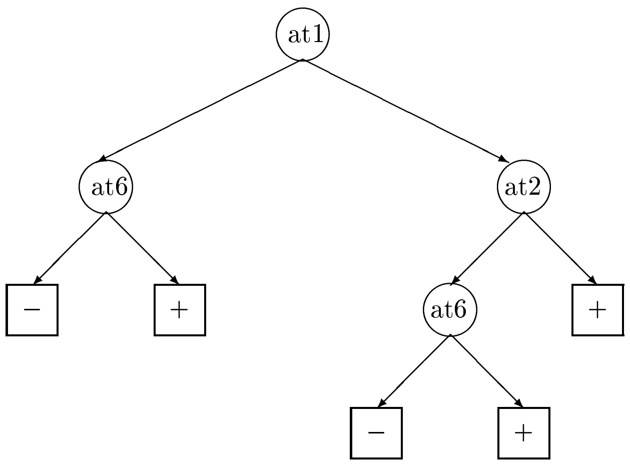


Figure 1.9 Simplified version of the tree from the previous figure

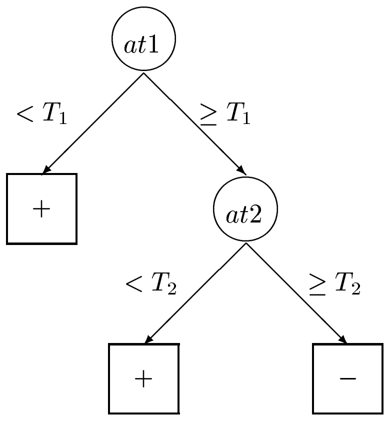


Figure 1.10 Decision tree induced from numeric attributes

Coping with Numeric Data

So far, the analysis has been restricted to symbolic attributes. However, decision trees can be induced also from numerical attributes. One possibility is to provide one additional step, the *binarization* of the numeric attributes, that means thresholding their numerical ranges into pairs of subintervals to be treated as symbols. Figure 1.10 shows a decision tree built from numeric data. At each node, the respective attribute value is tested against threshold T_i.

The threshold position in the range of values can, again, be determined by entropy. Suppose that attribute $at1$ is to be discretized. First, we order all the examples accord-

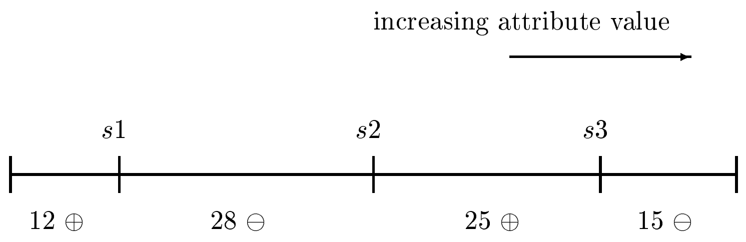


Figure 1.11 Examples sorted by the values of an attribute; $s1$, $s2$, and $s3$ are candidate split points

ing to the value of $at1$ and observe the classification values. In the case illustrated by Figure 1.11, the classification values $\oplus$ and $\ominus$ decompose the set of 80 examples into 4 regions (in realistic settings, the number of such regions is likely to be larger). The candidate splitting cuts lie on the boundaries between the regions. Then, the information gain of each of these cuts is calculated as explained earlier, and the cut providing the highest information gain is selected (for a detailed and rigorous discussion of this mechanism see Fayyad and Irani, 1992).

The numeric version of the TDIDT algorithm will thus look as follows:

Algorithm for numeric TDIDT

1. Use the entropy measure to find the optimal split for each of the numeric attributes;
2. Determine the attribute whose optimal split maximizes entropy and partition the example set along this attribute into two subsets;
3. If the termination criterion is not satisfied, repeat the procedure recursively for each subset.

Note that with each new subtree, the splitting cuts must be re-calculated. The optimal position for a cut is likely to be different in a different subset of examples.

1.4.2 Progressive Coverage: AQ Learning

The AQ learning is based on the idea of progressive coverage of the training data by consecutively generated decision rules. The approach has been implemented in a whole family of methods derived from an algorithm that was first published by Michalski (1969) and then adapted for machine learning purposes by Michalski (1973b). One the most recent versions of the family is described by Wnek et al. (1995).

The essence is to search for a set of rules (conjunctions of attribute-value pairs or, generally, arbitrary predicates) that covers all $\oplus$'s but none of the $\ominus$'s. Instead of partitioning the example sets, the AQ algorithm generalizes, step by step, the descriptions of selected positive examples, called *seeds.* This allows the rules to logically intersect whenever desirable.

Basic Principle

The principle is summarized in a simplified version of the algorithm presented below. Assume that the goal is to find a minimal set of *decision rules* characterizing the given concept. Decision rules will acquire the form:

$$\textbf{if } A_1 \ \textit{and} \ A_2 \ \textit{and} \ldots \textit{and} \ A_n \ \textbf{then } C$$

where C is the concept, and conditions A_i can acquire the common attribute-value form of $at_i = V$, or the more general form of $at_i = v_1 \vee v_2 \vee v_3 \ldots$, where an attribute can take on one of several values (linked by "internal" disjunctions).

AQ Algorithm

(simplified version)

1. Divide all examples into the subsets *PE* of $\oplus$'s and *NE* of $\ominus$'s;
2. Choose randomly or by design one example from *PE* and call it the *seed*;
3. Find a set of maximally general rules characterizing the seed. The limit of the generalization is defined by the set *NE*: a generalized description of the seed is not allowed to cover any object from *NE*. The obtained set of rules is called *star*;
4. According to some *preference criterion*, select the best rule in the star;
5. If this rule, jointly with all previously generated rules, covers all objects from *PE*, then stop. Otherwise, find another seed among the uncovered examples in *PE* and go to 3.

Step 3 is done by a special *star generation procedure* (Wnek et al., 1995). The *rule preference criterion* in step 4 should reflect the needs of the problem at hand. To this end, it can be a combination of various elementary criteria such as requirements to maximize the number of $\oplus$'s covered by the rule, minimize the number of involved attributes, maximize the estimate of generality (the number of covered $\oplus$'s divided by the number of all examples covered by the rule), minimize the costs of attribute-value measurements, and the like. It can also use attribute selection criteria used in decision tree learning, such as entropy, gain ratio, etc. The algorithm also makes it possible to construct a set of decision rules with different relationships among the individual rules. Rules may be logically intersecting, logically disjoint, or linearly ordered (which requires their evaluation in a sequential order).

The next example illustrates this simple version of the algorithm.

Table 1.4 A specification of a sample training set

example		$at1$	$at2$	$at3$	*classification*
	$e1$	y	n	r	$\oplus$
PE	$e2$	x	m	r	$\oplus$
	$e3$	y	n	s	$\oplus$
	$e4$	x	n	r	$\oplus$
	$f1$	x	m	s	$\ominus$
	$f2$	y	m	t	$\ominus$
NE	$f3$	y	n	t	$\ominus$
	$f4$	z	n	t	$\ominus$
	$f5$	z	n	r	$\ominus$
	$f6$	x	n	s	$\ominus$

Example

Suppose that the examples, described in terms of attributes $at1, at2$, and $at3$, are those listed in Table 1.4 and visualized in Figure 1.12. In Table 1.4, each row corresponds to one vector of attribute-values. Rows corresponding to positive examples are labeled by $\oplus$ and rows corresponding to negative examples are labeled by $\ominus$. Assume that the preference criterion favors rules that cover the maximum possible number of positive examples, and that the rules can intersect each other. The program leading to rule acquisition from the above examples will consist of the following steps:

Select 1st seed: $e1$

Pick the first negative example: $f1$. To create the star of seed $e1$ (that is the set of maximally general descriptions of $e1$) begin by creating the set of all descriptions of $e1$ that do not cover $f1$. They are:

$R1$: $(at3 = r \vee t)$
$R2$: $(at1 = y \vee z)$
$R3$: $(at2 = n)$

However, each of these descriptions also covers some of the negative examples. Therefore, these rules are specialized so as to exclude these negative examples, which is done by multiplying out the current rules by the negations of the negative examples, and applying absorption laws. The results are:

$R1'$: $(at1 = x \vee y)$ & $(at3 = r)$
$R2'$: $(at1 = y)$ & $(at3 = r \vee s)$

This is the star of $e1$. Suppose that the preference criterion recommends choosing from the star the rule that covers the most positive examples. Thus, $R1'$ is selected

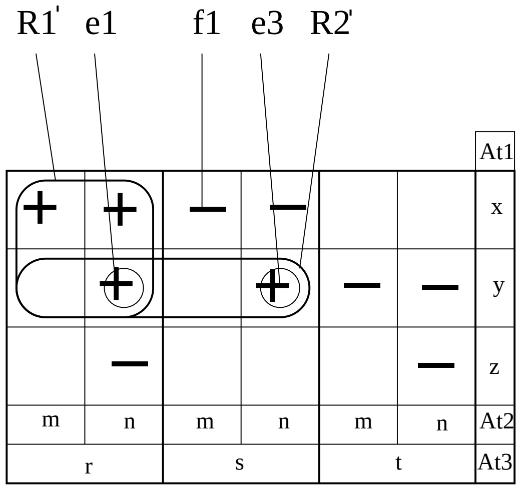


Figure 1.12 Visualization of the examples from Table 4

(it covers three examples while $R2'$ covers only two). The next step is to select a new seed from the positive examples still uncovered. Only one such example exists, $e3$.

Select next seed: $e3$
Again, determine the star for the new seed. Two rules are generated. The one that covers more examples is the same as $R2'$ shown above.

As there are no uncovered examples left, the selected rules R1' and R2' constitute a complete and consistent description of the concept, which optimizes the assumed preference criterion:

$$\begin{array}{llcl} R1' & (at1 = x \vee y) & \& & (at3 = r) \\ R2' & (at1 = y) & \& & (at3 = r \vee s) \end{array}$$

Learning systems based on the AQ-algorithm can easily incorporate background knowledge because such knowledge is often also represented by decision rules. Nevertheless, we will not treat this feature here because the employment of background knowledge is more typical for predicate-logic-based learning systems that will be addressed later.

The Two-Tiered Approach

In AQ-based methods, the output has the form of decision rules. To handle context imprecision and noise in the data, the *two-tiered* approach has been invented (for a detailed treatise, see Michalski, 1990). This approach also facilitates handling context sensitivity and improves comprehensibility in the AQ-paradigm.

The principal idea is to split the concept description into two parts: *base concept representation* (BCR) containing the explicit concept characterization stored in the learner's memory; and the *inferential concept interpretation* (ICI) containing a set of inference rules to be applied in the recognition phase.

For illustration, the BCR can contain the production rule

$$\textbf{if } A_1 \ and \ A_2 \ and \ldots and \ A_n \ \textbf{then } X$$

and the ICI can contain the interpretational rule

'at least 3 conditions out of $A_1 \ldots A_n$ must be satisfied.'

A very simple example of the two-tiered approach is the TRUNC method which consists of the following general steps:

1. Use AQ to derive the initial set of rules;
2. Determine the 'importance' of each rule. A simple measure of importance is the number of positive examples covered by the rule. Save only the most important rules in BCR;
3. Define the ICI procedure for the correct recognition of uncovered examples by the rules in BCR.

When the induced representation is used for recognition, the new example is assigned the class of that rule in BCR that provides the "best match" according to ICI. A *flexible matching* procedure was proposed for that purpose.

Readers interested in more details about the two-tried approach and some of its implementation are referred to Michalski (1990), Zhang and Michalski (1991), Bergadano et al. (1992), or Kubat (1996).

1.4.3 Assessment of Learning Algorithms

During the last two decades, machine learning researchers have come up with so many learning algorithms that criteria for their assessment and taxonomization are indispensable. Some of them are summarized in Table 1.5.

Perhaps the most important criterion is *accuracy.* As the usual motivation of concept learning is correct identification of future instances, the success of learning is quite straightforwardly measured by the percentage of correctly classified testing examples. Suppose that the learner is asked to classify a set of 200 examples, out of which 100 are $\oplus$ and 100 are $\ominus$. If the learner correctly classifies 80 $\oplus$'s and 60 $\ominus$'s, then the accuracy is $\frac{80+60}{200} = 0.7$, that is 70%.

Sometimes it is more important to know how many times the system recognizes the positives, while the negatives can be much less critical (or vice versa). In that

Table 1.5 Assessment of ML-algorithms

criterion	comments
accuracy	percentage of correctly classified $\oplus$'s and $\ominus$'s
efficiency	# examples needed, computational tractability
robustness	against noise, against incompleteness
special requirements	incrementality, concept drift
concept complexity	representational issues (examples and BK)
transparency	comprehensibility for the human user

case we draw distinction between two kinds of error: negatively classified positives (erros of omission) and positively classified negatives (errors of commision). These errors can tell the user that the learned concept description is too specialized or too general. In the case of overspecialization, the learner will tend to misclassify positive testing examples more often than negatives. In the case of overgeneralization, the learner will more frequently fail by misclassifying negative examples (classifying them as positives).

Ideally, the learner should develop a hypothesis (internal description of the concept) that is *consistent* (does not cover any $\ominus$) and *complete* (covers all $\oplus$'s). Inconsistency and incompleteness are illustrated in Figure 1.13. In the space described by two numeric variables (one represented by the horizontal axis and the other by the vertical axis) the concept to be found is the shaded area below the oblique line separating the upper and the lower part of the rectangle. The description represented by the oval is incomplete because it does not cover the entire area of the concept. The rectangle $ABCD$ is inconsistent because it covers also a part of the negative area.

Classification accuracy is only one of the criteria for the assessment of machine learning algorithms. The learner should also be efficient—able to achieve a certain level of accuracy with the minimum *number of learning examples*. The teacher might not be always able to provide many examples and, anyway, the ability to learn fast is a sign of intelligence. Also *computational requirements* are of interest—how much time the computer needs to arrive at a good hypothesis.

Another criterion is concerned with the *comprehensibility* of the induced concept description. It is often important that the generated description be understandable so that the user learns from it something new about the application domain. Such a description can be used by humans directly and understood as an enhancement to their own knowledge. This criterion also applies when the induced descriptions are to be employed in a knowledge-based system whose behaviour should be transparent. The criterion of comprehensility typically separates machine learning in artificial intelligence from other forms of learning, including neural networks and statistical methods. As mentioned in Section 1.2 of this chapter, Michie (1988) further elaborated criteria that are germane to comprehensibility. Early effort in this direction was reported by Michalski (1983).

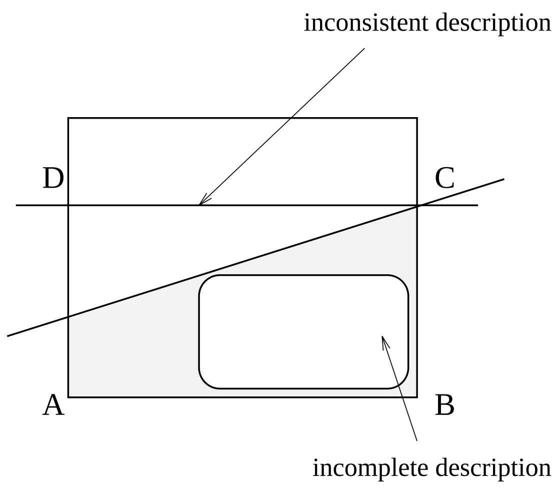


Figure 1.13 Illustration of incompleteness and inconsistency

A serious intricacy in learning is the presence of noise (error) in the examples and/or in their class labels. The measurement devices providing the attribute values may be imprecise or improperly adjusted, the attribute values supplied by the teacher may be too subjective, an accident can damage the data, or some of the data may get lost. Many unpleasant misfortunes can happen to the examples before their presentation to the machine. Quite logically, *robustness against noise* and *robustness against missing information* (e.g., missing attribute values) is required. However, this is no dogma! Some learning tasks are characterised by the presence of noise and missing information and others are characterized by perfect examples. The concrete application must decide whether this criterion matters.

The specific conditions of the application should really be taken seriously. For instance, the user can demand that the learner be able to acquire the knowledge on-line from a stream of examples arriving one by one (as opposed to the situation where all examples are present from the very beginning). Imagine that a new example is presented at the moment when an initial concept description has already been developed. In traditional batch algorithms, this would mean to re-run the entire learning procedure on all data. Such behavior can hardly be called intelligent. The learner should be capable of refining the previous knowledge, to *learn incrementally*, like humans.

Incremental learning is particularly important in the presence of *concept drift* or *evolution* (Widmer and Kubat, 1993; Widmer and Kubat, 1996). In some domains, the meaning of a concept changes from time to time. This can be illustrated by such terms as 'fashionable dress' or 'democracy.' The learner should be able to follow the drift in the same way as humans do.

Finally, the designer must consider the representational issues, which means to respect the language used to describe the examples as well as the language in which the background knowledge has been encoded. The reader has already seen that various representational languages differ in their expressiveness. For instance, while the TDIDT algorithm pertains to attribute-value logic, more advanced systems that will be discussed in the next section are able to learn concepts from examples described by predicate expressions.

1.5 How can Predicate Logic be Used?

As already mentioned, attribute-value languages are very useful but have their limitations. Even though a lot can be described in this way, logical descriptions with predicates that describe relations among objects or their parts are certainly more powerful. Consider the background knowledge about family relations shown in Figure 1.14 (for simplicity, the names of the persons are '1,' '2,' ... instead of 'John,' 'Bill,' ...). Here, the family relations are described by the single predicate `parent(X,Y)`, for instance `parent(1,2)` means `1` is parent of `2`.

The family tree can be encoded by the following set of relationships:

$$\texttt{parent} = \{(1,2)(1,3)(3,4)(3,5)(3,6)(4,7)\}$$

Suppose that with these relationships in the background knowledge, the system wants to learn the meaning of `grandparent(X,Y)`. Suppose, further, that the teacher provides the following positive examples of the concept:

```
grandparent(1,4)
grandparent(1,5)
grandparent(1,6)
grandparent(3,7)
```

Again, these are easier to encode by the set of relationships:

$$\texttt{grandparent} = \{(1,4)(1,5)(1,6)(3,7)\}$$

All other family relations among the persons $1, \ldots 7$ are assumed to be negative examples of `grandparent`.

Of course, these relations can be described also by attribute values. For instance, each possible pair of persons (X, Y) can be represented by a Boolean attribute whose truth value is determined by the truth value of the predicate relation `parent(X,Y)`. However, such descriptions are rather cumbersome and inflexible.

Higher-level description languages require more sophisticated algorithms and the task of the next section is to present a few ideas for learning in predicate logic. Effort will be made to begin with principles that are already known and gradually proceed to more elaborate techniques.

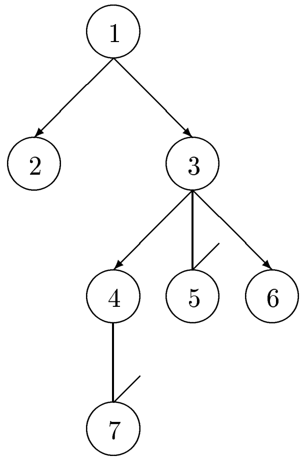


Figure 1.14 Family relations

The most popular approach to learning in predicate logic is Inductive Logic Programming (ILP) which is currently an intensively studied branch of machine learning. For more detailed coverage see the books by Muggleton (1992) or Lavrač and Džeroski (1994).

1.5.1 Learning Horn Clauses from Relations

Suppose you are asked to write a program capable of learning the concept of `grandparent` from examples of family relationships. The concept is to be described by Horn clauses. What learning strategies would you apply?

Trying to express the concept in the simplest possible way, you will start with the language where only those arguments are allowed in the body that appear also in the head. Assume that the concept description is a set of Horn clauses in the form of:

C_1 :- $L_{11}, L_{12}, \ldots L_{1m}$
C_2 :- $L_{21}, L_{22}, \ldots L_{2m}$
$\ldots$

where the predicate C_i is the head of a clause and the literals L_{ij} form the body of the clause. The commas separating the literals in the body indicate that they are linked by conjunction. Each of the literals represents a relation that has $n \geq 0$ arguments.

Recall the divide-and-conquer principle or the AQ method, where the learner started with a relatively general description involving a single attribute, and then gradually specialized by adding more conditions. Why not use the same principle here as well?

Since adding a literal to a clause body has a similar specializing effect as adding a condition to a decision rule in AQ or appending a node at the bottom of a decision tree, a good strategy is to start with a clause consisting solely of the head and then specialize it by adding literals to its body.

As there are no other predicates in the background knowledge except for *parent* and as only those variables are allowed in the body that appear also in the head, possible clauses to define grandparent in this language are:

```
grandparent(X,Y) :- parent(X,Y).
grandparent(X,Y) :- parent(Y,X).
grandparent(X,Y) :- parent(X,X).
grandparent(X,Y) :- parent(Y,Y).
```

Unfortunately, none of these clauses covers any of the positive examples. Obviously, the restriction should be relaxed allowing also for one argument that does not appear in the head of the clause. Four literals of this kind can be constructed: `parent(X,Z)`, `parent(Y,Z)`, `parent(Z,X)` and `parent(Z,Y)`. Suppose that the system selects the following option:

```
grandparent(X,Y) :- parent(X,Z).
```

Let us now examine for which triplets (X,Z,Y), satisfying this clause, the head of the clause represents a positive example and for which it represents a negative example. Note that there are $7^3 = 343$ possible triplets (X,Z,Y).

$\oplus$: (1,2,4) (1,2,5) (1,2,6) (1,3,4) (1,3,5) (1,3,6) (3,4,7) (3,5,7) (3,6,7)
$\ominus$: (1,2,1) (1,2,2) (1,2,3) (1,2,7) (1,3,1) (1,3,2) (1,3,3) (1,3,7) (3,4,1)
(3,4,2) (3,4,3) (3,4,4) (3,4,5) (3,4,6) (3,5,1) (3,5,2) (3,5,3) (3,5,4)
(3,5,5) (3,5,6) (3,6,1) (3,6,2) (3,6,3) (3,6,4) (3,6,5) (3,6,6) (4,7,1)
(4,7,2) (4,7,3) (4,7,4) (4,7,5) (4,7,6) (4,7,7)

A closer look reveals that the positive triplets include all 4 positive examples and the negative triplets include 17 negative examples: (1,1) (1,2) (1,3) (1,7) (3,1) (3,2) (3,3) (3,4) (3,5) (3,6) (4,1) (4,2) (4,3) (4,4) (4,5) (4,6) (4,7).

We say that the clause `grandparent(X,Y) :- parent(X,Z)` is inconsistent because it covers also negative examples. The inconsistenty can be reduced by a properly chosen specialization of the clause. This can be accomplished by adding some of the other permited literals to it. Let us try the following:

```
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

This clause covers the following triplets (X,Z,Y):

$\oplus$: (1,3,4) (1,3,5) (1,3,6) (3,4,7)
$\ominus$: none

Since all of the positives and none of the negatives are covered, the learner is satisfied with this clause and stops here.

What if some other literal instead of `parent(Z,Y)` were added? Consider the next clause:

```
grandparent(X,Y) :- parent(X,Z), parent(Y,Z).
```

Despite the fact that the clause does not cover any negative examples, the utility of the clause is questionable because it does not cover any positives either. Obviously, a suitable criterion to decide what literal to add to a clause is needed.

Let us examine a more complicated concept, say, `ancestor`. Based on the 7 persons whose family relations are known from Figure 14, the following positive examples are provided:

`ancestor` = {(1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (3,4) (3,5) (3,6) (3,7) (4,7)}

The search for the best literal (involving the same arguments as the head) arrives at the following description:

```
ancestor(X,Y) :- parent(X,Y).
```

This clause is consistent, so there is no need for specialization. The system will store it and—because the description is incomplete—will attempt to find an alternative clause covering those positives that lie outside the reach of the first clause. The result is then interpreted in such a way that at least one of the clauses should cover the example if it is to be a positive instance of the concept. Repeating the procedure, the following three clauses can be determined:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Z), parent(Z,W), parent(W,Y).
```

Even though these clauses cover all learning examples, we know that they are not complete because they cover only four generations. A reader acquainted with logic programming, would recommend a *recursive* description:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

A learning system can find this description only if it is allowed to define the predicate `ancestor` in terms of the initial understanding of the concept. This is the principle of recursion. To start with, the learner uses the predicate `parent`. After formulating the first clause, the learner also uses the predicate `ancestor`.

The procedure just outlined forms the kernel of the system FOIL, developed by Quinlan (1990b). The following algorithm formalizes the approach:

Algorithm FOIL

```
1. Initialize the clause by defining the head that represents the name of the concept
   to be learned and leave the body empty;
2. While the clause covers negative examples do:
       Find a "good" literal to be added to the clause body;
3. Remove all examples covered by the clause;
4. Add the clause to the emerging concept definition. If there are any uncovered pos-
   itive examples then go to 1.
```

The only remaining problem at this moment is how to find a "good" literal to be added to the clause (step 2 of the algorithm). To this end, FOIL makes use of an information criterion similar to that employed in the induction of decision trees.

Denote by T_i^+ the number of $\oplus$'s covered by the conjunction $L_1, L_2, \ldots, L_{i-1}$ and denote by T_i^- the number of $\ominus$'s covered by the same conjunction. Then the information provided by signalling a positive example among all the examples covered by this clause is:

$$I_i = -\log(\frac{T_i^+}{T_i^+ + T_i^-})$$

After adding a new literal L_i this information becomes:

$$I_{i+1} = -\log(\frac{T_{i+1}^+}{T_{i+1}^+ + T_{i+1}^-})$$

The success of literal L_i is measured by two factors:

1. The remaining information I_{i+1} (the smaller, the better);
2. The number T_i^{++} of $\oplus$ that remain covered by the clause after adding L_{i+1} (the higher, the better).

Accordingly, FOIL's measure of success is defined as:

$$Gain(L_i) = T_i^{++} \times (I_i - I_{i+1})$$

It is instructive to observe how FOIL combines the two approaches outlined in the previous section. In principle, the program carries out the AQ-algorithm, trying to cover the positive space by a set of clauses. In the inner loop, a clause is stepwise specialized by a procedure controlled by a function similar to that used in the divide-and-conquer method.

1.5.2 Inverse Resolution

Returning to our search-oriented conception of learning, we can easily see that FOIL exploits two search operators: the generalizing *add_a_clause* operator and the specializing *add_a_literal* operator. Providing also the complementary operators *delete_a_clause* for specialization and *delete_a_literal* for generalization, we arrive at the elementary repertoir of learning in first-order logic. The operators can be exemplified by the following four steps gradually changing the clause x :- a, b into x :- d, e.

– add a clause $\quad x \text{ :- } a, b \quad \Rightarrow \left\{ \begin{array}{l} x \text{ :- } a, b \\ x \text{ :- } c, d \end{array} \right\}$

– delete a clause $\quad \left\{ \begin{array}{l} x \text{ :- } a, b \\ x \text{ :- } c, d \end{array} \right\} \Rightarrow x \text{ :- } c, d$

– add a literal $\quad x \text{ :- } c, d \quad \Rightarrow x \text{ :- } c, d, e$

– delete a literal $\quad x \text{ :- } c, d, e \quad \Rightarrow x \text{ :- } d, e$

The main task of this subsection is to show how this collection can be extended by the following alternative inductive search operators:

– identification $\quad \left\{ \begin{array}{l} a \text{ :- } b, x \\ a \text{ :- } b, c, d \end{array} \right\} \quad \Rightarrow \quad \left\{ \begin{array}{l} a \text{ :- } b, x \\ x \text{ :- } c, d \end{array} \right\}$

– absorption $\quad \left\{ \begin{array}{l} x \text{ :- } c, d \\ a \text{ :- } b, c, d \end{array} \right\} \quad \Rightarrow \quad \left\{ \begin{array}{l} x \text{ :- } c, d \\ a \text{ :- } b, x \end{array} \right\}$

– inter-construction $\quad \left\{ \begin{array}{l} a \text{ :- } v, b, c \\ a \text{ :- } w, b, c \end{array} \right\} \quad \Rightarrow \quad \left\{ \begin{array}{l} a \text{ :- } u, b, c \\ u \text{ :- } v \\ u \text{ :- } w \end{array} \right\}$

– intra-construction $\quad \left\{ \begin{array}{l} a \text{ :- } v, b, c \\ a \text{ :- } w, b, c \end{array} \right\} \quad \Rightarrow \quad \left\{ \begin{array}{l} a \text{ :- } v, u \\ a \text{ :- } w, u \\ u \text{ :- } b, c \end{array} \right\}$

All of these operators can be derived from the *resolution principle* that is very popular in artificial intelligence: Denote two disjunctions of predicates by C_1 and C_2 and denote an arbitrary predicate by l. The resolution principle is deductive, and states the following:

If $(C_1 \vee l)$ is true and $(C_2 \vee \neg l)$ is true, then also $(C_1 \vee C_2)$ is true.

Put another way, two disjunctive expressions are assumed. One of them contains the literal l and the other one contains its negation $\neg l$. The disjunction of the two expressions, where l and $\neg l$ have been deleted, is also true and is called *resolvent*. The principle is depicted in Figure 1.15.

Interestingly, the basic schema of resolution can be reversed. Knowing the resolvent and one of the original strings, we construct the other original string. Depending on

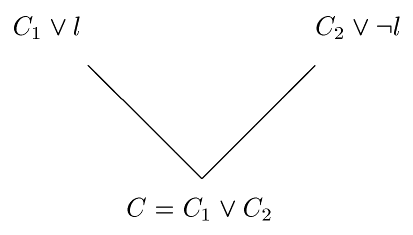


Figure 1.15 Resolution principle

whether the available clause contains the positive or the negated form of the predicate l, we speak about *identification* or *absorption*, respectively. The whole derivation scheme for both of them is shown in Figure 1.16. We will explain only identification, the derivation of absorption is analogous. The operators of inter-construction and intra-construction can be derived by slightly more complicated procedures that will not be described here.

Suppose we are given two clauses, a :- b, x and a :- b, c, d. Let the former be called 'original' and the latter be called 'resolvent.' The task is to find the unknown clause that would, together with the original, produce the resolvent. For simplicity, no arguments of the predicates are assumed.

Knowing that the formula A :- B can be rewritten as $A \vee \neg B$, we transform the two clauses into $a \vee \neg b \vee \neg x$ and $a \vee \neg b \vee \neg c \vee \neg d$. Both of the two clauses share the substring $a \vee \neg b$. Furthermore, the resolvent contains also the substring $\neg c \vee \neg d$ that might have been inherited from the unknown clause. The original, in turn, contains also the predicate $\neg x$ and, hence, its negation, x, is expected to appear in the unknown clause. Concatenating the contribution of the resolvent and the original, we will arrive at the string $\neg c \vee \neg d \vee x$. Turned back into the Horn clause, the string will change into x :- c, d. This newly created clause replaces the resolvent.

Unfortunately, in real-world applications the task gets complicated. Thus in the case of the reversed resolution, the learner will have to find proper substitutions of the arguments in the predicates l and $\neg l$ so that they are compatible. For instance, the predicates p_1 = `parent(john,bill)` and p_2 = `parent(X,eve)` are compatible only under the substitutions θ_1 = {`john`/X, `bill`/Y} in the first predicate, and θ_2 = {`eve`/Y} in the second predicate. The reader is referred to Muggleton (1991) for a more detailed discussion of this and related problems.

1.5.3 Theory Revision

Sometimes a body of *background knowledge* is available to guide the learning process. For illustration, suppose that the background knowledge contains partial information about the family relations similar to those in Figure 1.14 and that the additional infor-

– identification $\left\{ \begin{array}{l} a \text{ :- } b, x \\ a \text{ :- } b, c, d \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} a \text{ :- } b, x \\ x \text{ :- } c, d \end{array} \right\}$

a)
$$\begin{tikzcd} ? \arrow[dr, dash] & & a \text{ :- } b, x \arrow[dl, dash] \\ & a \text{ :- } b, c, d & \end{tikzcd}$$

b)
$$\begin{tikzcd} ? \arrow[dr, dash] & & a \vee \neg b \vee \neg x \arrow[dl, dash] \\ & a \vee \neg b \vee \neg c \vee \neg d & \end{tikzcd}$$

c)
$$\begin{tikzcd} \neg c \vee \neg d \vee x \arrow[dr, dash] & & a \vee \neg b \vee \neg x \arrow[dl, dash] \\ & a \vee \neg b \vee \neg c \vee \neg d & \end{tikzcd}$$

d)
$$\begin{tikzcd} x \text{ :- } c, d \arrow[dr, dash] & & a \text{ :- } b, x \arrow[dl, dash] \\ & \bullet & \end{tikzcd}$$

– absorption $\left\{ \begin{array}{l} x \text{ :- } c, d \\ a \text{ :- } b, c, d \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} x \text{ :- } c, d \\ a \text{ :- } b, x \end{array} \right\}$

a)
$$\begin{tikzcd} x \text{ :- } c, d \arrow[dr, dash] & & ? \arrow[dl, dash] \\ & a \text{ :- } b, c, d & \end{tikzcd}$$

b)
$$\begin{tikzcd} x \vee \neg c \vee \neg d \arrow[dr, dash] & & ? \arrow[dl, dash] \\ & a \vee \neg b \vee \neg c \vee \neg d & \end{tikzcd}$$

c)
$$\begin{tikzcd} x \vee \neg c \vee \neg d \arrow[dr, dash] & & \neg x \vee a \vee \neg b \arrow[dl, dash] \\ & a \vee \neg b \vee \neg c \vee \neg d & \end{tikzcd}$$

d)
$$\begin{tikzcd} x \text{ :- } c, d \arrow[dr, dash] & & a \text{ :- } x, b \arrow[dl, dash] \\ & \bullet & \end{tikzcd}$$

Figure 1.16 Derivation of the *identification* and *absorption* operators

mation about the sex of the individual persons is provided in terms of the predicates `male` and `female`. Being told that Jack is father of Bill, the learner is expected to derive the definition of the predicate `father`, previously not present in the background knowledge. The approach described here roughly builds on the algorithm that forms the core of the system CLINT—see de Raedt (1992).

The learner starts with some strongly restricted language, for instance, with the initial constraint demanding that each literal in the clause body is allowed to contain as arguments only those constants and variables that appear also in the head, as in `p(X,Y) :- q(X,Y),r(X)`. In the case of `father(jack,bill)`, this means that the system searches for all literals containing no other arguments except for `jack` and `bill`. Having found such literals, the system connects them with conjunctions.

For illustration, suppose that the system's background knowledge contains, among other things, the following predicates:

```
      ⋮
parent(jack,bill).
parent(tom,jack).
parent(tom,eve).
parent(eve,bill).
male(tom).
male(jack).
male(bill).
female(eve).
painter(bill).
singer(jack).
      ⋮
```

If there are no other predicates containing either of the arguments `jack` or `bill`, the attempt to construct the concept in terms of the above simple language will end up in the following description:

```
father(jack,bill) :- parent(jack,bill), male(jack), male(bill),
                     painter(bill), singer(jack).
```

Having found this concrete clause, the learner will generalize it by turning constants into variables, thus obtaining what is referred to as *initial clause.*

```
father(X,Y) :- parent(X,Y), male(X), male(Y), painter(Y), singer(X).
```

Obviously, this method of clause construction is inevitably rather blind, and even a superficial look at the 'invented' clause reveals that there is something wrong with it: the fact that `jack` is `bill's` father has nothing to do with `bill` being male, much less with his profession. To address this issue, the authors of CLINT provided the learner with the ability to refine the initial description of the concept by way of a simple dialog with the user.

During the dialog, the learner examines each predicate in turn and checks its necessity by creating new examples and asking the user to classify them. For instance the question

Is `father(tom,jack)` true?

is positively answered by the user, indicating that the literal `painter(Y)` is unnecessary (`jack` is singer and, still, `tom` is his father).

The next question should check whether it is necessary that Y be male. Knowing that `eve` is female, the learner finds in the background knowledge the literal `parent(tom,eve)` and asks the user the following question:

Is `father(tom,eve)` true?

A positive answer indicates that also `male(Y)` was unnecessary. On the other hand, the question:

Is `father(eve,bill)` true?

will be answered negatively, which means that `male(X)` cannot be discarded from the clause.

Obviously, in the course of this verification, the original clause can totally alter or, even, that all literals will be deleted from the body. Alternatively, it can happen that no initial clause is found. In both of these cases, the system proceeds by alleviating some of the constraints, for instance, the one imposed on the predicate arguments. Then the body predicates will be allowed to contain one and only one argument that does not appear in the head, as is the case of the clause:

```
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

In this way, the system generalizes the concept description with the objective to cover also those positive instances (created by the system and presented to the user) that have not been covered by the previous description.

Of course, the description can become too general in the sense that it covers also negative examples. In this case, respective measures must be taken to rectify this inconvenience. The solution implemented in CLINT consists of building the explanation tree for the negative example, identifying the culprit clause c responsible for the coverage of the negative example, deleting c from the knowledge base, and re-generalizing the resulting knowledge structure so that all positive examples that have previously been covered by c become covered again.

For more about the system CLINT see de Raedt (1992).

1.5.4 Constructive Induction

Let us now turn our attention to the problem of determining the appropriate representation space for learning, that is attributes or predicates relevant to the problem at hand. In standard methods, the learner analyzes the examples, describes them in terms of some predefined set of attributes or predicates, and produces the expected concept description using the operators of the description language, for instance conjunctions, disjunctions, and negations of the attribute values or predicates. Michalski (1991) calls this simple kind of induction *empirical.* It is carried out by the simplest versions of the TDIDT and AQ algorithms, where the concept was described by a subset of the attributes that have been used to describe the learning examples.

In some approaches, such as inverse resolution, the learning algorithm itself constructs new predicates that are to facilitate the learning process. Often, this method necessitates an interactive learning procedure. Since the concept is invented by the machine, the user (possessing more knowledge about which predicates make sense) is asked to acknowledge the new predicate and assign it a name. Meat-eating animals with claws can be accepted and given the name `predators`; big animals with yellow skin will probably not make a useful concept, and will be rejected by the user in the belief that the two features have appeared together by mere coincidence.

At this point, the constructive induction in *analogy-based* learning should be mentioned. Although the issue of analogy is discussed later, the idea of *second-order schemata* as implemented in the system CIA (see de Raedt, 1992) falls into the context of constructive induction.

The essence of the system consists in storing typical schemata of predicate expressions, such as:

```
p(X,Y) :- q(X, XW), q(Y, YW), r(XW, YW)
```

where not only the arguments `X, XW, Y`, and `YW`, but also the predicates `p, q`, and `r` represent variables. Thus the previous schema can be, by suitable substitutions, instantiated into the following clauses (the respective substitutions are also provided):

```
lighter(X,Y) :- weight(X,XW), weight(Y,YW), less(XW,YW)
```
$\Theta = \{p/\text{lighter},\ q/\text{weight},\ r/\text{less}\}$

```
same-color(X,Y) :- color(X,XC), color(Y,YC), eq(XC,YC)
```
$\Theta = \{p/\text{same-color},\ q/\text{color},\ r/\text{eq}\}$

```
brothers(X,Y) :- son(X,XP), son(Y,YP), eq(XP,YP)
```
$\Theta = \{p/\text{brothers},\ q/\text{son},\ r/\text{eq}\}$

The second-order schemata lend themselves quite straightforwardly to constructive induction. In CIA's setting, this happens whenever the system finds out that, after proper substitutions, the body of a schema becomes a subset of the body of some clause whose head is unknown. For illustration, the schema:

```
p(X,Y) :- q(X, XW), q(YW, Y), r(XW, YW))
```

can become a subset of the clause

```
:- male(F), male(C),  parent(F,M1), parent(M2,C), eq(M1,M2)
```

after the substitutions:

$\Theta = \{q/\text{parent}, r/\text{eq}\}$ and
$\rho = \{\text{X/F, Y/C, XW/M1, YW/M2}\}$

The instantiated schema is:

```
p(F,C) :- parent(F,M1), parent(M2,C), eq(M1,M2)
```

If prompted, the user will certainly acknowledge the new clause as sensible and will suggest the name `grandparent` for predicate `p`.

To conclude, the principle of constructive induction is very powerful and, combined with deeper studies of the nature of various representation languages, is generally considered as a very important research topic.

1.6 Artificial Discovery

The issue of Artificial Discovery is an instructive illustration of the general way of thinking in machine learning and, as such, deserves a brief elaboration here, even though it does not directly relate to the applications reported in the subsequent chapters.

So far, our interest was focused on *supervised* learning, where the learner seeks to develop a concept description from examples that have been preclassified by the teacher. The present section departs from this path in that it concentrates on *unsupervised* learning whose task is to generate conceptual taxonomies from non-classified objects.

Actually, this is what scientists (say, biologists) have been doing for centuries, developing such categories as vertebrates, subcategories as mammals or birds, and the like. The utility of the taxonomies and categories is obvious: any object that has been recognized as a member of a certain category inherits the general properties of the category. Being told that a horse is a mammal, we immediately know whether the animal lays eggs, whether it can fly, or whether its skin is covered with fur or feathers.

A related task is carried out by some traditional statistical techniques such as *cluster analysis*. The dots in Figure 1.17 represent objects described by two numeric attributes, x and y. Apparently, the objects can be partitioned into two groups which are easy to discover by relatively simple algorithms exploiting the notion of similarity as measured by the numeric distance between objects. Unfortunately, not every kind of similarity can be assessed numerically. Indeed, is the distance between cat and giraffe

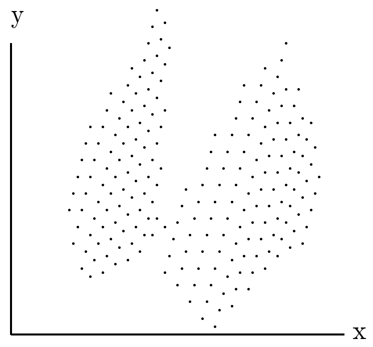


Figure 1.17 Traditional task for cluster analysis

greater than the distance between dog and elephant? Even though these distances can be transformed into numbers, any such a transformation would be difficult and subjective.

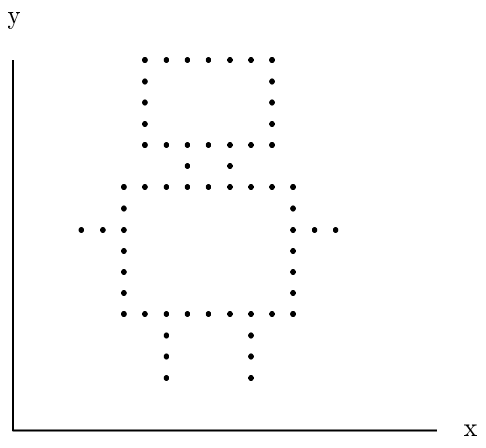
To come to understand another important issue, consider the task depicted in Figure 1.18. Here, the objects are already pre-odered in a way that can be described conceptually, and several interpretations can be offered depending on the particular context. Obviously, traditional distance-based cluster analysis will hardly produce reasonable outcomes on these data and yet, for humans, this task appears to be trivial. In machine learning, the search for concepts hidden in a set of objects is studied by the discipline known under the name *Concept Formation.*

To procede one step further, one might want to discover not only concepts but also laws defining the relations among them, with the ambition to create a computer-based system to assist human researchers in such disciplines as chemistry or biology. Even though to expect implementation of artificial scientists would be perhaps too optimistic, a few remarkable systems addressing simple discovery tasks have already been developed.

Below, we devote one subsection to Concept Formation and one subsection to Automated Discovery.

1.6.1 Concept Formation

Gennari, Langley, and Fisher (1989) divide the field of unsupervized concept learning into two different subfields: *Concept Discovery*, deriving concepts from a batch, and incremental *Concept Formation* that gradually forms the concepts from a stream of examples.



2 rectangles and 6 lines
3 rectangles and 4 lines
robot
part of a city map
⋮

Figure 1.18 Concept-discovery task of machine learning

Concept Discovery by Conceptual Clustering

Conceptual clustering has been introduced as a novel form of clustering in which clusters are not just collections of entities possessing numerical similarity. Rather, the clusters are understood as groups of objects that together represent a concept. Conceptual clustering produces not only clusters but also descriptions of the related concepts. The system *CLUSTER* (see Michalski and Stepp, 1983) is anchored in the same seed-and-star philosophy as AQ, and actually can be considered as its extension to the realm of non-classified examples.

A simple task for concept discovery is depicted in Figure 1.19. Eight non-classified examples are described by three attributes. Attribute *at1* is symbolic, attribute *at2* acquires integer values, and attribute *at3* acquires integer values that can be decomposed into three symbolic values. Background knowledge provides the type and range for each of the attributes and defines the decomposition of *at3*.

Michalski and Stepp's idea is that the learner picks *k* seeds and treats them as if they represented *k* different clusters. In a simplified version, the procedure can be summarized by the folowing algorithm:

example	at1	at2	at3
e1	a	2	110
e2	b	4	100
e3	b	2	9
e4	b	3	10
e5	c	5	20
e6	c	4	15
e7	b	5	200
e8	b	4	50

Background Knowledge:

$at1 : [a, b, c]$

$at2 : [2..6]$

$at3 : [1..300]$

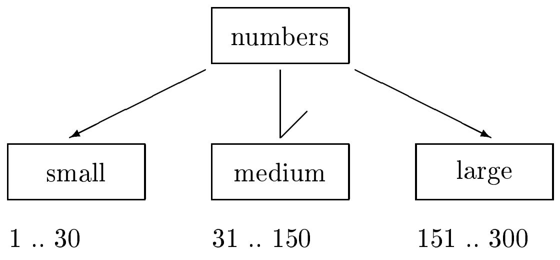


Figure 1.19 Simple task for concept discovery

CLUSTER-Algorithm

1. Pick k seeds, where k is a user-specified parameter;
2. Build k stars, each star being understood as a collection of the most general descriptions of one seed; the limits for the seed generalization are all other seeds;
3. Select from each star one rule so that each rule in the generated rule-set has the minimum logical intersection with the remaining rules, and the logical union of these rules covers the maximum number of instances.
4. If there are any uncovered instances, find rules with which they have the best "fit". Refine the rules so that they together cover all instances and are all logically disjoint. Instances that belong to an intersection of rules are redistributed so that each is covered by one and only one rule. At this moment, each rule represents a set of examples. From each of these sets, select a new seed;
5. Repeat the above procedure for the new seeds and keep repeating the entire procedure as long as each new solution makes an improvement over the previous solutions. Repeat for several different values of k, e.g., $k = 2, 3, ...7$, and determine the highest "quality" solution, the quality being determined on the basis of various criteria such as the simplicity of the rules in a clustering and their sparseness (measuring the degree of generalization of each rule over the instances covered by the rule; Michalski and Stepp, 1983).

Let us now apply this algorithm to the data from Figure 1.19. For simplicity, assume that numerical values have been replaced by symbolic values "small", "medium" or "large", according to Figure 1.19 (normally, CLUSTER itself proposes the most appropriate clusters of numerical values). The algorithm will roughly perform the following steps (k is assumed to be 2):

Choose randomly 2 seeds, say, *e1* and *e5*. Their descriptions are:

des(e1): $(at1 = a)\&(at2 = 2)\&(at3 = large)$
des(e5): $(at1 = c)\&(at2 = 5)\&(at3 = small)$

The initial stars are:

star(e1): $(at1 \neq c), (at2 \neq 5), (at3 \neq small)$
star(e5): $(at1 \neq a), (at2 \neq 4), (at3 \neq large)$

Each star has three one-condition rules and rules from different stars intersect. From each star, one rule is selected and modified in such a way that the rules in the obtained rule-set are logically disjoint and their union covers all instances (this is done by NID and PRO procedures described in Michalski and Stepp, 1983). The result is the following solution:

Cluster 1: $(at1 = a \vee b)\&(at2 = 2 \vee 3)$
Instances: e1, e3, e4

Cluster 2: $(at1 = b \vee c)\&(at2 = 4 \vee 5)$
Instances: e2, e5, e6, e7, e8

Since selecting new seeds from these above rules does not lead to an improved clustering, the above rules constitute the proposed solution for $k = 2$. A repetition of the algorithm for higher values of k also does not improve the solution, so that the above is the final result. For more details, see Michalski and Stepp (1983).

Crisp Conceptual Hierarchies

The algorithms for concept discovery from fixed sets of non-classified examples tend to be prohibitively expensive. On the other hand, concept formation algorithms attempt to simulate the development of taxonomies in humans as closely as possible in the sense that this process is supposed to be incremental. Moreover, emphasise is usually (not always) laid on generating *hierarchically ordered* concepts.

Most of these systems combine the process of classification and learning: whenever a new example arrives, the system integrates it (classifies) into the current knowledge

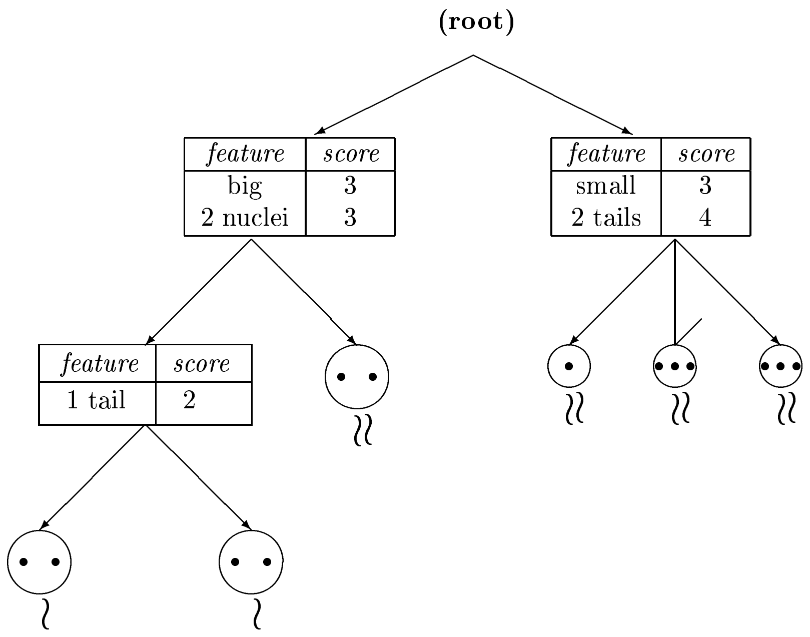


Figure 1.20 Representation in UNIMEM

structure. This is depicted in Figure 1.20, where the system UNIMEM (Lebowitz 1987) has developed a taxonomy from six examples of cells, described by their size, number of nuclei, and number of tails. When another example arrives (small, two-tailed, one nucleus), it is found to be most similar to the triplet in the righ-hand branch of the knowledge tree. Confronted with this new experience, the system creates a new subclass as shown in Figure 1.21.

Concept-formation algorithms have typically been conceived as search systems—defined by initial state, termination criterion, search operators, search strategy, and, of course, representational issues. The initial state is given by the description of the first example whereas the final state is the knowledge structure after the last example—the system is supposed to learn as long as the examples keep coming. The most common search policy is hill-climbing driven by a properly chosen criterion to determine the quality of the current structure.

The representation used by UNIMEM as seen in Figures 1.20 and 1.21 is self-explanatory. Each node (representing a concept formed by the system) is defined by a set of features such as `size(big)` (in the picture, the literal is reduced to the attribute value). Each feature is accompanied by an integer called `score` which tells the learner how many times the feature has so far been encountered. Note that the score reflects also examples that have been placed in other clusters—see, for instance, the score of the '2-tails' feature in the right-hand category. The score determines the strength of the feature. Small score indicates that the feature is rather irrelevant and

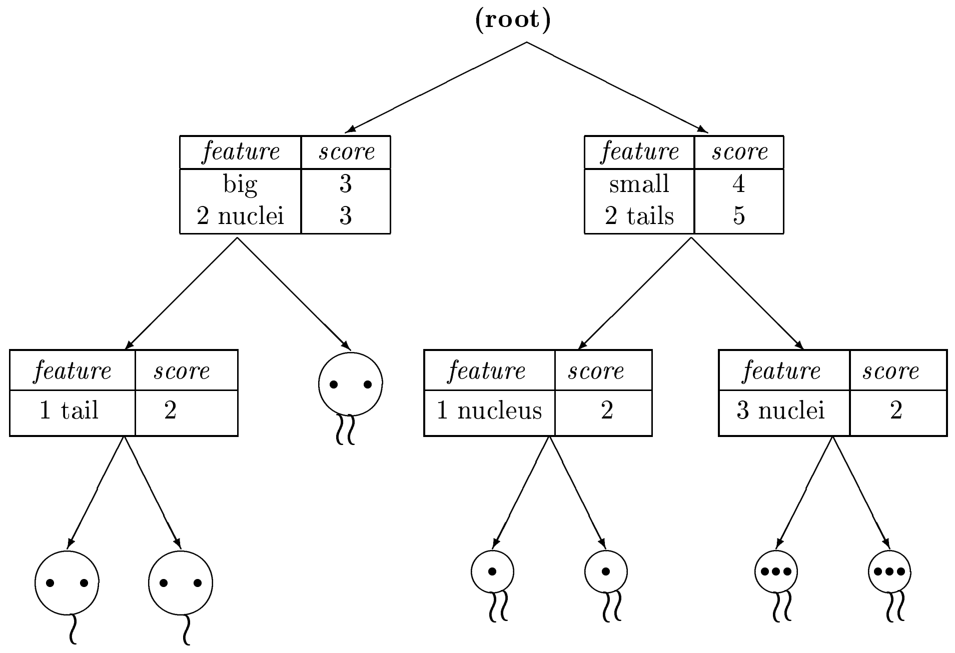


Figure 1.21 Absorbing a new example in the structure from the previous figure

should perhaps be discarded. Conversely, high score suggests that the feature should be 'fixed' in the structure and no longer threatened by deletion.

In principle, the following search operators underly the concept-formation process in UNIMEM:

1. Store a new instance in the closest node;
2. Create a new node if it improves the value of some general criterion assessing the quality of the created conceptual structure;
3. Fix a feature if its score exceeds a predefined threshold;
4. Delete a feature if its score is lower than the scores of the other features;
5. Delete an overly general node (containing only few features).

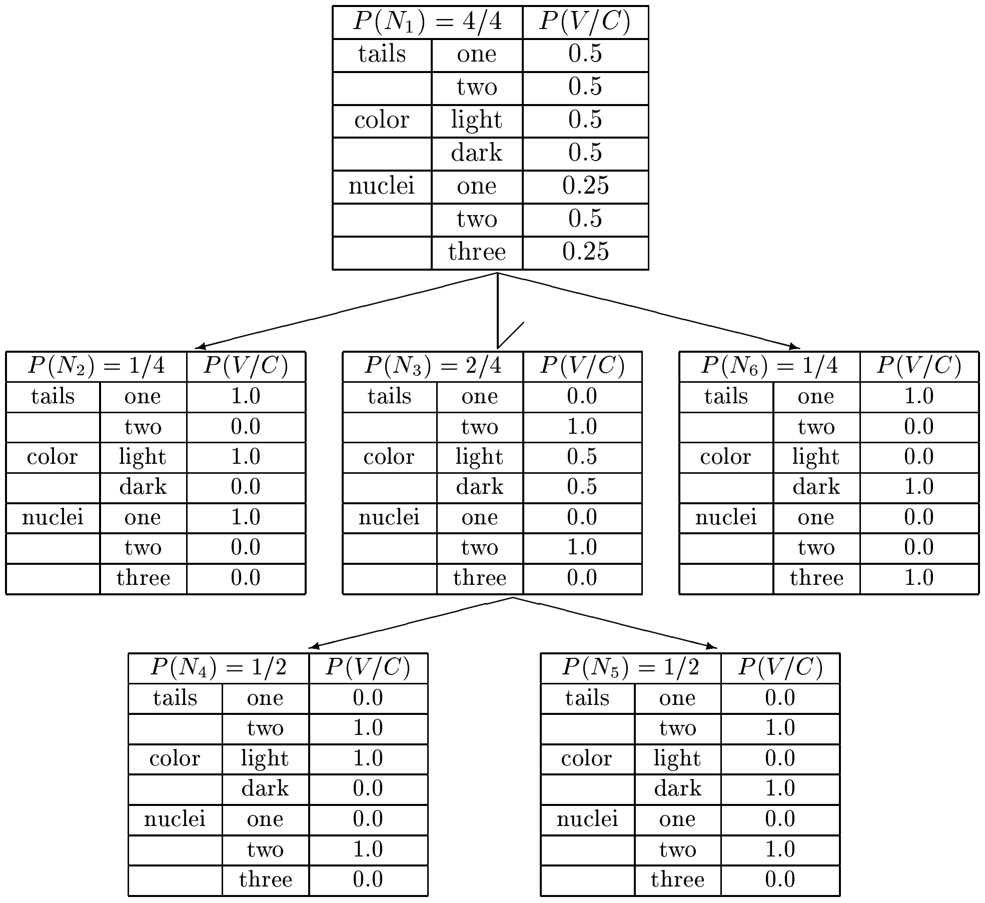
For more detailed information about the procedure carried out by the system UNIMEMsee Lebowitz (1987).

Probabilistic Conceptual Hierarchies

Other concept-formation systems differ from UNIMEM in the internal representational structure, in the description language (e.g. symbolic versus numeric attributes), in the search operators, and in the evaluation function guiding the search.

Thus in the system COBWEB, each node in the hierarchy contains a complete infor-

objects:
1 tail, light color, 1 nucleus
2 tails, light color, 2 nuclei
2 tails, dark color, 2 nuclei
1 tail, dark color, 3 nuclei



$P(N_1) = 4/4$		$P(V/C)$
tails	one	0.5
	two	0.5
color	light	0.5
	dark	0.5
nuclei	one	0.25
	two	0.5
	three	0.25

$P(N_2) = 1/4$		$P(V/C)$
tails	one	1.0
	two	0.0
color	light	1.0
	dark	0.0
nuclei	one	1.0
	two	0.0
	three	0.0

$P(N_3) = 2/4$		$P(V/C)$
tails	one	0.0
	two	1.0
color	light	0.5
	dark	0.5
nuclei	one	0.0
	two	1.0
	three	0.0

$P(N_6) = 1/4$		$P(V/C)$
tails	one	1.0
	two	0.0
color	light	0.0
	dark	1.0
nuclei	one	0.0
	two	0.0
	three	1.0

$P(N_4) = 1/2$		$P(V/C)$
tails	one	0.0
	two	1.0
color	light	1.0
	dark	0.0
nuclei	one	0.0
	two	1.0
	three	0.0

$P(N_5) = 1/2$		$P(V/C)$
tails	one	0.0
	two	1.0
color	light	0.0
	dark	1.0
nuclei	one	0.0
	two	1.0
	three	0.0

Figure 1.22 Representational structure of *COBWEB*

mation about the probability of the individual attribute values as shown in Figure 1.22 where the probabilities are estimated simply as relative frequencies.

What is peculiar about this representation is that the system does not store crisp descriptions. Rather, each attribute-value pair is accompanied by a number giving the probability that an instance of the concept will possess this particular attribute value. Each node in Figure 1.22 consists of a heading and a 3-column table. The heading contains information about the frequency, $P(N_i)$, with which an example falls into this category. The table contains the relative frequency of the occurence of any attribute-value pair.

COBWEB uses the following search operators:

1. *Incorporate* the new example into some of the existing nodes;
2. *Create* a new node for the example;
3. *Merge* two nodes into one;
4. *Split* a node into two nodes.

Whenever a new example is encountered, the learner must decide which of the operators applies best. Knowing that each operator can change the conceptual hierarchy, the system uses the formula assessing the *utility* of each of the potential new hierarchies.

$$\frac{IG - UG}{N}$$

where UG (*Uninformed Guess*) is the expected number of attribute values that can be correctly guessed from an unordered set of objects; IG (*Informed Guess*) is the expected number of attribute values that can be correctly guessed, given the conceptual hierarchy; and N is the number of categories that are currently present in the hierarchy.

More specifically, the following formula has been recommended (see Fisher, 1987, for a deteiled derivation):

$$\frac{\Sigma_{k=1}^{N} P(C_k)\Sigma_i\Sigma_j P(A_i = V_{ij} \mid C_k)^2 - \Sigma_i\Sigma_j P(A_i = V_{ij})^2}{N}$$

where $P(C_k)$ is the relative frequency of class C_k; $P(A_i = V_{ij})$ is the probability that attribute A_i will acquire the value V_{ij}; and $P(A_i = V_{ij} \mid C_k)$ is the corresponding conditional probability.

The point of this probabilistic approach is to create a conceptual hierarchy that maximizes the number of attribute values that can be predicted in an unseen example, given the information about the category into which the example falls.

1.6.2 Quest for Natural Laws

Many researchers claim that having powerful algorithms for concept formation at hand is not enough, that one should actually attempt to go one step further and try to build a system capable not only of constructing new concepts but also of describing their relations in terms of laws, as is the case in chemistry and physics.

Several reasons support activities in this field:

1. Nowadays, huge data bases from many scientific fields are available, waiting for someone to analyze them;

2. Powerful techniques in machine learning and artificial intelligence have been developed so that one can hope for a kind of "intelligent" analysis;
3. Even if intelligent automatic analyzers are not constructed, the research into artificial discovery may help to ellucidate some of the mysteries of human invention (e.g. inspiration, analogy, and abstraction).

Quantitative Empirical Laws

Suppose the task is to re-discover the ideal gas. The reader will recall from the high school that this law has the form $PV = 8.32NT$, where P is pressure, V is volume, N is gas amount, and T is temperature. A system capable of accomplishing this task has been proposed by Langley et al. (1987) and given the name BACON. Here, only a brief overview is possible, for more details see their paper.

BACON starts by suggesting a series of experiments that will provide the measurement data. The human operator carries them out and supplies the computer with the outcomes. As soon as enough data have been gathered, the system searches the space of mathematical functions with the objective to find an equation consistent with the data. One method of searching for the equation is to make one of the variables dependent while the others remain independent. Let the system have a repertoir of typical law-forms such as

$$y = ax^2 + bx + c$$
$$\sin(y) = ax + b$$
$$y^{-1} = ax + b$$

The principle consists in selecting the best law-form and tuning the parameters $a, b, \ldots$, with the objective to find an equation that best describes the observed data.

Suppose the equation $y^{-1} = ax + b$ has been selected. At the beginning, the parameters a and b are initialized to the values $1, 0$, and -1, so that the following combinations are considered as a set of initial states: $[a = 1, b = 1], [a = 1, b = 0], [a = 1, b = -1], [a = 0, b = 1], [a = 0; b = 0]$, etc.

In the search process, the parameters are tuned by adding or subtracting one parameter value at a time, starting with 0.5, then 0.25, 0.125, ... The evaluation function assessing the quality of each subsequent equation is defined by the correlation between the measured data and the values implied by the equation.

Suppose the values in Table 1.6 have been measured. BACON will investigate then in the following steps:

1. Find a function describing $V = f(P)$ for the triplets of examples assigned, in Table 1.6, to each of the three temperatures, $T = 10, T = 20$, and $T = 30$.

Suppose that $V^{-1} = aP + b$ with the folowing parameters provides the best fit:

$T = 10$: $a = 0.000425$, thus $V^{-1} = 0.000425P$
$T = 20$: $a = 0.000410$, thus $V^{-1} = 0.000410P$
$T = 30$: $a = 0.000396$, thus $V^{-1} = 0.000396P$

Table 1.6 Sample data for the system BACON

quantity	temperature	pressure	volume
N=1	T=10	P=1000	V=2.36
.	.	P=2000	V=1.18
.	.	P=3000	V=0.78
.	T=20	P=1000	V=2.44
.	.	P=2000	V=1.22
.	.	P=3000	V=0.81
.	T=30	P=1000	V=...
.	.	P=2000	V=...
.	.	P=3000	V=...
.			
N=2	⋮		
.			
.			
N=3	⋮		
.			

2. Since the parameter values evidently depend on the temperature T, the next task is to find the function relating a to T. Again, the best fit is achieved by the form $a^{-1} = cT + d$ with the values of the parameters, c and d, depending on N:

$N = 1$: $c = 8.32$ and $d = 2271.4$, thus $a^{-1} = 8.32T + 2271.4$
$N = 2$: $c = 16.64$ and $d = 4542.7$, thus $a^{-1} = 16.64T + 4542.7$
$N = 3$: $c = 24.96$ and $d = 6814.1$, thus $a^{-1} = 24.96T + 6814.1$

3. Find functions relating c to N and d to N. The best fit is achieved by $c = eN$ and $d = fN$, respectively, with $e = 8.32$ and $f = 2271.4$. These parameters do not depend on any other variable.

4. Substituting the equation into the equations found in the previous steps, the system obtains:

$$V^{-1} = (8.32NT + 2271.4N)^{-1}P$$

and this last expression can easily be transformed into:

$$PV = 8.32NT + 2271.4N$$

Factoring out 8.32N on the right-hand side, we arrive at:

$$PV = 8.32N(T + 273)$$

which, indeed, is the standard form of the ideal gas law. Note, that BACON has found that the Celsius temperature scale is improper. As a matter of fact, the system introduced the Kelvin scale, adding 273 to the observed Celsius value.

To conclude, the essence of BACON is to apply common search principles in the quest for an ideal *form* of a quantitative law, rather than just find the best fitting parameters as is the case of traditional regression techniques.

The qualitative counterpart of the previous quantitative discoverer is the system GLAUBER, which attempts to form qualitative chemical laws and concepts. GLAUBER turned out to be able to re-discover the concepts of acids and alkalis and to postulate some basic properties of these concepts. For more details, as well as for other interesting systems capable of automated discovery, see Langley et al. (1987). To conclude this subsection, let us briefly mention a slightly more advanced variation on the principles just outlined.

1.6.3 Discovery in Dynamic Systems

LAGRANGE is a program for discovering numerical laws in the submitted data, similarly as was the case in BACON. However, LAGRANGE differs in that it generates models from data measured on *dynamic* systems. LAGRANGE's models have the form of differential equations. As opposed to traditional system identification techniques used in control engineering, LAGRANGE finds the *structure* of the equations, not only the values of the parameters.

As an illustration consider an application from the domain of ecological modelling. Two variables, x and c, are assumed. x is the concentration of bacteria in a test-tube, and c is the concentration of nutrition for bacteria. The task for LAGRANGE is: given the tabular representation of the two curves in time $x(t)$ and $c(t)$, find a differential equation whose numerical solution corresponds to the two given behaviors. For the case of this particular biological domain—reported by Džeroski and Todorovski (1994)—LAGRANGE found the following differential equations:

$$\dot{c} = -\frac{1}{60}x - \frac{10}{6}\dot{x}$$

$$c\dot{x} = -x - 100\dot{x} + 0.09cx$$

For the concrete values of the system's parameters, such as the growth rate, this corresponds to the Monod model, well known from ecological modelling.

In general, the discovery problem for LAGRANGE is stated as follows:

Given:

Trace in time of a dynamic system:

$\vec{x}(t_0)$, $\vec{x}(t_0 + h)$, ...

Parameters:

o = order of differential equations
d = maximum depth of newly generated terms
r = maximum number of "independent regression variables"
t_R = significance threshold

Find:

Differential equations within the parameters (o, d, r) that match the data within significance threshold t_R.

As reported by Džeroski and Todorovski (1994) LAGRANGE successfully discovered (again, it should be admitted: *re*-discovered) differential equations for: a chemical reaction with three chemical substances, modelling the predator-prey chain, the so-called Brusselator chemical reactor, the pole-cart system etc.

1.7 How to Cope with the Vastness of the Search Space?

One of the principal problems of machine learning is that the space of all possible descriptions is often so large that the search either has to rely on heuristics, or becomes computationally intractable. Also the danger of converging to local maxima of evaluation functions is in large spaces more serious.

Two techniques to attack this problem deserve special section: the use of analogy and the idea of storing the original examples instead of their generalized descriptions.

1.7.1 Analogy Providing Search Heuristics

The principle of analogy has been extensively studied in the artificial-intelligence community because of the wide-spread belief that the ability to find proper analogies is one of the secrets of intelligence. Much work has been devoted to analogy-based reasoning.

What is the essence of this mechanism as viewed from the machine-learning perspective? Kodratoff (1988) coined the scheme depicted in Figure 1.23 as the general framework of analogy. Here, S stands for source, SC for source concept, T for target, and TC for target concept. The task is to derive the target concept from T in a way that is analogous to the way source concept was derived from the source. Thus having the target, the learner must find proper source.

Greiner (1988) suggests the following general procedure for any reasoning by analogy:

Reasoning-by-Analogy Algorithm

1. *Recognition.* Given a target concept, find in the background theory a source S that is 'similar' to T. The similarity can be measured by syntactic distance, by the existence of common generalization or of a pair of unifying substitutions, or by some hint supplied by the user;

$$\begin{tikzcd}
T \arrow[r] & TC \\
S \arrow[r] \arrow[u] & SC \arrow[u]
\end{tikzcd}$$

Figure 1.23 General scheme of analogy

2. *Elaboration.* Find SC, together with the inference chain $\vdash_S$ leading to it from S. Note that, for each S, a collection of SC's usually exist;
3. *Evaluation.* Among the SC's, find the one that best satisfies given criteria;
4. Apply to T an inference chain $\vdash_T$ 'similar' to $\vdash_S$, thus obtaining TC. Assess the utility of TC;
5. If necessary, repeat iteratively steps 1–4 to find S, SC, $\vdash_S$, and $\vdash_T$ that yield the most promising (useful) TC;
6. *Consolidation.* Include TC together with the inference chain $\vdash_T$ into the background theory.

Since the above framework is somewhat too general, reasonable constraints are usually needed. Thus the source S can be explicitly supplied by the user telling the system that if the task is to calculate the flowrate through a pipelining structure, then the laws analogical to those used in electrical engineering (Kirchhoff's laws) should be used. Another possibility is that the user takes over the evaluation process and selects proper SC for the source that has been suggested by the system. Greiner (1988) describes a system that was capable of learning to solve fluid flow problems, using as analogy prior knowledge about electrical circuits.

1.7.2 Instance-Based Learning

An explicit concept description is not always explicitly required. If the only reason for learning is the need to identify future examples, then the learner can adopt an alternative policy: instead of descriptions, store typical examples. This can preclude many troubles potentially entailed by the search through a prohibitively large space of generalizations. Note that a similar idea has already been adopted by some of the concept-formation systems treated earlier.

This section outlines the principle of the system IBL (Aha, Kibler, and Albert, 1991) which is able to store selected examples (described by attribute values) and use them according to the so-called *nearest-neighbor* principle: the newly arrived example is assigned the class of the closest one among the stored examples.

A simple formula to calculate the similarity between the examples x and y is used (x_i and y_i are the respective values of the i-th attribute):

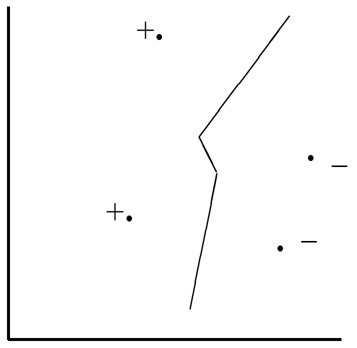


Figure 1.24 $\oplus$ and $\ominus$ examples defining the positive and negative space

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

where the function f is calculated for numeric attributes by:

$$f(x_i,y_i) = (x_i - y_i)^2$$

and for symbolic and Boolean attributes by:

$$f(x_i,y_i) = \begin{cases} \mathbf{1} & x_i \neq y_i \\ \mathbf{0} & x_i = y_i \end{cases}$$

The principle is illustrated in Figure 1.24 where four examples described by two numeric variables are depicted, together with the discrimination function separating the space of positive examples from the space of negative examples.

The learning assumes the availability of a feedback that will immediately inform the learner about the success or failure of each single classification attempt. A very simplified version of the IBL-algorithm involves the following steps:

Instance-Based-Learning Algorithm.

1. Define the *set of representatives* containing, at the beginning, the first example;
2. Read a new example x;
3. For each y in the set of representatives, determine *similarity(x,y)*;
4. Label x with the class of the closest example in the set of representatives;
5. Find out from the feedback whether the classification was correct;
6. Include x in the set of representatives and go to 2.

Two shortcomings degrade the utility of this elementary version: excessive storage requirements caused by the fact that *all* examples are stored; and sensitivity to noise.

The rectification consists in a selective storage of the examples by a 'wait-and-see' strategy whose essence can be summarized by the following principles:

1. Whenever a new instance has been classified, the 'significance-score' of each of the previous instances is updated (see below) and the instance is stored;
2. Instances with *good* scores are used for the classifications; instances with *bad* scores are deleted;
3. *Mediocre* instances are retained as potential candidates. However, they are not used for classification.

In the classification phase, the new arrival is assigned the class of the nearest *good* instance if a *good* instance exists. Otherwise, the new arrival is assigned the class of the nearest *mediocre* instance.

Then, the system increments the scores of those *mediocres* that are closer to the new arrival than the closest *good* instance. If no *good* instance is available, the system updates *mediocres* inside a randomly chosen hypersphere arround the new arrival.

A score is considered as *good* whenever the classification accuracy achieved by this instance is higher than the frequency of the example's class. *Classification accuracy* of class $\oplus$ is the percentage of correctly recognized positive examples in the set of *all* examples.

Instance-based learning has been reported to achieve a significant recognition power in attribute-value domains, especially when the number of examples is large and the attributes describing them are properly chosen. Also the robustness against noise is satisfactory. On the other hand, the power of the system degrades if the descriptions of the examples contain irrelevant attributes and/or if the number of examples available to the learning procedure is small.

1.8 Close Neighborhood of Machine Learning

The general label of machine learning is usually reserved to artificial-intelligence-related techniques, especially to those whose objective is to induce symbolic descriptions that are *meaningful* and *understandable* and at the same time help improve performance. In a broader understanding, though, the machine-learning task can be defined as any computational procedure leading to an increased knowledge or improved performance of some process or skill such as object recognition.

Particularly the learn-to-recognize task is often addressed by methods that are traditionally not strictly included in machine-learning paradigms but have the same or similar objective. Thus the statistical data analysis (see Everit, 1981) and traditional pattern recognition (see Duda and Hart, 1973) spawned many useful techniques. Even though a detailed discussion of the many alternative approaches would prohibitively extend the scope of this chapter, two techniques must be briefly mentioned because of their popularity and because of the many attempts to combine them with machine-learning algorithms: neural networks and genetic algorithms.

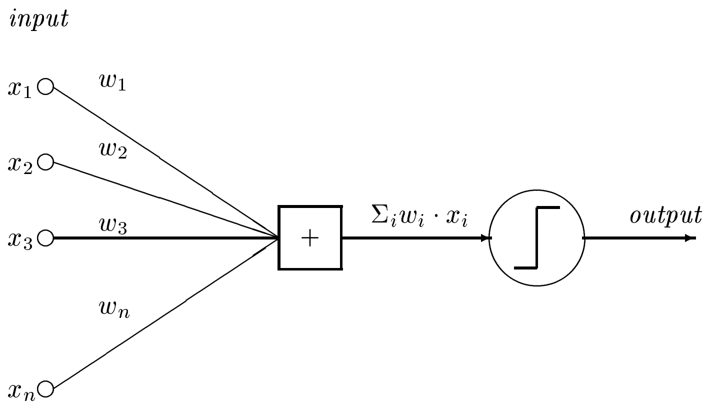


Figure 1.25 General scheme of a perceptron

1.8.1 Artificial Neural Networks

In the late fifties, Mark Rosenblatt suggested to use, for pattern-recognition purposes, a simple device, inspired by early mathematical models of biological neurons. In his famous paper (Rosenblatt, 1958) and book (Rosenblatt, 1962), he dubbed this device *perceptron* and showed how it can be trained for the recognition job simply by automatic adjustments of its parameters, based on a set of preclassified examples. The principle is shown in Figure 1.25. Several input signals, x_i, each multiplied by a weight w_i, are attached to a summation unit. The resulting *sum* $= \Sigma_i w_i \cdot x_i$ is subjected to a step function ensuring that if the sum exceeds a certain threshold θ, the output of the peceptron is 1, otherwise the output is 0. As an alternative to the values 1 and 0, any other pair of outputs can be considered, say 1 and -1.

Proper adjustments of the weights w_i and of the threshold θ ensure that the perceptron will react to input vectors with the required output value. The information is thus encoded in the weights assigned to each individual input, each input representing an attribute. More relevant attributes are assigned more weight and less relevant attributes have less weight. Perceptron's learning algorithm seeks such weight values that will accomplish the requested mapping from the space of input vectors to the set of two binary values, $R^n \rightarrow \{0, 1\}$.

Unfortunatelly, some concepts cannot be acquired by perceptron, among them, for instance, *exclusive* OR, as has been shown by Minsky and Papert (1969). That is why perceptrons are only rarely used in isolation. Rather, they are interconnected in structures such as the *Multilayer Perceptron*, depicted in Figure 1.26 (for an analysis of multilayer perceptrons, see Rumelhart, Hinton, and Williams, 1986).

In principle, multilayer perceptron consists of one layer of input nodes, one layer of output nodes, and one or more 'hidden' layers between them. During the recognition phase, the components of the input vector are clamped to the input layer. Obviously, some of the perceptrons 'fire' (their output is 1), when the weighted sum of their

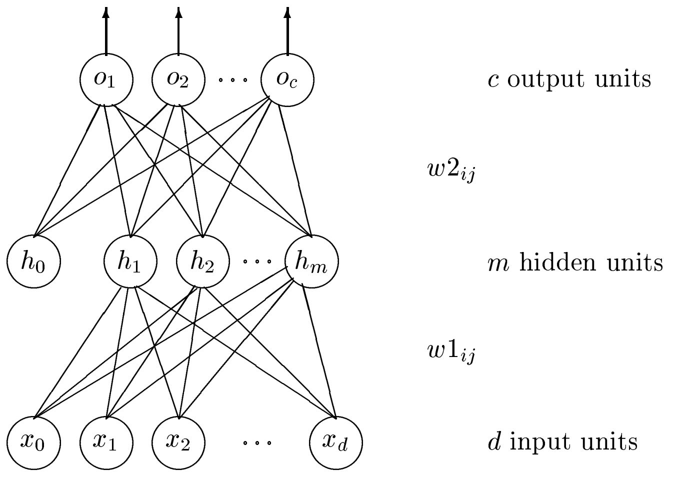


Figure 1.26 Multilayer perceptron

inputs exceeds the particular threshold. The value 1 or 0 is then propagated to the next layer, until the output of the network is reached.

As the basic threshold function is too rigid (it does not tolerate noise and does not facilitate learning), usually the *sigmoid function* is used to calculate the output of a single unit from its inputs:

$$f(sum) = \frac{1}{1 + e^{-sum}}$$

where *sum* is the weighted sum of the signals at the unit input. According to this formula the unit will output a real value between 0 and 1. For $sum = 0$, the output is 0.5; for large negative values of *sum* the output converges to 0; and for large positive values of *sum* the output converges to 1. The formula is more tolerant than the step function with respect to noisy signals.

Usually, only a single hidden layer is employed, as is the case of Figure 1.26. However, in many complicated tasks researchers made good experience when they used two or more hidden layers.

The procedure for the automatic adjustment of the weights is provided below without any further discussion. The interested reader is referred to some of the many monographs on neural networks. Among the many existing textbooks of neural networks, perhaps Beale and Jackson (1990) can be recommended as an easy-to-read introduction. For more comprehensive treatment, see, for instance, Haykin (1994).

Backpropagation Learning Algorithm

1. Define the configuration of the neural net in terms of the number of units in each layer;
2. Set the initial weights $w1_{ij}$ and $w2_{ij}$ to small random values, say, from the interval $[-0.1, 0.1]$;
3. Select an example and denote its attribute values by $x_1, \ldots, x_k$. Attach the example to the input layer;
4. *Propagate* the input values from the input layer to the hidden layer. The output value of the j-th unit is calculated by the function $h_j = \frac{1}{1+e^{-\sum_i w1_{ij} \cdot x_i}}$.
   Propagate the values thus obtained to the output layer. The output value of the j-th unit in this layer is caluculated by the function: $o_j = \frac{1}{1+e^{-\sum_i w2_{ij} \cdot h_i}}$;
5. Compare the outputs o_j with the teacher's classifications y_j calculate the correction error as $\delta 2_j = o_j(1-o_j)(y_j-o_j)$ and adjust the weights $w2_{ij}$ by the folowing formula:
   $$w2_{ij}(t+1) = w2_{ij}(t) + \delta 2_j \cdot h_i \cdot \eta$$
   where $w2_{ij}(t)$ are the respective weight values at time t and η is a constant such that $\eta \in (0, 1)$;
6. Calculate the correction error for the hidden layer by means of the formula $\delta 1_j = h_j(1-h_j) \sum_i \delta 2_i \cdot w2_{ij}$ and adjust the weights $w1_{ij}$ by:
   $$w1_{ij}(t+1) = w1_{ij}(t) + \delta 1_j \cdot x_i \cdot \eta$$
7. Go to step 3.

The above algorithm captures only the fundamental principle of learning in multilayer perceptrons and its practical use in many realistic applications suffers from various shortcoming and pitfalls that the user must be acquainted with. However, these caveats have been studied in great detail and, nowadays, neural networks represent a well-established scientific discipline.

1.8.2 Genetic Algorithms

The reader has seen that the learning procedure is in many cases conceived as a search through the space of representations permitted by the given language. This subsection presents a surprisingly powerful alternative to the traditional heuristic search techniques: the *genetic algorithm* that has been inspired by a similar principle in nature.

Generally speaking, the evolution in nature is controled by three fundamental principles:

1. *Survival of the fittest* means that the strongest specimens have the highest chance to survive and to reproduce, whereas the weak ones are likely to die before they reach the reproduction stage;

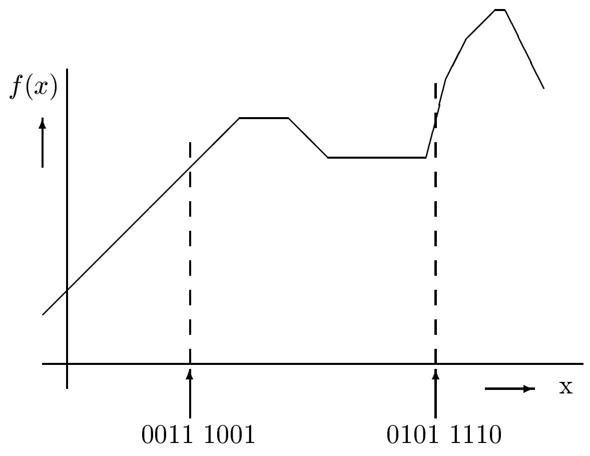


Figure 1.27 Evaluation function $f(x)$ and two binary specimens

2. In the *sexual reproduction* the specimens find partners they consider as best ones, thus further contributing to the survival-of-the-fittest principle. Then they recombine their genetic information, thus creating new speciments with somewhat different characteristics;
3. *Mutations* cause random, and relatively rare, changes in the genetic information.

The unquestionable success that this "search technique" has in nature inspired some researchers to investigate methods to turn it into algorithms that can be enbcoded in computer programs. A lucid introduction into the discipline of genetic algorithms has been written by Goldberg (1989).

In this chapter we present only the basic principles that are necessary for the implementation of a working version of this mechanism. At the beginning of any successful attempt to cast a technological problem in a setting that facilitates its solution by means of a genetic algorithm, two questions must be answered. How to encode the search space in chromosomes; and how to define the *fitness function* (see Figure 1.27) that plays the role of evaluation function in heuristic search. In most implementations, the chromosomes are represented by bit strings. Each bit can stand for a binary attribute, the presence of a multivalued attribute, the presence of a predicate, etc. The fitness function, measuring the survival chance of the specimen, can be defined as the accuracy of the description derived from the chromosome, the entropy of the partitioning imposed by this description (examples satisfying the description versus example that do not satisfy it), and the like.

The principle of the genetic algorithm is illustrated by the example in Table 1.7. Here, the fitness function is defined as $f(x) = 1/(x + 1)$, where x is the number represented in the chromosome in binary form (e.g. '111' = 7). Obviously, the maximum value of $f(x)$ will be reached for the string '000000.'

The table shows one step of the algorithm. The old generation contains four numbers: 37, 11, 20, and 7. The maximum value of the fitness function is reached for

Table 1.7 One step in the genetic search for the maximum of the function $1/(x+1)$ (no mutation)

	old gener.	x	$1/(x+1)$		*survivors*	*new gener.*	x	$1/(x+1)$
(1)	1 0 0 1 0 1	37	0.026	(4)	0 0 0 | 1 1 1	0 0 0 0 1 1	3	0.250
(2)	0 0 1 0 1 1	11	0.083	(2)	0 0 1 | 0 1 1	0 0 1 1 1 1	15	0.063
(3)	0 1 0 1 0 0	20	0.048	(4)	0 0 0 1 | 1 1	0 0 0 1 0 0	4	0.200
(4)	0 0 0 1 1 1	7	0.125	(3)	0 1 0 1 | 0 0	0 1 0 1 1 1	23	0.042

$f(7) = 1/(7+1) = 0.125\%$, so the chromosome representing $x = 7$ has the highest chance of survival. Conversely, the number $x = 37$ has the smallest fitness function, $f(37) = 1/(37+1) = 0.027$, and consequently, has a negligent chance to survive. This chance is given by a random number generator ensuring that the strongest specimens can be replicated more than once in the space of survivors (here, the 'technical' genetic algorithm somewhat departs from the 'natural' one) while the weakest specimens die out. This step is called *reproduction*.

In the next step, each survivor choses a mating partner and exchanges with it part of their genetic information. This step is called *recombination* and is modeled as the exchange of random substrings. For simplicity, the chromosomes in Table 1.7 exchange only tails of random length. After this step, a new generation of stronger specimens comes into being. Indeed, the values of the fitness function indicate that its maximum as well as the average value increased.

The *mutation* operator (not applied in Table 1.7) is modeled quite straightforwardly: with a very small likelihood, a bit is flip-flopped to its opposite value. The likelihood constant is usually adjusted so that in one generation no more than just a few (say, 0 through 5) mutations appear.

GA-Algorithm

1. Define the initial population as a set of binary strings generated randomly or by some pre-specified mechanism;
2. Replicate the specimens in the population into the set of survivors by a mechanism that ensures that specimens with high value of fitness function have higher chance of survival (and can be replicated more than once);
3. For each survivor, find a mate with which it exchanges part of the information encoded in the binary strings. With a very low frequency, a single bit is flip-flopped to model random mutations;
4. If the fitness function has not increased throughout several cycles, stop. Otherwise go to step 2.

The interested reader is referred to the monograph by Goldberg (1989) where a detailed analysis with extensive bibliography can be found.

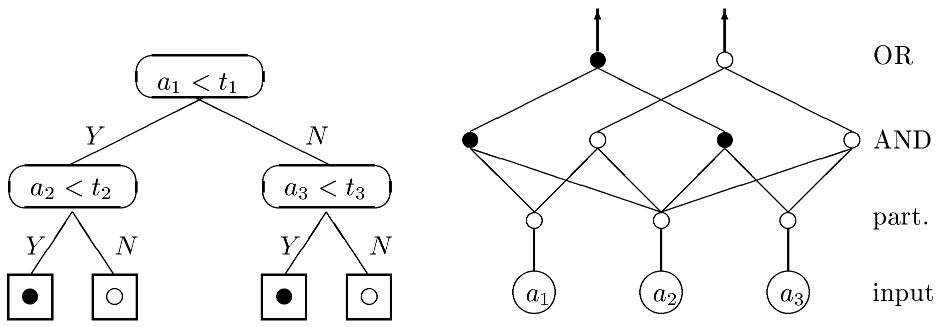


Figure 1.28 Functional decision tree and the corresponding entropy net

1.9 Hybrid Systems and Multistrategy Learning

The real world often poses problems that cannot be successfully tackled by one of the basic techniques described above. Each of these techniques has its assets and liabilities. For instance, TDIDT has been designed for attribue-valued data and is much less valuable when more sophisticated description languages together with substantial background knowledge are required. Likewise, systems based on predicate logic are good at dealing with Horn clauses but pay the price of high computational demands; neural nets are excellent at pattern recognition but suffer from their sensitivity to the initial topology and weights, as well as proper selection of attributes; and genetic algorithms, albeit surprisingly powerful, require smart encoding into chromosomes and can be very slow learners.

It is only natural that machine-learning researchers experiment with combinations of the individual approaches to bridge some of the chronical pitfalls. Research on buidling systems that combine different strategies or methods is at its very early stage and falls into a new subarea of machine learning, called multistrategy learning (Michalski and Tecuci, 1994; Wnek et al, 1995

Entropy Networks

As already mentioned, the performance of neural networks tends to degenerate whenever the input vector contains irrelevant features. Conversely, TDIDT-related systems, though good at pruning out noise and useless attributes, tend to build too rigid descriptions based on the strict ordering of attributes. These complementary deficiencies inspired successful attempts to merge the two approaches. An impressive simplicity and convincing results characterize the idea of *entropy nets* that was first introduced by Sethi (1990). The system was designed primarily for learning in domains where examples are described by numeric attributes.

The procedure for the generation of entropy nets consists of three steps: tree growing, translation of the tree into a neural net (which is then called entropy net), and training the entropy net.

For the decision-tree growing, the procedure described earlier in this chapter can be used. The fact that proper attributes are selected by a measure based on entropy has given the system its name.

The mapping of the decision tree to a neural network is facilitated by the observation that conjunctions and disjunctions of Boolean attributes are easy to implement by simple models of neurons: Suppose that all weights are equal to 1. Then setting the neuron's threshold value to $n - 0.5$ ensures that the neuron can be activated only if all inputs are 1. In this case, the weighted sum of the inputs is $\Sigma w_i a_i = n$, which exceeds the threshold value. Similarly, setting the neuron's threshold to 0.5 makes it to carry out disjunction of the inputs.

Figure 1.28 illustrates the mapping. The bottom layer of the network contains simply the inputs. Each of the units in the first hidden layer (called *partitioning layer*) carries out one of the decision tests (such as $a_1 < t_1$) at the internal nodes of the tree. Each leave of the decision tree is mapped to a corresponding unit in the second hidden layer, called AND-layer. These units perform the conjunction of the tests along the tree branch. Finally, each unit of the output layer (OR-layer) stands for one classification value and models the disjunction of the leaves with the same class label.

The subsequent training of the net uses the backpropagation algorithm that has been described in a previous section. The idea is to further increase the classification accuracy of the system as compared to the original decision tree. The tradeoff is that the interpretability of the encoded knowledge vanishes.

Knowledge-Based Neural Nets

Another shortcoming of neural nets is their negligence of background knowledge and complications with the search for the ideal topology. This is why Towell, Shavlik, and Noordewier (1990) experimented with their system KBANN that is able to learn in logic and then tune the acquired knowledge by way of a neural-network training.

Suppose that the background knowledge contains the following rules that, taken together, define some concept `a`:

```
a :- b, c.
b :- g, not(f).
b :- not(h).
c :- i, j.
```

`a` is defined as the conjunction of *intermediate concepts* `b` and `c`. These, in turn depend on the *supporting facts* `g, f, h, i`, and `j`. Supporting facts are those features that can be dirrectly measured on the objects serving as examples. Intermediate concepts are defined by the supporting facts and, potentially, by other intermediate concepts.

Two steps characterize the system. First, the knowledge is translated into the net-

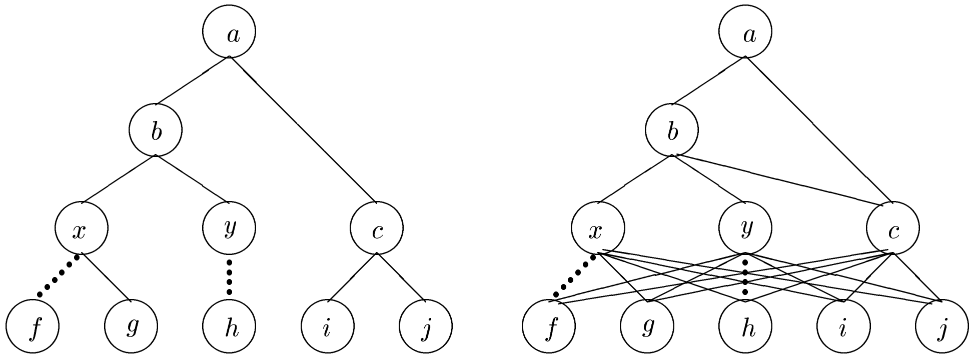


Figure 1.29 Translation of knowledge into a neural net

work where the supporting facts are modeled by input units, intermediate concepts by hidden units, and the final concepts by output units. The dependencies between units in different layers are represented by weights. At this stage, each of the weights has the same absolute value. In the second step, the net is enlarged to give a chance also to those predicates and facts that have not explicitly appeared in the background knowledge. Then, the weights are slightly perturbed by random numbers and the net is trained by the backpropagation algorithm.

Figure 1.29 illustrates the principle. The rules are translated into the rough topology in the left-hand part of the picture where the dotted lines represent links with negative weights (e.g. in the rule `b :- not(h)`). This topology is then refined by introducing supplementary low-weighted connections shown in the right-hand part of the picture. For more details see Towell, Shavlik, and Noordewier (1990). An approach to initialize neural networks with an AQ-based algorithm is studied by Bala, Michalski, and Pachowicz (1994).

We have already mentioned that as the network is a black-box system, an unpleasant consequence of the refinement of the knowledge by a neural-net training is that the user loses the interpretation of the results. To attack this problem, Towell, Craven, and Shavlik (1991) suggested a method to extract knowledge in the form of production rules from a trained neural network.

Genetic Search for Generalizations in AQ

One of the vulnerable aspects of the AQ-algorithm is the search for the optimal generalization of seeds. The reason is that the number of all possible generalizations can be so high that the computational tractability of the whole program might become an issue.

This problem motivated Venturini (1993) to develop the system SIA where the search for the ideal seed generalization is carried out by a mechanism inspired by the genetic algorithm. Each chromosome represents one production rule. However, the reproduction scheme as well as mutation are the same as described above. The recombination uses the crossover operator only in a relatively small proportion of the specimens, just to supplement the traditional generalization. The population size is variable.

To find the ideal generalization, SIA starts with a population containing only the most specific description of the seed (the initial size of the population is, therefore, $N = 1$). In each of the subsequent generations, one of the following operators is randomly chosen and applied with the probability indicated in the parentheses:

1. *Create* a new rule (probability of 10%);
2. Select an arbitrary rule and *generalize* it (probability of 80%);
3. Perform the traditional *crossover* of two rules by exchanging some conjuncts between them (probability of 10%).

For a more detailed explanation, see the original paper by Venturini (1993).

Combination of GA and Neural Nets

Finally, several researchers investigated the possibilities of the use of genetic algorithms to find the architecture and/or weights of neural networks. The work by Bornholdt and Graudenz (1992) can serve as an illustration of these efforts. Here, the genetic algorithm searches for the ideal topology of the network. The individul positions of the chromosome represent neurons and each of them contains pointers to other neurons, so the chromosome is more complicated than a simple bit string. The fitness function measures the quality of a given network.

However, more detailed discussion of these efforts would depart from the main stream of the learning algorithms described in this chapter.

1.10 Perspectives

As shown above, the field of machine learning has developed a great variety of approaches and techniques. A brief historical review of the evolution of many of them can be found in (Cohen and Feigenbaum, 1982) and (Michalski, Carbonell, and Mitchell, 1983).

The methods presented here fall into the general category of inductive concept learning, which constitutes perhaps the most advanced task in machine learning. The underlying assumption for most of these methods is that the learner induces a concept description from given concept instances. Such a process is inherently inductive, and the correctness of the created descriptions cannot be guaranteed. Therefore, the descriptions created by these techniques always have to be tested on new data.

Since these descriptions represent generalizations of given facts and can be incorrect, in many applications it is crucial that they be interpreted and understood by a human expert before they can be used. Therefore, we have pointed to the importance of the comprehensibility condition in concept learning.

The descriptions can be expressed in different forms, such as decision trees, decision rules, neural nets, Horn clauses, grammars, etc. Each representation requires a somewhat different method of information processing, and has its own advantages and disadvantages. To apply any of them to a given problem requires an analysis of the problem at hand and a decision which representation and learning strategy would be most appropriate.

For completness, it should be mentioned in conclusion, that there have been several other general approaches developed in the field, that are not covered in this chapter. They include:

1. Explanation-based learning, a methodology that deductively derives operational knowledge from a concept example and some apriori known abstract concept description—see, for instance DeJong and Mooney (1986) or Mitchell, Keller, and Kedar-Cabelli (1986);
2. Case-based learning, a learning method in which concept examples are stored, and new cases are recognized by determining the class of the closest past case (or cases)—see, for instance, Bareiss, Porter, and Wier (1987) or Rissland and Ashley (1989);
3. Reinforcement learning, in which numerical feedback about the performance at a given step is used to modify the parameters of the learning system—see, for instance, Sutton (1988).

Machine Learning is a relatively young discipline and it is likely that many new, more powerful methods will be developed in the future. The following chapters of this book demonstrate, however, that already the existing techniques can successfully be applied to many practical problems.

References

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

Bala J.W., Michalski, R.S., and Pachowicz, P.W. (1994). Progress on Vision through Learning at George Mason University. *Proceedings of ARPA Image Understadning Workshop* 191-207

Beale, R. and Jackson, T. (1990). *Neural Computing: An Introduction.* Adam Hilger, Bristol

Bareiss, E.R., Porter, B. and Wier, C.C. (1987). PROTOS: An Exemplar-Based Learning Apprentice. *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA, Morgan Kaufmann, 12–23

Bergadano, F., Matwin, S., Michalski, R.S., and Zhang, J. (1992). Learning Two-Tiered Descriptions of Flexible Concepts: The POSEIDON System. *Machine Learning*, 8, 5–43

Bornholdt, S. and Graudenz, D. (1992). General Assymmetric Neural Networks and Structure Design by Genetic Algorithms. *Neural Networks*, 5:327–334

Bratko, I. (1990). *PROLOG Programming for Artificial Intelligence*, Addison-Wesley Publishing Company (Second Edition)

Breiman, L., Friedman, J., Olshen, R. and Stone, C.J. (1984). *Classification and Regression Trees.* Belmont, California, Wadsworth Int. Group

Cestnik, B. (1990) Estimating probabilities: a crucial task in Machine Learning. *Proc. ECAO 90*, Stockholm, August 1990.

Cestnik, B. and Bratko, I. (1991) On estimating probability in decision tree pruning. *Proc. EWSL-91*, Porto, Portugal, March 1991. Springer-Verlag.

Cestnik, B. and Karalič, A. (1991). The Estimation of Probabilities in Attribute Selection Measures for Decision Tree Induction. *Proceedings of the Information Technologies Interface, ITI-91*, Cavtat, Croatia, June.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*, Addison-Wesley Publishing Company.

Cohen, P.R. and Feigenbaum, E. (eds.) (1992). The Handbook of Artificial Intelligence, vol. III, sec. XIV (written by T. Dietterich), 323-494.

DeJong, G.F. and Mooney, R.J. (1986). Explanation-Based Learning: An Alternative View. *Machine Learning*, 1:145–176

de Raedt, L. (1992). Interactive Concept-Learning and Constructive Induction by Analogy. *Machine Learning* 8:107–150

Duda, R.O. and Hart, P.E. (1973). *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York

Džeroski, S. and Todorovski, L. (1994) Discovering dynamics. *J. Intelligent Information Systems*, 1994.

Esposito, F., Malerba, D., and Semeraro, D. (1993) Decision tree pruning as a search in the state space. *Machine Learning: ECML-93)* (ed. P. Brazdil), Proc. European Conf. Machine Learning, Vienna, April 1993.

Everitt, B. (1981). *Cluster Analysis.* Heinemann, London

Fayyad, U.M and Irani, K.B. (1992). On the Handling of Continuous-Valued Attributes in Decision Tree Generation. *Machine Learning* 8:87–102

Fisher, D.H. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning* 2:139–172

Fisher, D.H., Pazzani, M.J., and Langleym P. (eds.) (1991). *Concept Formation: Knowledge and Experience in Unsupervised Learning.* Morgan Kaufmann, San Mateo

Gennari, J. Langley, P. and Fisher, D. (1989). Models of Incremental Concept Formation. *Artificial Intelligence* 40:11–62

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading

Greiner, R. (1988). Learning by Understanding Analogies. *Artificial Intelligence* 35:81–125

Haykin, S. (1994). *Neural Networks, A Comprehensive Foundation.* Maxmillan College Publishing Company, New York

Hirsh, H.(1990). *Incremental Version-Space Merging: A General Framework for Concept Learning.* Kluwer Academic Publishers

Klimesch, W. (1988). *Struktur und Aktivierung des Gedächtnisses. Das Vernetzungsmodell: Grundlagen und Elemente einer übergreifenden Theorie.* Verlag Hans Huber, Bern, 1984

Kodratoff, Y. (1988) *Introduction to Machine Learning.* Pitman, London

Kodratoff, Y. and Michalski, R.S. (1990) (eds.). *Machine Learning: An Artificial Intelligence Approach*, Vol. 3, Morgan Kaufmann

Kubat, M. (1996) Second Tier for Decision Trees. *Machine Learning: Proceedings of the 13th International Conference*, Morgan Kaufmann Publishers, San Francisco, CA, 293–301

Langley, P. (1996). *Elements of Machine Learning*, Morgan Kaufmann, San Francisco, California

Langley, P., Zytkow, J.M. ,Simon, H.A., and Bradshaw, G.L. (1986). The Search for Regularity: Four Aspects of Scientific Discovery. In: R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), *Machine Learning: An Artificial Approach*, Vol 2, Morgan Kaufmann, Los Altos

Langley, P., Simon, H.A., Bradshaw, G.L., and Zytkow, J.M.(1987). *Scientific Discovery: Computational Explorations of the Creative Processes.* MIT Press

Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, Hertfordhsire

Lebowitz, M. (1987). Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning* 2:103–138

Michalski, R.S. (1969). On the Quasi-Minimal Solution of the General Covering Problem *Proceedings of the 5th International Symposium on Information Processing (FCIP'69)*, Vol.A3, Bled, Slovenia, 125–128.

Michalski, R.S. (1973a). Discovering Classification Rules Using Variable-Valued Logic System CL1. *Proceedings of the 3rd International Conference on Artificial Intelligence, IJCAI*, pp. 162–172

Michalski, R.S. (1973b), "AQVAL/1–Computer Implementation of a Variable-Valued Logic System VL1 and Examples of its Application to Pattern Recognition," Proceedings of the First International Joint Conference on Pattern Recognition, Washington, DC, pp. 3-17. October 30 - November 1.

Michalski, R.S. (1980) "Pattern Recognition as Rule-Guided Inductive Inference," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, pp. 349-361, July.

Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. *Artificial Intelligence*, 20:111-161

Michalski, R.S. and Stepp, R. (1983). Learning from Observation: Conceptual Clustering. In: R.S. Michalski, J.G. Carbonnell, and T.M. Mitchell (eds): *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann

Michalski, R.S. (1990). Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-Tiered Representation. In: Y. Kodratoff and R.S. Michalski (eds.) *Machine Learning: An Artificial Intelligence Approach*, Volume III, Morgan Kaufmann,63–102

Michalski, R.S. (1991). Toward a Unified Theory of Learning: An Outline of Basic Ideas. *Proceedings of the First World Conference on the Fundamentals of Artificial Intelligence*, Paris, July 1–5, 1991.

Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (eds.) (1983). *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann

Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (eds.) (1986). *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, Morgan Kaufmann

Michalski, R.S. and Tecuci, G. (eds.) (1994) *Machine Learning: A Multistrategy Approach*, Volume IV, Morgan Kaufmann.

Michie, D. (1988) Machine learning in the next five years. *EWSL-88 – Proc. 3rd European Working Session on Learning*, Glasgow, 1988. London: Pitman.

Mingers, J. (1989a). An Empirical Comparison of Selection Measures for Decision Tree Induction. *Machine Learning* 3:319–342

Mingers, J. (1989b) An empiricial comparison of pruning methods for decision-tree induction. *Machine Learning*, Vol. 4, No. 2.

Minsky, M. (1975). A Framework for Representing Knowledge. In: P.H. Winston (ed.) *The Psychology of Computer Vision.* McGraw-Hill, New York, 221–277

Minsky. M. and Papert, S. (1969). *Perceptrons. MIT* Press, Cambridge, MA.

Mitchell, T.M. (1982). Generalization as Search, *Artificial Intelligence* 18:203–226

Mitchell, T.M. (1996). *Machine Learning*, McGraw Hill

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

Muggleton, S. (1991). Inductive Logic programming. *New Generation Computing* 8:295–318

Muggleton S. (ed.) (1992). *Inductive Logic Programming.* Academic Press.

Niblett, T. (1987). Constructing Decision Trees in Noisy Domains. In: I. Bratko and N. Lavrač (eds.) *Progress in Machine Learning.* Sigma Press, Wilmslow, England

Niblett, T. and Bratko, I. (1986) Learning decision trees in noisy domains. In: *Expert Systems 86: Proc. Expert Systems 86 Conf.* (ed. M. Bramer) Cambridge Univ. Press.

Núñez, M. (1991). The Use of Background Knowledge in Decision Tree Induction. *Machine Learning* 6:231–350

Quinlan, J.R. (1986). Induction of Decision Trees. *Machine Learning* 1:81–106

Quinlan, J.R. (1990a). Probabilistic Decision Trees. In: Kodratoff,Y. - Michalski,R.S. (eds.) *Machine Learning: An Artificial Intelligence Approach*, Volume III, Morgan Kaufmann, 140–152

Quinlan, J.R. (1990b). Learning Logical Definitions from Relations. *Machine Learning*, 5:239–266

Quinlan, J.R. and Cameron-Jones, R.M. (1993). FOIL: A Midterm Report. *Proceedings of the European Conference on Machine Learning*, 3–20

Rissland. E. and Ashley, K. (1989). A Case-Based System for Trade Secrets Law. *Proceedings of the First International Conference on Artificial Intelligence and Law*, Boston, MA: ACM Press, 60–66

Rosenblatt, M. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65:386–408

Rosenblatt, M. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington, D.C.

Rumelhart, D., Hinton, G. and Williams, J. (1986). Learning Internal Representations by Error Propagation. In: D. Rumelhart and J. McClelland (eds.), *Parallel Distributed Processing*, MIT Press, Cambridge, Vol.1, 318–362

Sethi, I.K. (1990). Entropy Nets: From Decision Trees to Neural Networks. *Proceedings of the IEEE*, 78:1605–1613

Sethi, I.K. and Sarvarayudu, G.P.R.(1982). Hierarchical Classsifier Design Using Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4: 441–445

Sutton, R.S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning* 3:9–44

Towell, G.G., Shavlik, J., and Noordewier, M.O. (1990). Refinement of Approximate Domain Theories by Knowledge-Based Networks. *Proceedings of the Eight National Conference on Artificial Intelligence*, 861–866

Towell, G.G., Craven, M.W., and Shavlik, J. (1991). Constructive Induction in Knowledge-Based Neural Networks. *Proceedings of the 8th International Workshop on Machine Learning*, San Mateo, 213–217.

Vafaie, H. and De Jong, K.A. (1994). "Improving the Performance of a Rule Induction System Using Genetic Algorithms," in Machine Learning: A Multistrategy Approach, Vol. IV, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA.

Venturini, G. (1993). SIA: a Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts. *Proceedings of the European Conference on Machine Learning*, Vienna, April 1993, 280–296

Widmer, G. and Kubat, M. (1993). Effective Learning in Dynamic Environments by Explicit Context Tracking. *Proceedings of the European Conference on Machine Learning ECML'93* Vienna, 3–7 April, 227–243

Widmer, G. and Kubat, M. (1996). Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23:69–101

Winston, P.H. (1970). Learning Structural Descriptions from Examples. Technical report AI-TR-231, MIT Cambridge, Mass, September

Wnek, J, Kaufman, K., Bloedorn, E., and Michalski, R.S. (1995). Inductive Learning System AQ15c: The Method and User's Guide. *Reports of the Machine Learning and Inference Laboratory, MLI 95-4*, Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA

Zhang, J. (1991): Integrating Symbolic and Subsymbolic Approaches in Learning Flexible Concepts. *Proceedings of the 1st International Workshop on Multistrategy Learning*, Harpers Ferry, U.S.A., November 7–9, 289–304