Knowledge Representation Representation Formalisms in Machine Learning

Claude Sammut
University of New South Wales

1 Introduction

In 1956, Bruner, Goodnow and Austin published their book *A Study of Thinking*, which became a landmark in psychology and would later have a major impact on machine learning. The experiments reported by Bruner, Goodnow and Austin were directed towards understanding a human's ability to categorise and how categories are learned.

> We begin with what seems a paradox. The world of experience of any normal man is composed of a tremendous array of discriminably different objects, events, people, impressions...But were we to utilize fully our capacity for registering the differences in things and to respond to each event encountered as unique, we would soon be overwhelmed by the complexity of our environment... The resolution of this seeming paradox ... is achieved by man's capacity to categorize. To categorise is to render discriminably different things equivalent, to group objects and events and people around us into classes... The process of categorizing involves ... an act of invention... If we have learned the class "house" as a concept, new exemplars can be readily recognised. The category becomes a tool for further use. The learning and utilization of categories represents one of the most elementary and general forms of cognition by which man adjusts to his environment.

The first question that they had to deal with was that of representation: what is a concept? They assumed that objects and events could be described by a set of attributes and were concerned with how inferences could be drawn from attributes to class membership. Categories were considered to be of three types: conjunctive, disjunctive and relational.

> ...when one learns to categorise a subset of events in a certain way, one is doing more than simply learning to recognise instances encountered. One is also learning a rule that may be applied to new instances. The concept or category is basically, this "rule of grouping" and it is such rules that one constructs in forming and attaining concepts.

The notion of a rule as an abstract representation of a concept in the human mind came to be questioned by psychologists and there is still no good theory to explain how we store concepts. However, the same questions about the nature of representation arise in machine learning, for the choice of representation heavily determines the nature of a learning algorithm. Thus, one critical point of comparison among machine learning algorithms is the method of knowledge representation employed.

In this chapter we will discuss various methods of representation and compare them according to their power to express complex concepts and the effects of representation on the time and space costs of learning.

2 Learning, Measurement and Representation

A learning program is one that is capable of improving its performance through experience. Given a program, P, and some input, x, a normal program would yield the same result $P(x) = y$ after every application. However, a learning program can alter its initial state so that its performance is modified with each application. Thus, we can say $P(x|q) = y$. That is, y is the result of applying program P to input, x, given the initial state, q. The goal of learning is to construct a new initial, q', so that the program alters its behaviour to give a more accurate or quicker result. Thus, one way of thinking about what a learning program does is that it builds an increasingly accurate approximation to a mapping from input to output.

The most common learning task is that of acquiring a function which maps objects, that share common properties, to the same class value. This is the categorisation problem to which Bruner, Goodnow and Austin referred and much of our discussion will be concerned with categorisation.

Learning experience may be in the form of examples from a trainer or the results of trial and error. In either case, the program must be able to represent its observations of the world, and it must also be able to represent hypotheses about the patterns it may find in those observations. Thus, we will often refer to the *observation language* and the *hypothesis language*. The observation language describes the inputs and outputs of the program and the hypothesis language describes the internal state of the learning program, which corresponds to its theory of the concepts or patterns that exist in the data.

The input to a learning program consists of descriptions of objects from the universe and, in the case of supervised learning, an output value associated with the example. The universe can be an abstract one, such as the set of all natural numbers, or the universe may be a subset of the real-world. No matter which method of representation we choose, descriptions of objects in the real world must ultimately rely on measurements of some properties of those objects. These may be physical properties such as size, weight, colour, etc or they may be defined for objects, eg. the length of time a person has been employed for the purpose of approving a loan. The accuracy and reliability of a learned concept depends heavily on the accuracy and reliability of the measurements.

A program is limited in the concepts that it can learn by the representational capabilities of both observation and hypothesis languages. For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others clearly places bounds on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language. The hypothesis language also places constraints on what may and may not be learned. For example, in the

language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first order logic, can easily be used to describe relationships.

Unfortunately, representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger the search space. When the search space is small, it is possible to use 'brute force' search methods. If the search space is very large, additional knowledge is required to reduce the search.

We will divide our attention among three different classes of machine learning algorithms that use distinctly different approaches to the problem of representation:

Instance-based learning algorithms learn concepts by storing prototypic instances of the concept and do not construct abstract representations at all.

Function approximation algorithms include connectionist and statistics methods. These algorithms are most closely related to traditional mathematical notions of approximation and interpolation and represent concepts as mathematical formulae.

Symbolic learning algorithms learn concepts by constructing a symbolic which describes a class of objects. We will consider algorithms that work with representations equivalent to propositional logic and first-order logic.

3 Prototypes

The simplest form of learning is memorisation. When an object is observed or the solution to a problem is found, it is stored in memory for future use. Memory can be thought of as a look up table. When a new problem is encountered, memory is searched to find if the same problem has been solved before. If an exact match for the search is required, learning is slow and consumes very large amounts of memory. However, approximate matching allows a degree of generalisation that both speeds learning and saves memory.

For example, if we are shown an object and we want to know if it is a chair, then we compare the description of this new object with descriptions of 'typical' chairs that we have encountered before. If the description of the new object is 'close' to the description of one of the stored instances then we may call it a chair. Obviously, we must defined what we mean by 'typical' and 'close'.

To better understand the issues involved in learning prototypes, we will briefly describe three experiments in *Instance-based learning* (IBL) by Aha, Kibler and Albert (1991). IBL learns to classify objects by being shown examples of objects, described by an attribute/value list, along with the class to which each example belongs.

Experiment 1

In the first experiment (IB1), to learn a concept simply required the program to store every example. When an unclassified object was presented for classification by the program, it used a simple Euclidean distance measure to determine the nearest neighbour of the object and the class given to it was the class of the neighbour.

This simple scheme works well, and is tolerant to some noise in the data. Its major disadvantage is that it requires a large amount of storage capacity.

Experiment 2

The second experiment (IB2) attempted to improve the space performance of IB1. In this case, when new instances of classes were presented to the program, the program attempted to classify them. Instances that were correctly classified were ignored and only incorrectly classified instances were stored to become part of the concept.

While this scheme reduced storage dramatically, it was less noise-tolerant than the first.

Experiment 3

The third experiment (IB3) used a more sophisticated method for evaluating instances to decide if they should be kept or not. IB3 is similar to IB2 with the following additions. IB3 maintains a record of the number of correct and incorrect classification attempts for each saved instance. This record summarised an instance's classification performance. IB3 uses a significance test to determine which instances are good classifiers and which ones are believed to be noisy. The latter are discarded from the concept description. This method strengthens noise tolerance, while keeping storage requirements down.

Discussion

Figure 1 shows the boundaries of an imaginary concept in a two dimensions space. The dashed lines represent the boundaries of the target concept. The learning procedure attempts to approximate these boundaries by nearest neighbour matches. Note that the boundaries defined by the matching procedure are quite irregular. This can have its advantages when the target concept does not have a regular shape.

Learning by remembering typical examples of a concept has several other advantages. If an efficient indexing mechanism can be devised to find near matches, this representation can be very fast as a classifier since it reduces to a table look up. It does not require any sophisticated reasoning system and is very flexible. As we shall see later, representations that rely on abstractions of concepts can run into trouble with what appear to be simple concepts. For example, an abstract representation of a chair may consist of a description of the number legs, the height, etc. However, exceptions abound since anything that can be sat on can be thought of as a chair. Thus, abstractions must often be augmented by lists of exceptions. Instance-based representation does not

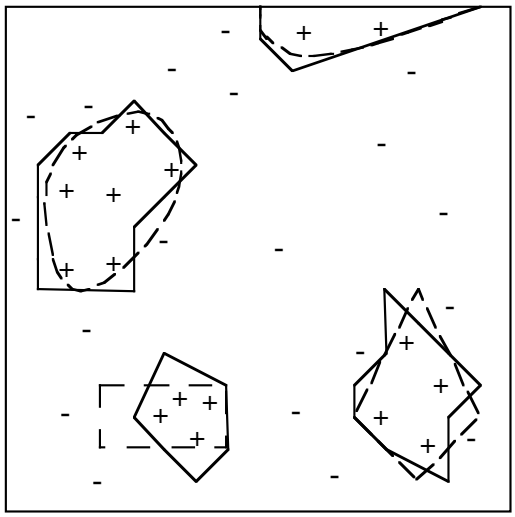


Figure 1. The extension of an IBL concept is shown in solid lines. The dashed lines represent the target concept. A sample of positive and negative examples is shown. Adapted from Aha, Kibler and Albert (1991).

suffer from this problem since it only consists exceptions and is designed to handle them efficiently.

One of the major disadvantages of this style of representation is that it is necessary to define a similarity metric for objects in the universe. This can often be difficult to do when the objects are quite complex.

Another disadvantage is that the representation is not human readable. What does a collection of typical instances tell us about the concept that has been learned?

4 Function approximation

Statistical and connectionist approaches to machine learning are related to function approximation methods in mathematics. For the purposes of illustration let us assume that the learning task is one of classification. That is, we wish to find ways of grouping objects in a universe. In Figure 2 we have a universe of objects that belong to either of two classes '+' or '–'.

By function approximation, we describe a surface that separates the objects into different regions. The simplest function is that of a line and linear regression methods and perceptrons are used to find linear discriminant functions.

A perceptron is a simple pattern classifier. Given a binary input vector, **x**, a weight vector, **w**, and a threshold value, T, if,

$$\sum_i w_i \cdot x_i > T$$

then the output is 1, indicating membership of a class, otherwise it is 0, indicating exclusion from the class. Clearly, $\mathbf{w} \cdot \mathbf{x} = T$ describes a hyperplane

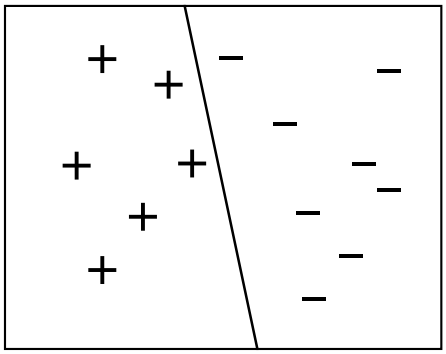


Figure 2: A linear discrimination between two classes

and the goal of perceptron learning is to find a weight vector, **w**, that results in correct classification for all training examples. The perceptron learning algorithm is quite straight forward. All the elements of the weight vector are initially set to 0. For each training example, if the perceptron outputs 0 when it should output 1 then add the input vector to the weight vector; if the perceptron outputs 1 when it should output 0 then subtract the input vector to the weight vector; otherwise, do nothing. This is repeated until the perceptron yields the correct result for each training example. The algorithm has the effect of reducing the error between the actual and desired output.

The perceptron is an example of a *linear threshold unit* (LTU). A single LTU can only recognise one kind of pattern, provided that the input space is linearly separable. If we wish to recognise more than one pattern, several LTU's can be combined. In this case, instead of having a vector of weights, we have an array. The output will now be a vector:

$$\mathbf{u} = \mathbf{W}\mathbf{x}$$

where each element of **u** indicates membership of a class and each row in **W** is the set of weights for one LTU. This architecture is called a *pattern associator.*

LTU's can only produce linear discriminant functions and consequently, they are limited in the kinds of classes that can be learned. However, it was found that by cascading pattern associators, it is possible to approximate decision surfaces that are of a higher order than simple hyperplanes. In cascaded system, the outputs of one pattern associator are fed into the inputs of another, thus:

$$\mathbf{u} = \mathbf{W}(\mathbf{V}\mathbf{x})$$

This is the scheme that is followed by multi-layer neural nets (Figure 3).

To facilitate learning, a further modification must be made. Rather than using a simple threshold, as in the perceptron, multi-layer networks usually use a non-linear threshold such a sigmoid function, such as

$$\frac{1}{1 + e^{-x}}$$

Like perceptron learning, back-propagation attempts to reduce the errors between the output of the network and the desired result. However,

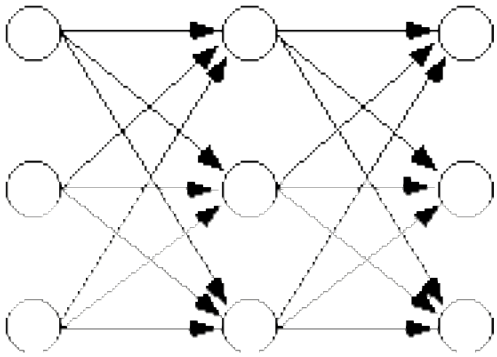


Figure 3. A multi-layer network.

assigning blame for errors to hidden units (ie. nodes in the intermediate layers), is not so straightforward. The error of the output units must be propagated back through the hidden units. The contribution that a hidden unit makes to an output unit is related to the strength of the weight on the link between the two units and the level of activation of the hidden unit when the output unit was given the wrong level of activation. This can be used to estimate the error value for a hidden unit in the penultimate layer, and that can, in turn, be used in make error estimates for earlier layers.

Despite the non-linear threshold, multi-layer networks can still be thought of as describing a complex collection of hyperplanes that approximate the required decision surface.

Discussion

Function approximation methods, such as the ones discussed above, can often produce quite accurate classifiers because they are capable of construction complex decision surfaces. However, knowledge is stored as weights in a matrix. Thus, the results of learning are not easily available for inspection by a human reader. Moreover, the design of a network usually requires informed guesswork on the part of the user in order to obtain satisfactory results. Although some effort has been devoted to extracting meaning from networks, the still communicate little about the data.

Connectionist learning algorithms are still computationally expensive. A critical factor in their speed is the encoding of the inputs to the network. This is also critical to genetic algorithms and we will illustrate that problem in the next section.

5 Genetic Algorithms

Genetic algorithms (Holland, 1975) perform a search for the solution to a problem by generating candidate solutions from the space of all solutions and testing the performance of the candidates. The search method is based on ideas from genetics and the size of the search space is determined by the representation of the domain. An understanding of genetic algorithms will be aided by an example.

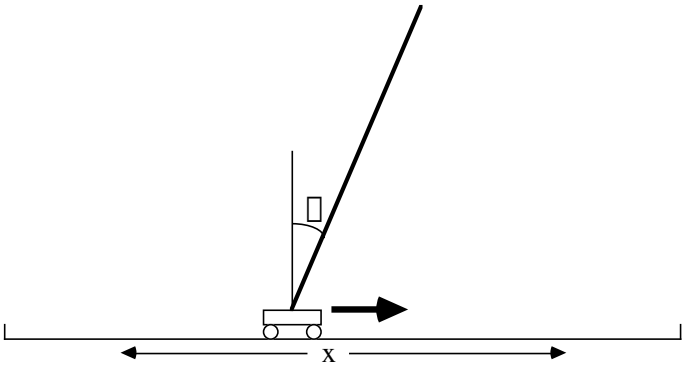


Figure 4. A Pole Balancer

A very common problem in adaptive control is learning to balance a pole that is hinged on a cart that can move in one dimension along a track of fixed length, as show in Figure 4. The control must use 'bang-bang' control, that is, a force of fixed magnitude can be applied to push the cart to the left or right.

Before we can begin to learn how to control this system, it is necessary to represent it somehow. We will use the BOXES method that was devised by Michie and Chambers (1968). The measurements taken of the physical system are the angle of the pole, θ, and its angular velocity and the position of the cart, x, and its velocity. Rather than treat the four variables as continuous values, Michie and Chambers chose to discretise each dimension of the state space. One possible discretisation is shown in Figure 5.

This discretisation results in 3 x 3 x 6 x 3 = 162 'boxes' that partition the state space. Each box has associated with it an action setting which tells the controller that when the system is in that part of the state space, the controller should apply that action, which is a push to the left or a push to the right. Since there is a simple binary choice and there are 162 boxes, there are 2^{162} possible control strategies for the pole balancer.

The simplest kind of learning in this case, is to exhaustively search for the right combination. However, this is clearly impractical given the size of the search space. Instead, we can invoke a genetic search strategy that will reduce

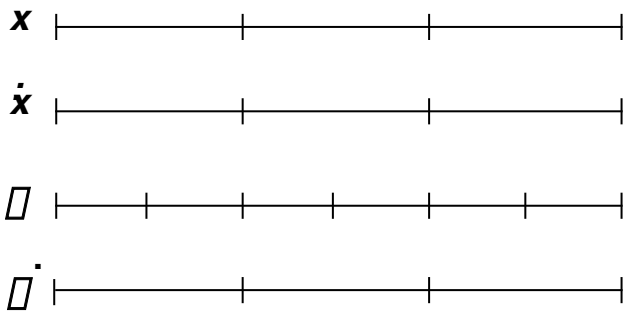


Figure 5. Discretisation of pole balancer state space.

the amount of search considerably.

In genetic learning, we assume that there is a population of individuals, each one of which, represents a candidate problem solver for a given task. Like evolution, genetic algorithms test each individual from the population and only the fittest survive to reproduce for the next generation. The algorithm creates new generations until at least one individual is found that can solve the problem adequately.

Each problem solver is a *chromosome*. A position, or set of positions in a chromosome is called a *gene*. The possible values (from a fixed set of symbols) of a gene are known as *alleles*. In most genetic algorithm implementations the set of symbols is {0, 1} and chromosome lengths are fixed. Most implementations also use fixed population sizes.

The most critical problem in applying a genetic algorithm is in finding a suitable encoding of the examples in the problem domain to a chromosome. A good choice of representation will make the search easy by limiting the search space, a poor choice will result in a large search space. For our pole balancing example, we will use a very simple encoding. A chromosome is a string of 162 boxes. Each box, or gene, can take values: 0 (meaning push left) or 1 (meaning push right). Choosing the size of the population can be tricky since a small population size provides an insufficient sample size over the space of solutions for a problem and large population requires a lot of evaluation and will be slow. In this example, 50 is a suitable population size.

Each iteration in a genetic algorithm is called a *generation*. Each chromosome in a population is used to solve a problem. Its performance is evaluated and the chromosome is given some rating of fitness. The population is also given an overall fitness rating based on the performance of its members. The fitness value indicates how close a chromosome or population is to the required solution. For pole balancing, the fitness value of a chromosome may be the number of time steps that the chromosome is able to keep the pole balanced for.

New sets of chromosomes are produced from one generation to the next. Reproduction takes place when selected chromosomes from one generation are recombined with others to form chromosomes for the next generation. The new ones are called *offspring*. Selection of chromosomes for reproduction is based on their fitness values. The average fitness of population may also be calculated at end of each generation. For pole balancing, individuals whose fitness is below average are replaced by reproduction of above average chromosomes. The strategy must be modified if two few or two many chromosomes survive. For example, at least 10% and at most 60% must survive.

Operators that recombine the selected chromosomes are called genetic operators. Two common operators are *crossover* and *mutation*. Crossover exchanges portions of a pair of chromosomes at a randomly chosen point called the crossover point. Some Implementations have more than one crossover point. For example, if there are two chromosomes, *X* and *Y*:

X = 1001 01011 Y = 1110 10010

and the crossover point is 4, the resulting offspring are:

O1 = 100110010 O2 = 1110 01011

Offspring produced by crossover cannot contain information that is not already in the population, so an additional operator, mutation, is required. Mutation generates an offspring by randomly changing the values of genes at one or more gene positions of a selected chromosome. For example, if the following chromosome,

Z = 100101011

is mutated at positions 2, 4 and 9, then the resulting offspring is:

O = 110001010

The number of offspring produced for each new generation depends on how members are introduced so as to maintain a fixed population size. In a *pure* replacement strategy, the whole population is replaced by a new one. In an *elitist* strategy, a proportion of the population survives to the next generation.

In pole balancing, all offspring are created by crossover (except when more the 60% will survive for more than three generations when the rate is reduced to only 0.75 being produced by crossover). Mutation is a background operator which helps to sustain exploration. Each offspring produced by crossover has a probability of 0.01 of being mutated before it enters the population. If more then 60% will survive, the mutation rate is increased to 0.25.

The number of offspring an individual can produce by crossover is proportional to its fitness:

$$\frac{\text{fitness value}}{\text{population fitness}} \times \text{No. of children}$$

where the number of children is the total number of individuals to be replaced. Mates are chosen at random among the survivors.

The pole balancing experiments described above, were conducted by Odetayo (1988) and required an average of 8165 trials before balancing pole. Of course, this may not be the only way of encoding the problem for a genetic algorithm and so a faster solution may be possible. However, this requires effort on the part of the user to devise a clever encoding.

6 Propositional Learning Systems

Rather than searching for discriminant functions, symbolic learning systems find expressions equivalent to sentences in some form of logic. For example, we may distinguish objects according to two attributes: size and colour. We may say that an object belongs to class 3 if its colour is red and its size is very small to medium. Following the notation of Michalski (1983), the classes in Figure 2 may be written as:

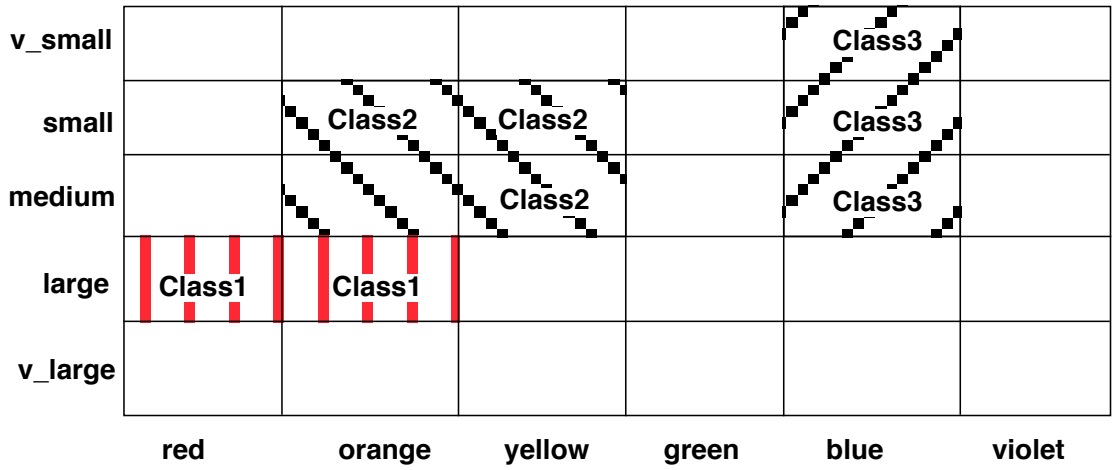


Figure 6: Discrimination on attributes and values

$$class1 \leftarrow size = large \wedge colour \in \{red, orange\}$$
$$class2 \leftarrow size \in \{small, medium\} \wedge colour \in \{orange, yellow\}$$
$$class3 \leftarrow size \in \{v\_small \ldots medium\} \wedge colour = blue$$

Note that this kind of description partitions the universe into blocks, unlike the function approximation methods that find smooth surfaces to discriminate classes.

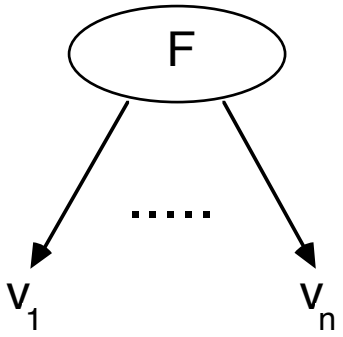
Interestingly, one of the popular early machine learning algorithms, Aq (Michalski, 1973), had its origins in switching theory. One of the concerns of switching theory is to find ways of minimising logic circuits, that is, simplifying the truth table description of the function of a circuit to a simple expression in Boolean logic. Many of the algorithms in switching theory take tables like Figure 6 and search for the best way of covering all of the entries in the table.

Aq, uses a *covering algorithm*, to build its concept description:

```
cover := {}
repeat
        select one positive example, e
        construct the set of all conjunctive expressions that
             cover e and no negative example in E-
        choose the 'best' expression, x, from this set
        add x as a new disjunctof the concept
        remove all positive examples covered by x
until there are no positive examples left
```

The 'best' expression is usually some compromise between the desire to cover as many positive examples as possible and the desire to have as compact and readable a representation as possible. In designing Aq, Michalski was particularly concerned with the expressiveness of the concept description language.

A drawback of the Aq learning algorithm is that it does not use statistical information, present in the training sample, to guide induction. However, decision tree learning algorithms (Quinlan, 1979; 1993) do. The basic method



- The algorithm operates over a set of training instances, *C*.
- If all instances in *C* are in class *P*, create a node *P* and stop. Otherwise select a feature, *F* and create a decision node.
- Partition the trannie in *C* into subsets according to the values v_i of *F*.
- Apply the algorithm recursively to each if the subsets of *C*.

Figure 5. Decision tree learning

of building a decision tree is summarised in Figure 5. An simple attribute/value representation is used and so, like Aq, decision trees are incapable of representing relational information. They are, however, very quick and easy to build.

Decision tree learning algorithms can be seen as methods for partitioning the universe into successively smaller rectangles with the goal that each rectangle only contains objects of one class. This is illustrated in Figure 6.

Discussion

Michalski has always argued in favour of rule-based representations over tree structured representations, on the grounds of readability. When the domain is complex, decision trees can become very 'bushy' and difficult to understand, whereas rules tend to be modular and can be read in isolation of the rest of the knowledge-base constructed by induction. On the other hand, decision trees induction programs are usually very fast. A compromise is to use decision tree induction to build an initial tree and then derive rules from

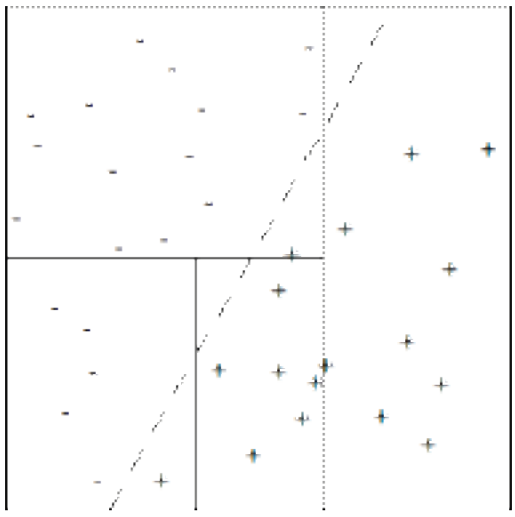


Figure 6. The dashed line shows the real division of objects in the universe. The solid lines show a decision tree approximation.

the tree thus transforming an efficient but opaque representation into a transparent one (Quinlan, 1987).

It is instructive to compare the shapes that are produced by various learning systems when they partition the universe. Figure 6 demonstrates one weakness of decision tree and other symbolic classification. Since they approximate partitions with rectangles (if the universe is 2-dimensional) there is an inherent inaccuracy when dealing with domains with continuous attributes. Function approximation methods and IBL may be able to attain higher accuracy, but at the expense of transparency of the resulting theory. It is more difficult to make general comments about genetic algorithms since the encoding method will affect both accuracy and readability.

As we have seen, useful insights into induction can be gained by visualising it as searching for a cover of objects in the universe. Unfortunately, there are limits to this geometric interpretation of learning. If we wish to learn concepts that describe complex objects and relationships between the objects, it becomes very difficult to visualise the universe. For this reason, it is often useful to rely on reasoning about the concept description language.

As we saw, the cover in Figure 4 can be expressed as clauses in propositional logic. We can establish a correspondence between sentences in the concept description language (the hypothesis language) and a diagrammatic representation of the concept. More importantly, we can create a correspondence between generalisation and specialisation operations on the sets of objects and generalisation and specialisation operations on the sentences of the language.

For example, Figure 7 shows two sets, labelled class 1 and class 2. It is clear that class 1 is a generalisation of class 2 since it includes a larger number of objects in the universe. We also call class 2 a specialisation of class 1. By convention, we say the *description* of class 1 is a generalisation of the *description* of class 2. Thus,

$$class1 \leftarrow size = large \tag{1}$$

is a generalisation of

$$class2 \leftarrow size = large \wedge colour = red \tag{2}$$

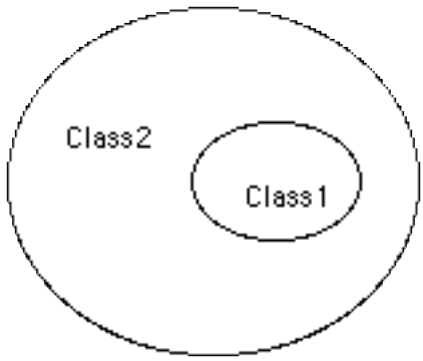


Figure 7: Generalisation as set covering

Once we have established the correspondence between sets of objects and their descriptions, it is often convenient to forget about the objects and only consider that we are working with expressions in a language. The reason is simple. Beyond a certain point of complexity, it is not possible to visualise sets, but it is relatively easy to apply simple transformations on sentences in a formal language. For example, clause (2) can be generalised very easily to clause (1) by dropping one of the conditions.

In the next section we will look at learning algorithms that deal with relational information. In this case, the emphasis on language is essential since geometric interpretations no longer provide us with any real insight into the operation of these algorithms.

7 Relations and Background Knowledge

Inductions systems, as we have seen so far, might be described as 'what you see is what you get'. That is, the output class descriptions use the same vocabulary as the input examples. However, we will see in this section, that it is often useful to incorporate background knowledge into learning.

We use a simple example from Banerji (1980) to the use of background knowledge. There is a language for describing instances of a concept and another for describing concepts. Suppose we wish to represent the binary number, 10, by a left-recursive binary tree of digits '0' and '1':

[head: [head: 1; tail: nil]; tail: 0]

'head' and 'tail' are the names of attributes. Their values follow the colon. The concepts of binary digit and binary number are defined as:

$$x \in digit \equiv x = 0 \quad x = 1$$

$$x \in num \equiv (tail(x) \in digit \wedge head(x) = nil)$$
$$(tail(x) \in digit \wedge head(x) \in num)$$

Thus, an object belongs to a particular class or concept if it satisfies the logical expression in the body of the description. Predicates in the expression may test the membership of an object in a previously learned concept.

Banerji always emphasised the importance of a description language that could 'grow'. That is, its descriptive power should increase as new concepts are learned. This can clearly be seen in the example above. Having learned to describe binary digits, the concept of digit becomes available for use in the description of more complex concepts such as binary number.

Extensibility is a natural and easily implemented feature of horn-clause logic. In addition, a description in horn-clause logic is a logic program and can be executed. For example, to recognise an object, a horn clause can be interpreted in a forward chaining manner. Suppose we have a set of clauses:

$$C_1 \leftarrow P_{11} \wedge P_{12} \quad (3)$$
$$C_2 \leftarrow P_{21} \wedge P_{22} \wedge C_1 \quad (4)$$

and an instance:

$$P_{11} \wedge P_{12} \wedge P_{21} \wedge P_{22} \quad (5)$$

Clause (3) recognises the first two terms in expression (5) reducing it to

$$P_{21} \square P_{22} \square C_1$$

Clause (4) reduces this to C2. That is, clauses (3) and (4) recognise expression (5) as the description of an instance of concept C2.

When clauses are executed in a backward chaining manner, they can either verify that the input object belongs to a concept or produce instances of concepts. In other words, we attempt to prove an assertion is true with respect to a background theory. Resolution (Robinson, 1965) provides an efficient means of deriving a solution to a problem, giving a set of axioms which define the task environment. The algorithm takes two terms and resolves them into a most general unifier, as illustrated in Figure 7 by the execution of a simple Prolog program.

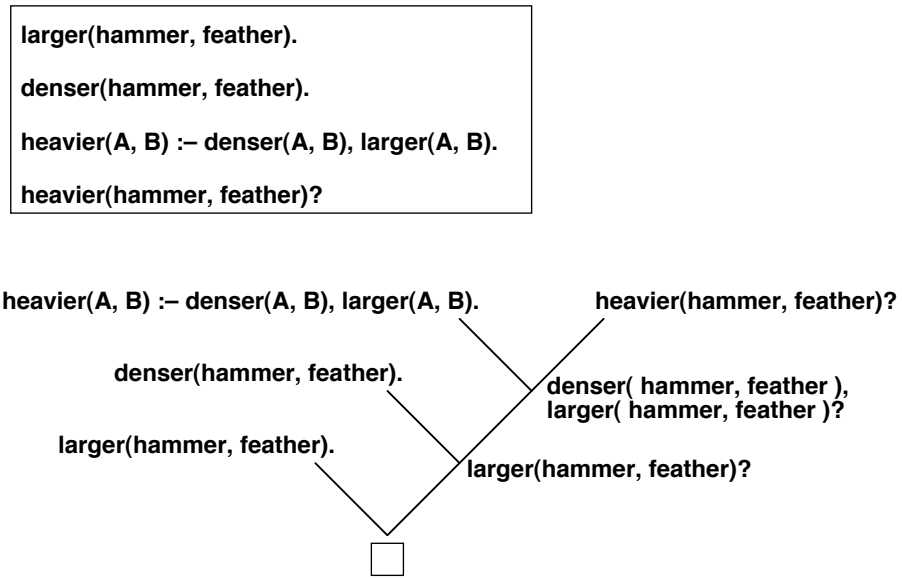


Figure 7: A resolution proof tree from Muggleton and Feng (1990).

The box in the figure contains clauses that make up the theory, or knowledge base, and the question to be answered, namely, "is it true that a hammer is heavier than a feather"? A resolution proof is a proof by refutation. That is, answer the question, we assume that it is false and then see if the addition, to the theory, of this negative statement results in a contradiction.

The literals on the left hand side of a Prolog clause are positive. Those on the left hand side are negative. The proof procedure looks for *complimentary* literals in two clauses, i.e. literals of opposite sign that unify. In the example in Figure 10,

$$heavier\,(A, B)$$

and

$$heavier\,(hammer, feather)$$

unify to create the first resolvent,

$$: \square denser(hammer, feather), heavier\,(hammer, feather)\ \text{de.}$$

A side effect of unification is to create variable substitutions $\{A / hammer, B / feather\}$. By continued application of resolution, we can eventually derive the empty clause, which indicates a contradiction.

Plotkin's (1970) work "originated with a suggestion of R.J. Popplestone that since unification is useful in automatic deduction by the resolution method, its dual might prove helpful for induction. The dual of the most general unifier of two literals is called the least general generalisation". At about the same time that Plotkin took up this idea, J.C. Reynolds was also developing the use of least general generalisations. Reynolds (1970) also recognised the connection between deductive theorem proving and inductive learning:

> Robinson's Unification Algorithm allows the computation of the greatest common instance of any finite set of unifiable atomic formulas. This suggests the existence of a dual operation of 'least common generalization'. It turns out that such an operation exists and can be computed by a simple algorithm.

The method of least general generalisations is based on *subsumption*. A clause C_1 subsumes, or is more general than, another clause C_2 if there is a substitution θ such that $C_2 \supseteq C_1\theta$.

The least general generalisation of

p(g(a), a) (6)

and p(g(b), b) (7)

is p(g(X), X). (8)

Under the substitution {a/X}, (8) is equivalent to (6). and under the substitution {b/X}, (8) is equivalent to (7). Therefore, the least general generalisation of *p(g(a), a)* and *p(g(b), b)* is *p(g(X), X)* and results in the inverse substitution {X/{a, b}}.

Buntine (1988) pointed out that simple subsumption is unable to take advantage of background information which may assist generalisation. Suppose we are given two instances of a concept cuddly_pet,

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge dog(X) \quad (9)$$
$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge cat(X) \quad (10)$$

Suppose we also know the following:

$$pet(X) \leftarrow dog(X) \quad (11)$$
$$pet(X) \leftarrow cat(X) \quad (12)$$

According to subsumption, the least general generalisation of (4) and (5) is:

$$cuddly\_pet(X) \leftarrow fluffy(X) \quad (13)$$

since unmatched literals are dropped from the clause. However, given the background knowledge, we can see that this is an over-generalisation. A better one is:

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge pet(X) \quad (14)$$

The moral being that a generalisation should only be done when the relevant background knowledge suggests it. So, observing (9), use clause (11) as a

rewrite rule to produce a generalisation which is clause (14). which also subsumes clause (10).

Buntine drew on earlier work by Sammut (Sammut and Banerji, 1986) in constructing his generalised subsumption. Muggleton and Buntine (1998) took this approach a step further and realised that through the application of a few simple rules, they could invert resolution as Plotkin and Reynolds had wished. Here are two of the rewrite rules in propositional form:

Given a set of clauses, the body of one of which is completely contained in the bodies of the others, such as:

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E$$
$$Y \leftarrow A \wedge B \wedge C$$

the *absorption* operation results in:

$$X \leftarrow Y \wedge D \wedge E$$
$$Y \leftarrow A \wedge B \wedge C$$

Intra-construction takes a group of rules all having the same head, such as:

$$X \leftarrow B \wedge C \wedge D \wedge E$$
$$X \leftarrow A \wedge B \wedge D \wedge F$$

and replaces them with:

$$X \leftarrow B \wedge D \wedge Z$$
$$Z \leftarrow C \wedge E$$
$$Z \leftarrow A \wedge F$$

These two operations can be interpreted in terms of the proof tree shown in Figure 7. Resolution accepts two clauses and applies unification to find the maximal common unifier. In the diagram, two clauses at the top of a "V" are resolved to produce the resolvent at the apex of the "V". Absorption accepts the resolvent and one of the other two clauses to produce the third. Thus, it inverts the resolution step.

Intra-construction automatically creates a new term in its attempt to simplify descriptions. This is an essential feature of inverse resolution since there may be terms in a theory that are not explicitly shown in an example and may have to be invented by the learning program.

Discussion

These methods and others (Muggleton and Feng, 1990; Quinlan, 1990) have made relational learning quite efficient. Because the language of Horn-clause logic is more expressive than the other concept description languages we have seen, it is now possible to learn far more complex concepts than was previously possible. A particularly important application of this style of learning is knowledge discovery. There are now vast databases accumulating information on the genetic structure of human beings, aircraft accidents, company inventories, pharmaceuticals and countless more. Powerful induction programs that use expressive languages may be a vital aid in discovering useful patterns in all these data.

For example, the realities of drug design require descriptive powers that encompass stereo-spatial and other long-range relations between different

parts of a molecule, and can generate, in effect, new theories. The pharmaceutical industry spends over $250 million for each new drug released onto the market. The greater part of this expenditure reflects today's unavoidably "scatter-gun" synthesis of compounds which *might*} possess biological activity. Even a limited capability to construct predictive theories from data promises high returns.

The relational program Golem was applied to the drug design problem of modelling structure-activity relations (King *et al*, 1992). Training data for the program was 44 trimethoprim analogues and their observed inhibition of E. coli dihydrofolate reductase. A further 11 compounds were used as unseen test data. Golem obtained rules that were statistically more accurate on the training data and also better on the test data than a Hansch linear regression model. Importantly, relational learning yields understandable rules that characterise the stereochemistry of the interaction of trimethoprim with dihydrofolate reductase observed crystallographically. In this domain, relational learning thus offers a new approach which complements other methods, directing the time-consuming process of the design of potent pharmacological agents from a lead compound, variants of which need to be characterised for likely biological activity before committing resources to their synthesis.

8 Conclusion

We have now completed a rapid tour of a variety of learning algorithms and seen how the method of representing knowledge is crucial in the following ways:

- Knowledge representation determines the concepts that an algorithm can and cannot learn.
- Knowledge representation affects the speed of learning. Some representations lend themselves to more efficient implementation than others. Also, the more expressive the language, the larger is the search space.
- Knowledge representation determines the readability of the concept description. A representation that is opaque to the user may allow the program to learn, but a representation that is transparent also allows the user to learn.

References

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-Based Learning Algorithms. *Machine Learning*, **6**(1), 37-66.

Banerji, R. B. (1980). *Artificial Intelligence: A Theoretical Approach*. New York: North Holland.

Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1956). *A Study of Thinking*. New York: Wiley.

Buntine, W. (1988). Generalized Subsumption and its Applications to Induction and Redundancy. *Artificial Intelligence*, **36**, 149-176.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan: University of Michigan Press.

King, R. D., Lewis, R. A., Muggleton, S., & Sternberg, M. J. E. (1992). Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationship of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy Science*, **89**.

Michalski, R. S. (1973). Discovering Classification Rules Using Variable Valued Logic System VL1. In *Third International Joint Conference on Artificial Intelligence*. (pp. 162-172).

Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Palo Alto: Tioga.

Michie, D., & Chambers, R. A. (1968). Boxes: An Experiment in Adaptive Control. In E. Dale & D. Michie (Eds.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd.

Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In R. S. Michalski, T. M. Mitchell, & J. G. Carbonell (Eds.), *Proceedings of the Fifth International Machine Learning Conference*. (pp. 339-352). Ann Arbor, Michigan: Morgan Kaufmann.

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *First Conference on Algorithmic Learning Theory*, Tokyo: Omsha.

Odetayo, M. (1988) Genetic Algorithms for Control a Dynamic Physical System. M.Sc. Thesis, Strathclyde University.

Plotkin, G. D. (1970). A Note on Inductive Generalization. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 5*. (pp. 153-163). Edinburgh University Press.

Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In D. Michie (Eds.), *Expert Systems in the Micro-Electronic Age*. Edinburgh: Edinburgh University Press.

Quinlan, J. R. (1987). Generating production rules from decision trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*. (pp. 304-307). San Mateo, CA: Morgan Kaufmann.

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

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Reynolds, J. C. (1970). Transformational Systems and the Algebraic Structure of Atomic Formulas. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 5*. (pp. 153-163).

Robinson, J. A. (1965). A Machine Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, **12**(1), 23-41.

Sammut, C. A., & Banerji, R. B. (1986). Learning Concepts by Asking Questions. In R. S. Michalski Carbonell, J.G. and Mitchell, T.M. (Eds.), *Machine Learning: An Artificial Intelligence Approach, Vol 2*. (pp. 167-192). Los Altos, California: Morgan Kaufmann.

Shapiro, E. Y. (1981). *Inductive Inference of Theories From Facts* (Technical Report No. 192). Yale University.