

# Machine Learning: Techniques and Recent Developments

Peter Clark, Turing Institute

1990

## Abstract

The use of expert systems is becoming more and more widespread, making the need for appropriate machine learning techniques more acute to help ease the knowledge acquisition bottleneck. Additionally, the increasing number of large databases offers a vast potential for the automatic generation of new knowledge by machines and its communication to people in a comprehensible form. In response to these events, this paper provides an overview of current machine learning work with a particular emphasis on rule induction techniques. Firstly we provide a summary of existing rule induction techniques, including descriptions of the ID3 and AQ algorithms. Secondly, we review recent developments in rule induction technology which overcome some of the practical limitations of these basic algorithms including noise handling, probabilistic classification, large data sets and incremental learning. Finally, we describe the state of current research in machine learning and the directions in which it is heading, addressing the difficult problems of constructive induction and representation change.

## 1 Introduction

### 1.1 The Role of Machine Learning

The use of expert systems is becoming more and more widespread. A survey in 1988 reported the number of deployed systems had risen sharply from around 50 in the previous year to 1400, and the number under development increased from 2500 to 8500 [41]. Expert systems are characterised by the use of a particular programming methodology in which domain-specific knowledge is clearly separated from the more general inference machinery within the system. This methodology has several advantages, including easier inspection and modification of the knowledge which the system is using, and the generation of explanations by the system describing how it arrived at its conclusions.

As a result of this expansion, issues of how to incorporate and refine knowledge within such systems have risen in importance. The difficulty of manually acquiring knowledge from an expert is now well recognised (sometimes referred to as the 'Feigenbaum Bottleneck'), and hence systems for automatically learning in ways more sophisticated than simply by being told are in increasing use.

In addition to assisting in the knowledge acquisition task, there are other motivational factors for developing machines which can learn for themselves. Firstly, the ability to learn is a primary characteristic of intelligence and hence is an essential area of study in the quest for developing artificially intelligent systems. Secondly, there exists the goal

of building machines which can acquire knowledge and form theories which were previously unknown, and then communicate this knowledge to people. In this way, machines may enhance the global body of scientific and other knowledge. Michie has referred to this phenomenon as ‘superarticulacy’, and it has already been demonstrated in limited domains of application [24].

One particular form of learning, that of inducing classification rules from a set of training examples, has received substantial attention within the machine learning field and represents the learning method most frequently and successfully used in expert system applications (e.g. [28]). Because of its relative success in applications we focus on this technique in this paper, first describing two algorithms for rule induction forming the basis of many rule induction systems, and then reporting on recent developments in this area seeking to extend their applicability and overcome the difficulties presented by real-world applications. Following this, we examine some of the more fundamental limitations of this approach and briefly survey some of the other methods currently being developed which strive towards producing more powerful and flexible learning machines.

## 1.2 The Nature of Machine Learning

An adaptive system must have, by definition, the capacity to perform a task in more than one way. Consequently the system must make choices about the most appropriate course of action to take, and if the system is adaptive it should be able to modify its choice-making behaviour should a choice turn out to be inappropriate.

In order to make choices it is necessary to predict the likely outcomes of making them, requiring some kind of internal model of the world. An adaptive system can be viewed as performing two processes involving this model: firstly, to use it to respond most appropriately to the environment, and secondly to continuously extend and correct such a model in the light of success or failure, keeping it up-to-date and efficient.

This process of self-analysis and modification can usefully be viewed as a special kind of problem-solving task – one in which the problem to be solved is that of self-diagnosis and improvement rather than the diagnosis of something external to the system. Thus it follows that many of the problem-solving techniques used in ‘non-learning’ systems are also important in learning systems. A learning system can be viewed as performing a search not for a solution to some external problem but for an improved representation of knowledge within itself. Inductive leaps and generalisation by a machine at one level can be regarded as the result of this special type of problem-solving search at a higher level.

This paradigm of learning as search is of central importance to machine learning, and many systems operate by searching a space of representations to find that best fitting known observations. Two examples of algorithms which do this are the ID3 and AQ algorithms, described in detail later. However, these algorithms use a fairly limited representation language requiring careful formulation of the learning task by the user in order to permit effective learning.

To move forward from such systems brings the designer face to face with a serious dilemma, concerning the trade-off between flexibility and tractability of learning. Allowing the system greater flexibility to learn by introducing a more powerful representation language for embodying learned knowledge immediately creates a major search problem, which can easily become computationally infeasible to perform. Methods for handling this trade-off are a major focus of current research, and we review some of these developments later.

### 1.3 Michie’s Criteria

Before embarking on more detailed descriptions of learning algorithms, we make some more general comments concerning types of machine learning. Recently, alternative paradigms of machine learning besides the traditional AI-type symbolic learning have received substantial attention. In particular, increasing power of computers is enabling connectionist and genetic techniques to become feasible areas of study, although their role as artificial intelligence research remains controversial.

Michie [23] presents three criteria for machine learning. The **weak criterion** states that machine learning occurs when a “system uses sample data to generate an updated basis for improved performance on subsequent data”. This definition assumes that all that is important is problem-solving performance, and ignores other desirable properties of an intelligent learning system such as ability to explain its reasoning. Statistical, genetic and connectionist methods of learning also fall within this criterion.

Michie’s **strong criterion** of machine learning states that the system must additionally be able to “communicate its internal updates in explicit symbolic form”, hence be able to explain in an understandable way what it has learned. The **ultra-strong criterion** goes one step further in insisting that internal updates be communicated in “operationally effective” as well as explicit form, i.e. in a form which additionally allows the expert to improve his or her own performance as well as understand the machine’s behaviour.

We have drawn on these different criteria to highlight the distinction between different forms of machine learning. In this paper we are concerned with the strong and ultra-strong type of learning requiring comprehensibility as well as performance, and thus we focus on systems learning structured, symbolic representations of the world.

## 2 The Inductive Rule Learning Methodology

### 2.1 Introduction

There are a wide variety of techniques used for machine learning – however the technique which has perhaps received the most attention and has been most commercially successful to date (e.g. [28]) has been the paradigm of learning classification rules from a set of training examples. In this paradigm the learning system searches a space of rules to find those which ‘best’ classify the training examples, where ‘best’ is defined in terms of accuracy and comprehensibility. The rules represent generalisations of the training examples with which the system was presented.

In this section we first describe two algorithms for learning such rules, namely the ID3 and AQ algorithms. These algorithms have been used as the basis of several machine learning systems, for example ID3 in ASSISTANT [15], ACLS [34] and C4 [40], and the AQ algorithm in AQ11 [20], AQR [10] and AQ15 [17]. An example of the form of the inputs and outputs which these algorithms receive and produce is shown in Figure 1. Each training example is described by giving **values** for a fixed number of (user-selected) **attributes**, plus the corresponding **class** of which the example is a member.

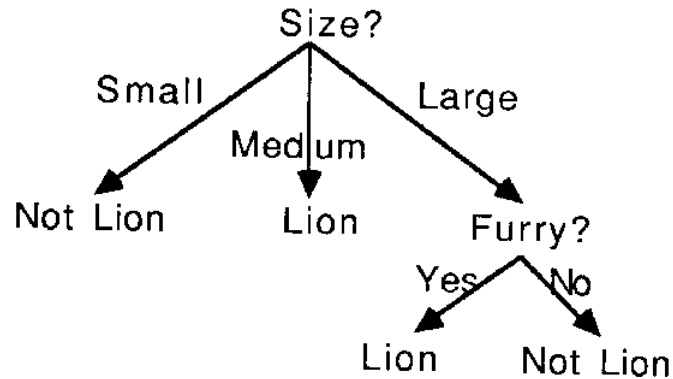
The inductive rule learning paradigm used in the AQ and ID3 algorithms (as well as others) is based on a simple pattern recognition model of learning, in which correlations between observable features and some final classification are sought for. The initial features used to describe examples are chosen manually, and classification is made into one of a fixed number of user-defined classes. Clancey refers to this as a task of “heuristic

Figure 1: Examples of Rule Induction

Training Examples Input to Rule Induction Algorithms

Attributes			Class
Furry?	Age?	Size?	
furry	old	large	lion
not furry	young	large	not lion
furry	young	medium	lion
furry	old	small	not lion
furry	young	small	not lion
furry	young	large	lion
not furry	young	small	not lion
not furry	old	large	not lion

Decision Tree Output by ID3



Decision Rules Output by AQ

```

if    furry=yes
and   size=large
then  class=lion.

if    size=medium
then  class=lion.

if    furry=no
then  class=not lion.

if    size=small
then  class=not lion.
  
```

classification” [6]. The systems do not make use of any other domain-specific information beyond that of the training examples themselves.

The rules which ID3 and AQ produce constitute a simple ‘model’ of the world, automatically generated from the observations with which they have been presented. Although the structure of this model is simple, the operations they perform of generalising, compressing and organising data are fundamental to learning. Later in this paper we discuss the possibilities and problems of extending these learning systems to acquire and refine more complex, structured representations of the world.

## 2.2 The ID3 Algorithm

### 2.2.1 Knowledge Representation

The ID3 algorithm [38] is a descendent of Hunt et al’s Concept Learning System [13]. The ‘rules’ which ID3 learns are represented as *decision trees*. A decision tree is like a flow chart, in which a node of the tree represents a test on an attribute and each outgoing branch corresponds to a possible result of this test. Each leaf node represents a classification to be assigned to an example, as shown in Figure 1.

To classify a new example, a path from the root of the decision tree to a leaf node is traced. At each internal node reached, the branch corresponding to the value of the attribute tested at that node is followed. The class at the leaf node represents the class prediction for that example.

### 2.2.2 The Learning Algorithm

A decision tree is generated or ‘grown’ in stages. The tree starts as a single node, containing all the training examples. ID3 looks to see if all these examples are of the same class. If they are not, it needs to grow branches from that node which will test an attribute of the examples, and sort them into groups corresponding to the different values of that attribute. An attribute is a good test to use if it sorts the examples into classes well, i.e. when most examples with the same attribute value also have the same class value. ID3 uses a function called **entropy** to measure how well an attribute test sorts examples into classes (the lower the entropy, the better the sorting). ID3 places the attribute test yielding minimum entropy at the node being expanded, and attaches branches to the node corresponding to the different values of that attribute.

Each of the leaf nodes in the new tree is now examined. If all the examples at a leaf node are found to be of the same class, then this node is finished, and this class assigned to that node. If there are examples of more than one class, then ID3 must grow the tree further at this node to sort them apart. To do this, the same procedure is used (consider all attribute tests - choose the best, and expand the node by adding branches representing results of this test).

When there are no nodes left to expand, the procedure is finished. Figure 2 summarises this algorithm.

### 2.2.3 Heuristic Functions

The entropy measure which ID3 uses to select the best attribute test is defined as follows:

$$Entropy = \sum_i w_i E_i$$

Figure 2: The ID3 Algorithm

```

let examples = a set of training examples
let atts = the set of all attributes

procedure id3(examples,atts):
if examples are all in a single class c
then return a leaf node labelled 'predict class c'
else for each attribute a in atts
    sort examples according to their values vi of attribute a
    calculate the entropy of the sorted examples
endfor
select abest, the attribute which yielded the lowest entropy
for each value vi of attribute abest
    select examples ei from examples for which abest = vi
    generate subtreei using id3(ei,atts)
endfor
return a node which tests abest and has subtrees subtreei attached

```

where

$w_i$  = the weight of the  $i$ 'th branch  
 (= no. of exs. in branch  $i$  / Total number exs at the parent node)  
 $E_i$  = entropy of the  $i$ 'th branch  
 $= -\sum_j p_j \log_2 p_j$   
 where  $p_j$  = probability of  $j$ 'th class in this branch,  
 estimated from the training data.

For example, consider evaluating the entropy of placing a test on the attribute **furry** at the root of the tree, using the data in Figure 1. If we placed the test, the examples would be sorted at this node into two groups as follows:

**furry = yes:** 3 lions, 2 non-lions  
**furry = no:** 0 lions, 3 non-lions

Hence, to calculate the entropy (a measure of how well the test sorts the examples into classes) for the attribute test **furry**:

$$\begin{aligned}
 Entropy_{LeftBranch} &= -(3/5 \log_2 3/5 + 2/5 \log_2 2/5) \\
 Entropy_{RightBranch} &= -(3/3 \log_2 3/3 + 0/3 \log_2 0/3) \\
 Entropy_{Total} &= 5/8 \times E_{Left} + 3/8 \times E_{Right} = 0.607
 \end{aligned}$$

Similarly, the entropy for tests on attributes **age** and **size** are 0.955 and 0.500 respectively. As a result, **size** is chosen for the test at the node, as it has the lowest entropy.

## 2.2.4 Summary

- A decision tree is produced in stages, each stage being by adding an attribute test and branches to a leaf node in the tree.
- To choose which attribute test to use, consider what the new tree would look like for each possible attribute, and choose the best.
- ‘Best’ is decided by applying the evaluation function **entropy** to each of the possible expansions of the node - this function returns a number which is a measure of how well the attribute test has sorted the examples at the node.
- If all the examples at a leaf node are of the same class, this node doesn’t need to be expanded further. If more than one class, it must be expanded further.
- The procedure continues expanding the unfinished nodes until no more need expanding.

## 2.3 The AQ Algorithm

### 2.3.1 Knowledge Representation

Unlike ID3, the AQ algorithm outputs a set of ‘if...then...’ classification rules rather than a decision tree. This is useful for expert system applications based on the production rule paradigm, and is often a more comprehensible representation than a decision tree especially when the decision tree produced by ID3 is large.

We deviate from the terminology introduced by Michalski [16] in order to maintain continuity with the description of ID3 above. Each decision rule which AQ induces is of the form ‘if <condition> then predict <class>’, where <condition> is a conjunct of attribute tests. There may be more than one rule for each class<sup>1</sup>. For simplicity, we use ‘test’ to refer to an attribute test (e.g. ‘furry = yes’) and ‘conjunct’ to refer to a conjunct of attribute tests. A test or conjunct of tests is said to *cover* an example if all the tests are satisfied by the example.

In AQ, a new example is classified by finding which of the induced rules have their conditions satisfied by the example. If the example satisfies only one rule, then the class predicted by that rule is assigned to the example. If the example satisfies more than one rule, the most common class of training examples that were covered by those rules is predicted. If the example is not covered by any rule, then it is assigned by default to the class that occurred most frequently in the training examples.

### 2.3.2 The Learning Algorithm

AQ generates decision rules for each class in turn. This generation occurs in stages; each stage generates a single rule, and then removes the examples the rule covers from the training set. This step is repeated until enough rules are found to cover all the examples of the chosen class. This whole process is repeated for each class.

To generate a single rule, AQ first selects a ‘seed’ example for the rule to cover and starts with the most general rule ‘if true **then** predict class *c*’ (i.e. ‘all examples are

---

<sup>1</sup>In other descriptions of AQ, the <condition> is sometimes referred to as a *complex* and rules for each class are sometimes combined into a single rule whose <condition> is a disjunct of the complexes, sometimes referred to as a *cover*.

class  $c$ '), where  $c$  is the class of the seed. Specialisations of this rule are then repeatedly generated and explored until a rule which covers only examples of class  $c$  and no examples of other classes has been found. Figure 3 summarises the AQ algorithm.

Several 'best specialisations-so-far' are retained and explored in parallel, thus AQ conducts a variation on a beam search of the space. This set of solutions being explored is called a **star**. The search is shown schematically in Figure 4. Note that AQ is guaranteed to return rules completely consistent with the training data (if such rules exist). This is a desirable property in noiseless domains, but as we examine later is undesirable when there is noise in the training data.

### 2.3.3 Heuristic Functions

AQ uses a heuristic function to decide which <condition>s in the star are 'best'. This function is used to decide

- (i) which <condition>s to throw out of the star should its size exceed the user-defined maximum size
- (ii) which is the best <condition> in the final star to use for a new rule

The particular heuristic function used by the AQ algorithm varies from implementation to implementation. One example is to "sum of positive examples covered and negative examples excluded" and prefer <condition>s with highest score. In the case of a tie for either heuristic, the system prefers <condition>s with fewer attribute tests. Seeds are chosen at random and negative examples are chosen according to their distance from the seed (nearest ones are picked first, where distance is the number of attributes with different values in the seed and negative example).

The AQ algorithm can be efficiently implemented using the Incidence Calculus methods of Bundy [4], whereby a bit-string for each attribute test is constructed, each bit representing a different example ('1' if the test is passed and '0' if it is not). Thus, calculation of the coverage of a conjunct is simply performed by the ANDing of the bit-strings for each test in the conjunct and then summing the '1's in the resultant bit-string.

### 2.3.4 Summary

- AQ induces a set of 'if...then...' rules rather than a decision tree
- Each rule is generated in turn, using a variation of a *beam search* of the space of rules
- A size-limited set called the **star** of best conditions-so-far is maintained during the search for a rule
- The algorithm locates rules which are *completely consistent* with the training data

## 2.4 Current Developments

### 2.4.1 Limitations

There are two classes of limitation with the AQ and ID3 algorithms presented above. Firstly, there are a number of deficiencies with the mechanisms they use which can be overcome by refining the algorithms. Considerable work in the machine learning community has been devoted towards such improvements in recent years, and we review the major developments in this area below. Secondly, there are restrictions imposed not

Figure 3: The AQ Algorithm

```
let examples = a set of training examples
let classes = the set of all classes

procedure aq(examples, classes):
let allrules = {}
for each class c in classes:
    sort examples into pos (members of c) and neg (the rest)
    generate rules by aqrules(pos,neg,c)
    add rules to allrules
endfor
return allrules.

procedure aqrules(pos,neg,c):
let rules = {}
for each member of pos (each 'seed') not covered by any rule in rules:
    call aqrule(seed,neg,c) to generate a rule covering seed
    add rule to rules
endfor
return rules

procedure aqrule(seed,neg,c):
let mgc = the most general <condition> ('true')
let star initially contain only the mgc
for each negative example n in neg:
    for each <condition> c in star:
        if c covers n
        then remove c from star
            & generate all specialisations of c which still cover seed
            but no longer cover n by adding an extra attribute test to c
            ('c' becomes 'c & test' for each test true of seed and false of n)
            & add them all to star.
        if size of star > maxstar (a user-defined constant)
        then remove worst <condition>s in star until size of star = maxstar.
    endfor
endfor
select the best <condition> bestcond in star
return the rule 'if bestcond then predict c'
```



by the particular algorithms themselves but by the whole paradigm of rule induction from examples. These restrictions are caused by the inherent limitation of learning rules with such a simple structure (simply looking for input-output correlations). To move beyond this paradigm a more radical change of approach is required, and we discuss the possibilities and difficulties of creating more powerful learning systems of this form later.

The algorithms described above make several limiting assumptions as follows:

1. There is no noise in the training data
2. The features used to describe examples are *adequate*, i.e. correct classification rules can be formed by boolean combinations of tests on an example's features.
3. The number of training examples is small enough that the algorithms run to completion in acceptable time

These assumptions reduce the applicability of these algorithms to many real-world applications. Indeed, these basic algorithms are rarely used on their own. We now describe some of the recent developments in work to overcome these limitations to widen their applicability.

#### 2.4.2 Noise Tolerance

One of the most serious limitations of the above algorithms is the 'noiseless domain' assumption which they make, namely that any genuine regularity in the data will be perfect (i.e. without counter-examples). Consequently the systems will return only rules which are completely consistent with the training examples. By searching for the most general, consistent rules, these systems locate regularities which involve large numbers of training examples and hence are statistically unlikely to be due to chance choice of the training examples. Instead, they reflect genuine correlations between attributes and classes in the domain and consequently perform well.

However, in most real-world domains genuine correlations are rarely perfect and instead often have a few counter-examples in the training data. Such counter-examples arise due to the presence of noise and an inadequate description language for representing the domain. Searching only the space of consistent rules does not find such regularities and instead more specific rules based on only a few training examples tend to be selected. Although these rules perform perfectly on the training examples, their predictive accuracy on future test examples is often lower because rules formed on the basis of small numbers of examples are susceptible to noise. The small trends they reflect are more likely to be due to chance choice of training set compared with other rules found reflecting major correlations, but rejected due to the presence of counter-examples in the data. As a consequence, the rule set is both large and not of the highest predictive power. This is sometimes referred to as an *overfitting* of the rules to the data [48].

There are several techniques which have been developed for coping with this problem, and we briefly review these here.

##### **Data Filtering**

One technique is to perform induction using only *representative* training examples, as selected by the expert or automatically. This technique was used in AQ11's application to the task of soya bean diagnosis [19], where the ESEL system [21] was first used to select representative training examples.

##### **Pruned general-to-specific search**

One of the most common techniques for filtering out noisy components of data is to

perform the search for a generalised description of observations in a general-to-specific manner, and then to halt the search before complete consistency with the training examples is reached. Such halting occurs when the number of examples supporting the current hypothesis falls too low. The general-to-specific search acts as a ‘covers lots of examples’-to-‘covers few examples’ search, which can be seen as a reliable-to-unreliable search that can be prematurely halted should a level of unreliability be passed.

The ID3 algorithm lends itself to easy modification due to the nature of its general-to-specific search. Tree pruning techniques (e.g. [39, 32]), as used for example in the systems C4 [40] and ASSISTANT [15], have proved to be effective methods of avoiding overfitting. These use statistical methods to assess confidence in the correlations the decision tree branches represent.

The AQ algorithm, however, is less easy to modify due to its dependence on specific training examples during its search. There are two possibilities of remedy here. Firstly, the basic algorithm can be left intact and noisy data can be dealt with by using pre-processing (e.g. data filtering, mentioned above) or post-processing techniques (e.g. post-pruning, see below). Existing implementations (e.g. AQ11 [20] and AQ15 [17]) have adopted this approach. A second approach is to modify the algorithm itself, removing its dependence on specific examples and increasing the space of rules searched. This approach was taken in the CN2 system [10], where instead of conducting a beam search of rules covering a seed but excluding negative examples, the beam search explores (in a general-to-specific manner) the space of *all* rules, and uses a statistical significance test to prevent statistically insignificant specialisations being made. A diagram depicting the CN2 search is shown in Figure 5.

### **Post-Pruning**

An alternative to prematurely halting a general-to-specific search is to allow it to run to completion (i.e. form a rule set completely consistent<sup>2</sup> with observation) then to remove components from (‘post-prune’) the final rules which are deemed unreliable. The advantage of this is that the quality of pruned and unpruned versions of a rule set can be directly compared rather than having to estimate the latter’s quality during search.

Again in rule induction systems this technique is common, for example in the most recent ID3 descendants C4 [40], ASSISTANT-86 [5] and by Niblett and Bratko [33]. Niblett [32] gives a review of pre- and post-pruning techniques used for decision trees. AQ15 [17] employs a post-pruning technique for production rules (termed ‘rule truncation’). The use of post-pruning of decision tree branches to generate production rules has been used by Corlett [11] and Quinlan [37].

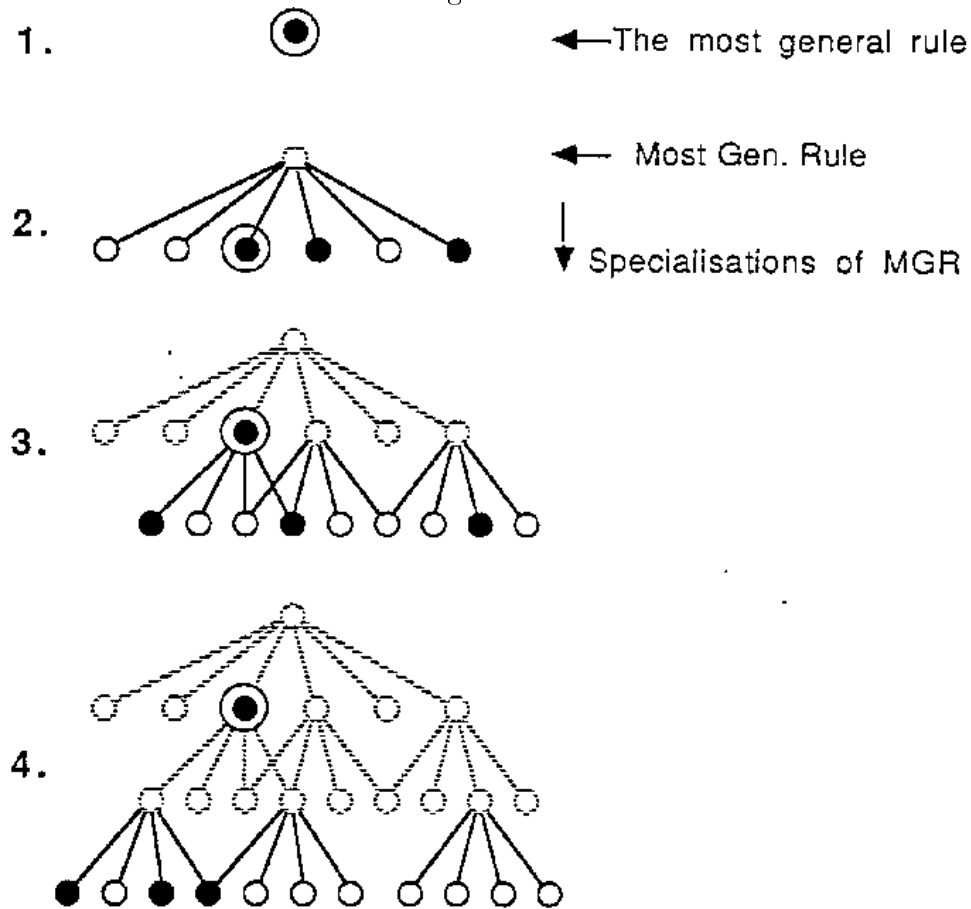
### **Corroborative Rule Application**

Another technique to prevent unreliable rules degrading performance is to allow *all* rules to contribute in some way during classification, with different weights attached to their decisions. (This requires rules which ‘nearly’ fire to also contribute towards the final class prediction). This avoids heavy reliance on a specific, possibly unreliable, part of the system, allowing noisy effects to be over-ridden and smoothed out by other components, but also degrades the comprehensibility of the rule set. Statistical methods such as Bayesian techniques (e.g. [22]) can be employed. AQ15 [17] performs weighted rule application in this way, and Quinlan [35] suggests how decision tree application can be made less brittle by introducing a degree of corroboration between decision tree branches.

---

<sup>2</sup>or as nearly consistent as possible if complete consistency is impossible.

Figure 5:



### The CN2 Search for a rule

A set or 'star' of current best nodes is maintained.

At each step all nodes in the star are specialised.

All specialisations are considered.

A record of the best, statistically significant node is maintained.

On completion of the search, this best node is returned.

- The <n> best new nodes at each stage are stored in the 'beam' (or 'star'), where <n> =< maximum star size (here it's 3). All other nodes are discarded.
- Nodes explored and rejected in the current iteration of search.
- ⊖ Discarded nodes.
- ⊙ The best, statistically significant node found so far.

### 2.4.3 Probabilistic Classification

Another limitation with simple rule induction methods is that no degree of certainty is attached to the classification they give. Degrees of certainty are often required for applications (most expert systems use some measure of uncertainty as part of their problem-solving strategy).

In fact probabilities and measures of certainty are easily attached to classifications, as probabilities can easily be calculated by examining the performance of the rules on the training data. This technique has been demonstrated in recent systems based on ID3 (e.g. [35]) and AQ (e.g. [17]).

### 2.4.4 Large Data Sets

As computers increase their speed and memory capacity, larger and larger data sets can be handled. However, with data sets containing millions of examples there may still be too much data for induction to be feasible within time and memory constraints of application machines.

One technique for handling large data sets is the method of *windowing*. We describe it here applied to ID3 but it can be applied also to other induction algorithms. First, a subset or ‘window’ of the training examples is selected at random (the window size defined by the user) and ID3 generates a decision tree. Then the tree is tested on the whole data set and misclassified examples are used to replace other examples in the window. The examples displaced from the window are chosen as those least essential for the tree; examples from leaves covering a large number of examples are prime candidates, whereas the example from a leaf covering only that one example is important to retain in the window, as it is singularly responsible for the presence of that leaf. ID3 is then re-run on the new window, and the process iterated until the number of classification errors of subsequent trees reaches a minimum. This technique is described in more detail in [36].

### 2.4.5 Incremental Learning

Another constraint sometimes cited against rule induction systems is their non-incremental nature. The algorithms as described above do not permit modification of existing rule sets/decision trees, but require them to be grown again from scratch.

It should be noted that the problem of non-incrementality is one of efficiency rather than of rule quality. Recently, incremental versions of both ID3 and AQ have been proposed which will refine decision trees or rules given new training data rather than re-induce them from scratch. The resulting updated trees or rules are (usually) those which re-induction would have produced anyway, but found more efficiently.

ID3 can be easily modified to act incrementally by storing the counts of examples at each node in the tree, and updating them as new examples arrive. Should a new example ‘tip the balance’ so the existing attribute test at a node is no longer the best, this triggers ID3 to change the attribute test and re-induce the subtrees which are attached to the node. This technique is described and evaluated by Utgoff in the ID5 system [45].

AQ can similarly be modified to act incrementally. Given a new training example, existing rules which are inconsistent with it are deleted and the AQ algorithm re-run to cover all those examples now without rules covering them. This technique is described by Michalski [18] and used, for example, by Mozetic [29].

Table 1: Examples of Binary Numbers and their Associated Parity

No.	Attributes				Parity (= Class Value)
	A1 $2^3$	A2 $2^2$	A3 $2^1$	A4 $2^0$	
9	1	0	0	1	even
7	0	1	1	1	odd
8	1	0	0	0	odd
2	0	0	1	0	odd
5	0	1	0	1	even

### 3 Beyond Rule Induction

#### 3.1 Increasing Representational Power

Algorithms such as ID3 and AQ can be seen as forming ‘theories’ about the world using data with which they have been presented. However, these theories are of a very simple form, with minimal structure. Observations are linked to predictions by a single inference step, and there is no attempt to learn more structured representations explaining observations.

Unfortunately, reasoning in most real-world domains is substantially more complicated than simply applying input-output correlations, as reflected by the degree of structure in most expert systems where many intermediate concepts are used to solve problems. The introduction of intermediate concepts (sometimes referred to as ‘new terms’) and more structured representations can have a profound effect on a learning system’s ability to acquire knowledge. The reason for this can be understood as follows:

1. Generalisation is essential if old information is to be applied to new, previously unseen problems.
2. The process of generalisation involves a search to sift out relevant from irrelevant information. This is a *semantic* aim of generalisation (i.e. an aim concerning the *meaning* of generalisations rather than their particular syntactic form).
3. Generalisation in a learning system is defined in terms of *syntactic* operations (e.g. ‘delete part of the <condition> of this rule’) on a representation.
4. For generalisation to be effective, the representation must be in a form in which the syntactic operations of generalisation can achieve the semantic aims. In other words, we can easily exclude irrelevant information by simple syntactic changes to the representation.
5. The introduction of intermediate terms and structure can dramatically improve this coupling of syntactic operations to semantic form.

A simple example of how introducing terms can profoundly affect learning is that of learning the concept of **parity** from examples, for instance as shown in Table 1. Here, ID3 and AQ will be unable to learn a good classification rule for classifying numbers as being ‘even parity’ or ‘odd parity’, because the generalisation operations of removing attribute tests always remove important information. However by introducing new terms, for example as shown in Table 2, the problem becomes solvable because syntactic operations now allow essential information to be retained. Rules such as ‘if A7=2 then parity=even’ are now present in the search space.

Table 2: Examples of Binary Numbers and their Associated Parity with New Attributes Introduced

No.	Attributes							Parity (= Class Value)
	A1 $2^3$	A2 $2^2$	A3 $2^1$	A4 $2^0$	A5,= A1+A2	A6,= A3+A4	A7,= A5+A6	
9	1	0	0	1	1	1	2	even
7	0	1	1	1	1	2	3	odd
8	1	0	0	0	1	0	1	odd
2	0	0	1	0	0	1	1	odd
5	0	1	0	1	1	1	2	even

Thus, the role of a representational scheme in a machine learning system is a dual one:

- As ‘program’ to perform the application task for which the system has been designed
- As ‘data’ to be manipulated by the learning component of the system

In order to learn effectively, there must be correspondence between these two roles. The represented knowledge must interact appropriately not only with the inference machinery during a performance task but also with the machinery for learning. Introducing intermediate terms and additional structure can greatly assist in creating a good correspondence.

However, allowing a system greater representational power immediately results in a vastly expanded search of possible ‘theories’ which can account for observations. Controlling this search is a major problem. Some of the most common methods which are used for controlling the search are as follows:

1. Use of representations at **multiple levels of abstraction** (e.g. [29, 46]). Here a representation at a coarse level of detail is first acquired, then a more detailed representation is formed. The abstract (coarse) level serves to constrain search at the more detailed level.
2. Use of an **oracle** (e.g. [31, 30]). Here, the user is asked to verify the system’s operation at each step.
3. Assuming **no noise** (e.g. [43]). By making this assumption, any learned knowledge for which counterexamples exist can immediately be rejected, greatly reducing the search space.

### 3.2 Constructive Induction

We have argued that more structured representations are to be aimed for in order to allow the system to acquire more complex knowledge. Many learning systems (e.g. those just cited) can make intermediate inferences as well as the restricted ‘if <inputs> then <output>’ inferences permitted by ID3 and AQ, but the majority are constrained to use only concepts and terms supplied by the user.

To go beyond this, ideally the learning system itself should automatically introduce ‘new’ concepts, as functions of already known concepts, if it would prove beneficial to structuring knowledge. This automatic introduction of terms is sometimes referred to as ‘constructive induction’. An analysis of the importance of constructive induction in

the learning of certain classes of concepts is made by Rendell [42]. The importance of structuring problems in this way has also been investigated by Shapiro and Niblett [44]. Research in this area is fairly new; the system CIGOL [31] is one example of recent work, using the principle of inverse resolution to introduce new terms in logic programs.

### 3.3 Using Additional Domain Knowledge

Another limitation of inductive approaches as used in the ID3 and AQ algorithms is that no additional domain-specific knowledge is employed to control search beyond that of the training examples themselves. Although additional domain knowledge has been applied manually by the expert before the induction program is run (in the form of his or her selection of appropriate attributes and classes for the system), this extra knowledge is not available for the system to use during learning.

The class of problem which ID3 and AQ deal with – namely those with a large number of training examples and where no additional domain knowledge is available – is perhaps not typical of the majority of real-world applications. More commonly, problems are characterised by the availability of

- only a small number of examples of problem solving, too few to perform reliable induction with on their own
- a large amount of available domain knowledge, but not precise or certain enough to reliably solve problems with alone

In this case, a system using both previous examples and available domain knowledge is required, a hybrid of ‘pure’ expert system and rule induction methods. This raises complex issues of both how to represent uncertain domain knowledge and how to integrate and modify it as new evidence appears.

### 3.4 Case-Based Reasoning

One approach to integrating the use of domain knowledge and examples is that of ‘case-based’ or ‘exemplar-based’ reasoning. Case-based approaches are characterised by retaining examples of previously solved problems in memory, and then new problems solved by first retrieving a solution to a similar old problem and secondly modifying it to solve the new problem. In this way, old solutions act as ‘anchor points’, providing islands of reliable information, and domain knowledge has a reduced role of relating a new problem to an old problem rather than solving the entire new problem from scratch. This technique allows uncertain domain knowledge to be used in conjunction with examples to solve problems, and the accumulation of new solutions over time results in improved performance. Additionally, this technique models the behaviour of much human problem solving and thus contributes towards its psychological validity and comprehensibility by an expert. Examples of recent work in case-based reasoning are Chef [12], by Clark [8, 9], Protos [2] and Nexus [3] (also see [1]). A comparison of this approach with rule induction approaches is given in [7]. While case-based approaches are rising in popularity, it should be noted that many remain rather ad hoc with little theoretical underpinning.

### 3.5 Knowledge Compilation

In addition to learning ‘new’ knowledge (knowledge which does not follow deductively from what is already known), some work in machine learning has been devoted to making

existing knowledge more efficient. The area of explanation-based learning (EBL) covers a variety of techniques, most working with some first order logic variant such as Horn clauses or production rules as a representational language. The most common technique, explanation-based generalisation (EBG), involves generating and storing a general solution to a problem [27, 14]. Solving a problem often involves instantiating and applying a number of operators or ‘rules’ – explanation-based ‘generalisation’ (‘re-generalisation’ is perhaps more appropriate) collects and simplifies the uninstantiated operator sequence, storing it for later use. EBG can in fact be viewed as selectively applying the logic programming method of partial evaluation [47].

EBG improves efficiency when the cost of generating, storing and retrieving these learned solution sequences is outweighed by the improved speed they give (a learned solution sequence no longer needs to be recalculated and instead can be immediately applied). Early work assumed generalising solutions to old problems would yield such an overall benefit (thus assuming new problems will be similar to old problems). However, this is not necessarily the case – Minton [26, 25] has recently conducted more detailed analyses of the criteria for deciding which generalisations to store in order to achieve an increase in the system’s performance.

## 4 Conclusion

Machine learning is becoming an increasingly important area of study within the field of artificial intelligence. Despite its simplicity, the technology of rule induction from examples is rapidly becoming an important area of application of machine learning and we have described two systems for this task in detail in this paper. To progress beyond rule induction to systems capable of learning more complex knowledge about the world, there are still major problems of search, knowledge representation and knowledge integration which must be addressed and we have surveyed some of the recent work in this area targeted at these problems.

## References

- [1] *Proc. First Case-Based Reasoning Workshop* Ca, Kaufmann.
- [2] Bareiss, E. R., Porter, B. W., and Wier, C. C. (1987). PROTOS: an exemplar-based learning apprentice. In *Proc. 4th International Workshop on Machine Learning* Ca, P. Langley, Ed., Kaufmann.
- [3] Bradshaw, G. (1987). Learning about speech sounds: the nexus project. In *Proc. 4th International Workshop on Machine Learning* Ca, P. Langley, Ed., Kaufmann, pp. 1–11.
- [4] Bundy, A. (1985). Incidence calculus: a mechanism for probabilistic reasoning. *Journal of Automated Reasoning* 1, 3, 263–283.
- [5] Cestnik, B., Kononenko, I., and Bratko, I. (1987). Assistant 86: a knowledge-elicitation tool for sophisticated users. In *Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning)*, I. Bratko and N. Lavrač, Eds., Sigma, Wilmslow, UK, pp. 31–45.
- [6] Clancey, W. J. (1984). Classification problem solving. In *AAAI-84*, pp. 49–55.
- [7] Clark, P. (1988). A comparison of rule and exemplar-based learning systems. In *International workshop on Machine Learning, Meta-reasoning and Logics* Portugal, Portugal, Ed., Faculdade de Economia, Univ. Porto, pp. 69–81. (Proceedings to be published in book form in 1989).
- [8] Clark, P. (1988). *Exemplar-based Reasoning in Geological Prospect Appraisal*. TIRM 034, Turing Institute, Glasgow, UK.
- [9] Clark, P. (1988). Representing arguments as background knowledge for constraining generalisation. In *Proc. Third European Working Session on Learning (EWSL-88)* London, D. Sleeman, Ed., Pitman, pp. 37–44.

- [10] Clark, P., and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning Journal* 3, 261–283.
- [11] Corlett, R. A. (1983). Explaining induced decision trees. In *Expert Systems 83*, pp. 136–142.
- [12] Hammond, K. (1986). CHEF: a model of case-based planning. In *AAAI-86*.
- [13] Hunt, E. B., Marin, J., and Stone, P. T. (1966). *Experiments in Induction*. Academic Press, NY.
- [14] Kedar-Cabelli, S. T., and McCarty, L. T. (1987). Explanation-based generalization as resolution theorem proving. In *Proc. 4th International Workshop on Machine Learning* Ca, P. Langley, Ed., Kaufmann, pp. 383–389.
- [15] Kononenko, I., Bratko, I., and Roskar, E. (1984). *Experiments in Automatic Learning of Medical Diagnostic Rules*. Technical Report, Faculty of Electrical Engineering, E. Kardelj University, Ljubljana.
- [16] Michalski, R. (1983). A theory and methodology of inductive learning. In *Machine Learning, vol. 1*, J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, Eds., Tioga, Palo Alto, Ca, pp. 83–134.
- [17] Michalski, R., Mozetic, I., Hong, J., and Lavrac, N. (1986). The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *AAAI-86* Ca, Kaufmann, pp. 1041–1045.
- [18] Michalski, R. S. (1985). *Knowledge repair mechanisms: evolution vs. revolution*. ISG 85-15, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science, Urbana.
- [19] Michalski, R. S., and Chilausky, R. (1980). Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean diagnosis. *Policy Analysis and Information Systems* 4, 2, 125–160.
- [20] Michalski, R. S., and Larson, J. (1983). *Incremental Generation of  $VL_1$  Hypotheses: the underlying Methodology and the Description of Program AQ11*. ISG 83-5, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana.
- [21] Michalski, R. S., and Larson, J. (1978). *Selection of most Representative Training Examples and Incremental Generation of  $VL_1$  Hypotheses: the underlying Methodology and the Description of Programs ESEL and AQ11*. UIUCDCS-R 78-867, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana.
- [22] Michie, D. (1982). Bayes, turing and the logic of corroboration. In *Machine intelligence and related topics: an information scientist's weekend book*, Gordon & Breach, NY.
- [23] Michie, D. (1988). Machine learning in the next five years. In *Proc. Third European Working Session on Learning (EWSL-88)* London, D. Sleeman, Ed., Pitman, pp. 107–122.
- [24] Michie, D. (1986). The superarticulacy phenomenon in the context of software manufacture. In *Proceedings of the Royal Society (Series A)*, pp. 185–212. (Also available as internal report TIRM-85-13, Turing Institute, Glasgow, UK).
- [25] Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *AAAI-88* Ca, Kaufman, pp. 564–569.
- [26] Minton, S. (1985). Selectively generalizing plans for problem-solving. In *IJCAI-85*, pp. 596–599.
- [27] Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: a unifying view. *Machine Learning Journal* 1, 1, 47–80.
- [28] Mowforth, P. (1986). *Some Applications with Inductive Expert System Shells*. TIOP 86-002, Turing Institute, Glasgow, UK.
- [29] Mozetic, I. (1987). The role of abstractions in learning qualitative models. In *Proc. 4th International Workshop on Machine Learning* Ca, P. Langley, Ed., Kaufmann.
- [30] Muggleton, S. (1987). Duce: an oracle-based approach to constructive induction. In *IJCAI-87* Ca, J. McDermott, Ed., Kaufmann, pp. 287–292.
- [31] Muggleton, S., and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proc. 5th Int. Conf. on Machine Learning* Ca, J. Laird, Ed., Kaufmann, pp. 339–352.
- [32] Niblett, T. (1987). Constructing decision trees in noisy domains. In *Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning)*, I. Bratko and N. Lavrač, Eds., Sigma, Wilmslow, UK, pp. 67–78.

- [33] Niblett, T., and Bratko, I. (1986). Learning decision rules in noisy domains. In *Expert Systems 86, Brighton, UK*.
- [34] Paterson, A., and Niblett, T. *ACLS Manual. Version 1*. Glasgow, 1982.
- [35] Quinlan, J. R. (1987). Decision trees as probabilistic classifiers. In *Proc. 4th International Workshop on Machine Learning* Ca, P. Langley, Ed., Kaufmann.
- [36] Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In *Expert Systems in the Micro-Electronic Age*, D. Michie, Ed., Edinburgh Univ. Press, UK, pp. 168–201.
- [37] Quinlan, J. R. (1987). Generating production rules from decision trees. In *IJCAI-87* Ca, J. McDermott, Ed., Kaufmann, pp. 304–307.
- [38] Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames. In *Machine Learning, vol. 1*, J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, Eds., Tioga, Palo Alto, Ca.
- [39] Quinlan, J. R. (1987). Simplifying decision trees. *Int. Journal of Man-Machine Studies* 27, 3, 221–234.
- [40] Quinlan, J. R., Compton, P. J., Horn, K. A., and Lazarus, L. (1987). Inductive knowledge acquisition: a case study. In *Applications of Expert Systems*, Addison-Wesley, Wokingham, UK, pp. 157–173.
- [41] Reddy, R. (1988). Foundations and grand challenges of artificial intelligence. *AI Magazine* 9, 4, 9–21.
- [42] Rendell, L. (1988). Learning hard concepts. In *Proc. Third European Working Session on Learning (EWSL-88)* London, D. Sleeman, Ed., Pitman, pp. 177–200.
- [43] Sammut, C., and Banerji, R. (1986). Learning concepts by asking questions. In *Machine Learning, vol. 2*, J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, Eds., Tioga, Palo Alto, Ca.
- [44] Shapiro, A. D. (1987). *Structured Induction in Expert Systems*. Turing Inst. Press, in association with Addison-Wesley, Wokingham, UK.
- [45] Utgoff, P. E. (1988). Id5: an incremental id3. In *Proc. 5th Int. Conf. on Machine Learning* Ca, J. Laird, Ed., Kaufmann, pp. 107–120.
- [46] Van de Velde, W. (1988). Learning through progressive refinement. In *Proc. Third European Working Session on Learning (EWSL-88)* London, D. Sleeman, Ed., Pitman, pp. 211–226.
- [47] VanHarmelen, F., and Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *Artificial Intelligence* 36, 3, 401–412.
- [48] Watkins, C. J. C. H. (1987). Combining cross-validation and search. In *Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning)*, I. Bratko and N. Lavrač, Eds., Sigma, Wilmslow, UK, pp. 79–87.