A Note on the Utility of Incremental Learning

Christophe Giraud-Carrier
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
Email: cgc@cs.bris.ac.uk

Historically, inductive machine learning has focused on non-incremental learning tasks, i.e., where the training set can be constructed a priori and learning stops once this set has been duly processed. There are, however, a number of areas, such as agents, where learning tasks are incremental. This paper defines the notion of incrementality for learning tasks and algorithms. It then provides some motivation for incremental learning and argues in favour of the design of incremental learning algorithms for solving incremental learning tasks. A number of issues raised by such systems are outlined and the incremental learner ILA is used for illustration.

Keywords: incrementality, inductive learning

1. Introduction

Machine learning is concerned with the design of algorithms which, rather than encoding explicit instructions or programs for the solution of specific problems, encode inductive mechanisms whereby solutions to broad classes of problems may be derived from examples. This learning approach is clearly more flexible than the programming approach and significantly widens the range of interesting problems that computers may help to solve.

The traditional formulation of the machine learning problem has been as a classification problem (e.g., see [22,23]). Informally, one is given some domain of individuals for which a general classification is required. The classification is given by a function from this domain to some small finite set corresponding to the classes. Examples are available as a training set which provides the class of some typical individuals. In addition, some background knowledge relevant to the inductive task at hand may be available. From this, the general classification of the individuals must be induced.

Although not explicit in this formulation, there has often been an implicit assumption that the training set is available a priori and that learning ceases once this set has been duly processed. Henceforth, the induced classification is used exclusively to make predictions about new instances. Despite its applicability to a wide range of problems (e.g., mine-rock discrimination, loan underwriting, medical diagnosis), this approach to learning is clearly not all-encompassing. There are a number of interesting situations where learning must take place over time, in a kind of continuous fashion rather than as a one-shot experience.

At the heart of the distinction between these forms of learning is the notion of incrementality. For historical reasons, incremental learning has been largely forgotten in machine learning research [18]. This paper is an attempt at re-instating incrementality, by showing its utility in widening the scope of applicability of machine learning, especially in areas of emerging technologies, such as agents. The distinction between incremental learning tasks and incremental learning algorithms is clarified. It is argued that incrementality is rather ubiquitous in learning and that the most natural and flexible way to tackle incremental learning tasks is with incremental learning algorithms. A number of key design issues for incremental algorithms are also discussed along with the description of an illustrative incremental learning system.

2. Definitions

The term incremental has been applied both to learning tasks and to learning algorithms, thus leading to some confusion. This section attempts to elucidate this confusion by providing formal definitions and examples of incrementality for tasks and for algorithms.

2.1. Incremental Learning Tasks

The following characterises the notion of incrementality as it applies to learning tasks.

Definition 1 *A learning task is incremental if the training examples used to solve it become available over time, usually one at a time.*

Note that, if one is prepared to wait long enough, any incremental learning task can, in principle, become a non-incremental one.[1] Hence, for incremental learning tasks, there is an implicit assumption that waiting is undesirable and/or impractical. In particular, the nature of the application may render unfeasible the timely generation of a sufficiently large number of representative examples, either because the environment changes in time (and thus learning becomes situated or context-sensitive) or because the rate at which examples become available may be too slow.

The following are examples of incremental learning tasks. In the first three, incrementality is due to changes in the target over time. In the last one, the target does not change, but the acquisition of data is untimely.

- *User modelling/profiling*. With humans, today's behaviour is not necessarily a good indication of tomorrow's. Experts argue that users' behaviour and interests vary radically within as little as six months. Hence, the useful task of learning user profiles requires on-line monitoring of individual users.
- *Robotics*. In all but the simplest of cases, a robot's environment is changing and often unpredictable. Hence, in order to survive (e.g., by negotiating collision-free navigation) and to carry out its tasks successfully, a robot must be able to react and adapt incrementally to environmental cues.
- *Intelligent agents*. Agents are software implements characterised by both reactiveness and proactiveness. Hence, as with robots, incrementality is inherent in agent learning. Traditional agent applications where learning is useful include network management (e.g., load balancing, routing) and intelligent user interfaces.
- *Software project estimation*. Estimating the cost, effort and duration of software projects is largely a matter of experience. Due to the rather long timescales of such projects however, useful data becomes available in a piecemeal way, over time. It has been argued that the construction of an adequate baseline for estimation may take up to three years. Yet, even limited experience is better than no experience at all in improving the accuracy of estimates.

In contrast, classification problems (e.g., discriminating between rocks and mines from a sample of labelled sonar readings) are standard examples of non-incremental tasks since all training data is usually available a priori.

[1]There is one notable exception. If examples are drawn from a dynamically changing environment, one may have to wait forever before learning could start!

The main characteristics of an incremental learning task are:

- Examples are not available a priori but become available over time, usually one at a time.
- Learning may need to go on (almost) indefinitely.

2.2. Incremental Learning Algorithms

The following characterises the notion of incrementality as it applies to learning algorithms.

Definition 2 *A learning algorithm is incremental if, for any given training sample $e_1, \ldots, e_n$, it produces a sequence of hypotheses $h_0, h_1, \ldots, h_n$, such that h_{i+1} depends only on h_i and the current example e_i.*

Incremental learning algorithms are also called memoryless on-line algorithms (e.g., see [1]). For consistency with learning tasks, the term incremental is preferred here.

The following is a small sample of incremental learning algorithms.

- CANDIDATE-ELIMINATION [23,24]. This algorithm induces binary classifications. Given representation languages for examples and generalisations, both the set S of maximally specific generalisations and the set G of maximally general generalisations consistent with the training data are stored. G is initialised to the most general concept in the space and S to the first positive training instance. S keeps generalising to cover new positive training instances and G keeps specialising to avoid covering negative training instances. Changes to S and G are effected as training instances are presented one at a time.
- COBWEB [11]. This algorithm induces taxonomies or categorisations. The number of clusters, depth of the hierarchy and category memberships are determined by a global probabilistic metric, called category utility. Any time a new training instance is presented, the algorithm considers the overall quality of either placing it in an existing category or modifying the current hierarchy to accommodate it (e.g., create a new category, merge categories). Only probabilities are stored and updated.

– ID5 [32]. This algorithm induces decision trees. It is an incremental version of ID3 [27]. Instead of building a tree from a batch of training examples, it incrementally updates a tree after each training instance is presented. All transformations are performed by simply maintaining appropriate counters at each node.
– ILA [16]. This algorithm induces classifications. It is described in some detail in section 5.

Note that, if one relaxes slightly the above definition to allow the next hypothesis to depend on the previous one and a small subset of new training examples (rather than a single one), then an iterative implementation of the bayesian framework, where the previous iteration's posterior is the current iteration's prior, would also be suited to incremental learning.

In contrast with the above, most well-known learning algorithms, such as ID3 [27], CN2 [5] and backpropagation [28], are not incremental. Essentially, they assume that the training set can be constructed a priori, that examples may be processed more than once and that learning stops once the training set has been duly processed.

The main characteristics of an incremental learning algorithm are:

– No re-processing of previous examples is necessary.
– Since each h_i can be viewed as a best approximation so far of the target application, the learner can, at any time, produce an answer to a query and the quality of its answers improves over time.[2]

Clearly, non-incremental learning algorithms can be used to solve incremental learning tasks. For example, the ID3 and backpropagation algorithms can learn by processing one example at a time. However, all previously encountered examples must remain available explicitly to enable learning – in ID3 to compute entropies and in backpropagation to compute mean sum-squared errors. In other words, learning incremental tasks with non-incremental algorithms requires complete re-training after each example. This issue is revisited in section 3.2.

3. Motivation

This section outlines some of the motivation for incremental learning. First, aspects of human learning are highlighted, which show that incrementality is ubiquitous in general learning. Then, a short discussion of applicability versus utility provides the motivation for the development of incremental learning algorithms.

[2]This property is also characteristic of anytime algorithms [9], except in learning the input determines the process.

3.1. The Ubiquity of Incremental Tasks

It has long been argued that self-adaptation is a prerequisite for general intelligence [25] and that learning, in particular, involves the ability to improve performance over time [29]. Clearly, humans acquire knowledge over time, i.e., incrementally, since all of the information necessary to learn many concepts is rarely available a priori. Rather, new pieces of information become available over time, and knowledge is constantly revised (i.e., evolves) based on newly acquired information. Most of the applications described in section 2, such as those involving intelligent agents and robots, require self-adaptation.

Although inductive mechanisms play a crucial role in human learning, they are by no means exclusive. It is quite clear that commonsense, built-in mechanisms (e.g., pain) and engagement in complex social interactions (e.g., families, schools, work places) also provide means of knowledge acquisition/exchange through rote learning [30]. Hence, humans need not always "reinvent the wheel", but may learn from both rules given by a "teacher" (i.e., rote learning) and rules obtained from examples (i.e., inductive learning). Consider the following illustrative examples.

– Computer programming is largely learned by being taught and further enhanced through experience.
– Foreign languages are acquired through a combination of teaching and learning. For example, grammar rules are often taught explicitly (e.g., in English, the plural of a noun is usually formed by tagging an s onto the end of the singular form of that noun) whilst exceptions are discovered through experience/mistake (e.g., geese instead of gooses, mice instead of mouses).
– Children, through a built-in pain mechanism and precepts taught by others (e.g.,"it will burn you"), learn (almost instantaneously) not to put their hand in the fire or on a hot stove.

The combination of rote and inductive learning has been recognised as critical in the design of practical embedded (autonomous) agents [17,2,3]. In particular, it allows the direct exchange of theories between

		Learner	
		Incr.	**Non-incr.**
Task	**Incr.**	Yes	Yes
	Non-incr.	Yes	Yes

Fig. 1. Applicability Matrix

		Learner	
		Incr.	**Non-incr.**
Task	**Incr.**	High	Low
	Non-incr.	Low	High

Fig. 2. Utility Matrix

agents in multi-agent learning systems. Furthermore, empirical evidence suggests that the ability to use rules and examples increases learning speed by pruning and constraining the search for generalisations in the input space, reduces memory requirements and improves overall predictive accuracy (e.g., see [14]). Although in some cases, such as theory revision/refinement (e.g., see [13,26,31]), all of the "teachable" knowledge may be available a priori, most learning tasks using both rote learning and induction are inherently incremental.

Finally, empirical evidence suggests that in some cases, learning is only possible when data is presented incrementally. For example, in the context of learning grammar with a recurrent network, it has been shown that the "network fails to learn the task when the entire data set is presented all at once, but succeeds when the data are presented incrementally [with an easy to hard ordering]" [10].

3.2. The Utility of Incremental Learners

From the definitions of section 2, it is clear that, in principle, the matrix of applicability of learning algorithms to learning tasks, from the perspective of incrementality, is full (see Fig. 1). On the other hand, it seems also rather clear that, in practice, the matrix of utility has high values in the main diagonal only (see Fig. 2).

The low utility of incremental learners for non-incremental tasks derives from the fact that incremental learners can only make use of information present in the current hypothesis and example. If all the examples are available a priori, then useful information may be unnecessarily ignored. In such cases, incremental learners are myopic. They induce knowledge from local information only (i.e., current hypothesis and example) when global information (i.e., all training data) is actually available.

The low utility of non-incremental learners for incremental tasks derives from the additional computational complexity, both in terms of memory requirement and time, caused by the storage of past examples and the need for re-training. Clearly, some incremental learning algorithms can also have high complexity. For example, the sets S and G, stored by the CANDIDATE-ELIMINATION algorithm [24], may have exponential size, thus requiring exponential storage and computation. However, for a fixed class of learning algorithms (e.g., decision tree learners), some general observations can be made as follows.

Let $S = (e_1, \ldots, e_i)$ be a sequence of training examples for some incremental task T. Let h_i^{non} and h_i^{inc} be the hypotheses obtained from S by some non-incremental (resp., incremental) processes, P^{non} and P^{inc}. In addition to storing h_i^{non}, P^{non} must also store S since T is an incremental task and further training examples may become available. Hence, the storage requirement of P^{non} is $\mid h_i^{non} \mid + \mid S \mid$. On the other hand, the storage requirement for P^{inc} is $\mid h_i^{inc} \mid$. Because of the myopic effect mentioned above and the inherent expectation that new examples may be encountered later (see also section 4.1), P^{inc} often incurs some storage overhead, so that in general, $\mid h_i^{inc} \mid > \mid h_i^{non} \mid$.[3] On the other hand, it is also reasonable to expect that the size of the overhead (i.e., $\mid h_i^{inc} \mid - \mid h_i^{non} \mid$) is less than that of S, especially as i gets larger. Hence, P^{inc} requires less storage space than P^{non}. Similarly, it can be expected that the computational cost of constructing h_{i+1}^{non} by re-training P^{non} on $S \cup \{e_{i+1}\}$ is larger than the cost of constructing h_{i+1}^{inc} using P^{inc}. For example, it has been shown that, given I training instances and A attributes, the total number of attributes that ID5 must examine is O($I.A^2$) against O($I^2.A^2$) for ID3 with re-training after each instance [32].

It follows from the above that, although all learning algorithms are applicable to all learning tasks, the most natural and flexible way to handle incremental learning tasks is with incremental learners.

[3]Since P^{non} assumes that S is "complete", such overhead is unnecessary.

4. Design Issues

This section highlights some of the main issues raised in the design of incremental learners. A number of these are also found in [8].

4.1. Ordering Effects

Chronology, or the order in which knowledge is acquired, is an inherent aspect of incrementality. Thus, it is possible to integrate time implicitly as a factor and another source of bias in learning. For example, the system could choose to give precedence to newly acquired knowledge, recognising the possibility that early guesses may be incorrect (e.g., see [21]). Alternatively, a learning system's experience with the world could be appropriately ordered (e.g., general rules before exceptions) so as to increase its efficiency (see section 2.1).

Although chronology may be used as a bias, it presupposes that the ordering of the data carries some meaning that should be implicitly captured by the learning system. Such is not always the case. Techniques have been proposed to deal with ordering effects in incremental systems (e.g., [6,7,12,20]), but the question of whether order independence can be achieved (if desired) remains largely open. One can argue that for large (and rich) enough training sets, the effects of ordering become more limited, as the current representation of the system eventually matches reality. In other words, global statistical patterns ultimately override errors or misrepresentation due to local patterns of noise, lack of information, or plain errors. Indeed, incremental learning presupposes redundancy in the data, as is true in human learning. Similar situations have a tendency to reproduce themselves so that general rules (or analogies) can ultimately be drawn from them.

4.2. Learning Curve

An incremental system may start from scratch and gain knowledge from examples given one at a time over time. As a result, the system experiences a sort of learning curve, where the quality of its predictions improves (possibly slowly) over time. Inherent in this is the fact that the system is not very trustworthy early on. Hence, although the system can make predictions at any time, one must be cautious with how much value is placed on them. Furthermore, it is difficult to determine the point at which the system has learned "enough" to be trusted.

4.3. Open-world Assumption

Informally, non-incremental learning assumes that the world is closed. That is, the system essentially works under the premise that its experience of the world, i.e., its training set, *is* the world. Although this can be theoretically convenient, it certainly does not hold in everyday life, where information, however much of it is available, is generally uncertain and incomplete.

Consider, for example, the complementary situations of having to accommodate exceptions and recovering from them, as illustrated in the case of learning whether birds fly.

– *Accommodating Exceptions*. Suppose that the training sample is made out of common European birds. Then, birds will be predicted to fly and, if the world is assumed closed, this will apply to all birds, even the penguins and ostriches the system may later encounter.
– *Recovering from Exceptions*. Suppose that the training sample is rather atypical, consisting of birds from Australia and the South Pole (i.e., mostly ostriches and penguins). Then, birds will be predicted not to fly and, if the world is assumed closed, this will apply again to all birds.

In both situations, further experience must be allowed to alter the the truth of previously accepted facts, if a solution consistent with the *real* world is to be found. Humans use commonsense reasoning to deal with their partial representation of the open world (e.g., see [4, 19]).

The need for an open-world assumption is clearly a consequence of incrementality. If all the data relevant to the problem at hand is indeed available a priori, then the world may be assumed closed. Otherwise, there is a need for special learning mechanisms that invalidate portions of knowledge, while not affecting the rest of it.

In order to deal with ordering effects and the open-world assumption incremental learners often need to store more information than their non-incremental counterparts, as alluded to in section 3.2.

5. Implementation: An Example

To illustrate how the aforementioned issues may be addressed in a learning system, this section briefly describes ILA [16].

ILA implements its knowledge base in the nodes of a network that is a balanced binary tree. Training data are presented incrementally and the system adapts by dynamically adding nodes to the network. The final network is thus the result of a sequence of self-organising transformations. All computations require only simple broadcast and gather mechanisms. Hence, ILA makes use of inherent parallelism in execution, and exhibits low-order polynomial complexity. The choice of a binary tree architecture is historical and only useful for the efficient identification of best matches (see below).

ILA's representation language is an instance of the attribute-value language (AVL). In ILA, attributes may range over nominal domains and bounded linear domains, including closed intervals of continuous numeric values. The basic elements of knowledge in AVL are vectors defined over the cross-product of the domains of the attributes. The components of a vector specify a value for each attribute. Since ILA induces classifications, one attribute is designated as the target or class attribute. The following simple extension is made to AVL. If A is an attribute, other than the target one, and D is the domain of A, then A takes on values from $D \cup \{\star, ?\}$. The special symbols $\star$ and ? stand for *don't-care* and *don't-know*, respectively. The semantics associated with $\star$ and ? are different. An attribute whose value is $\star$ is one that is known (or assumed) to be irrelevant in the current context, while an attribute whose value is ? may be relevant but its actual value is currently unknown. Hence, the $\star$ symbol allows the encoding of rules, while the ? symbol accounts for missing attribute values in real-world observations.

Let x and y be two vectors. The following definitions are necessary to describe ILA's learning algorithm.

- x is a $rule$ if it contains attributes whose values are $\star$, otherwise x is an $example$. Rules can be given explicitly by a teacher or induced from examples by dropping conditions [22].
- x and y are $concordant$ if and only if they have the same class value.
- x and y are $near\_match$ if and only if they differ in the value of exactly one non-$\star$ attribute (class excluded).
- x is a $subset$ of y if and only if all non-$\star$ attributes of x have identical values in x and y (class excluded).[4]

[4]Linear values are deemed identical if they are close to each other (see [16] for details).

```
Broadcast($y$)
Gather(($D(x, y)$, MIN), (num_asserted, MAX),
        (priority, MAX), (num_covers, MAX))
```

Fig. 4. Execution Phase

- x $covers$ y if and only if x and y are concordant, and y is a subset of x.
- $num\_asserted(x)$ is the number of non-$\star$ values in x.

Each node stores a vector, together with the vector's priority value (pty), counters for each class value (counters), and a counter for vectors covered (num_covers). A node's priority is a static value with default 0. These static priorities are useful for dealing intensionally with conflicting rules. A node's counters are used for noise handling. Exactly one counter value is incremented each time an incoming training vector is equal to the node's stored vector. The value incremented corresponds to the training vector's class. The counter value that is highest represents the "most probable" class for the node's stored vector, and can thus be used as the node's output in execution. A node's num_covers value corresponds to the number of training vectors (examples and rules) that the node covers. It can be regarded as a kind of dynamic priority value for dealing extensionally with conflicting rules, and is a measure of the level of confidence in the node's stored vector.

A node's activation is the "distance" of its stored vector, x, to the vector, y, it is presented. The distance is computed from x to y as shown in Fig. 3. D is an extension of the heterogeneous distance measure proposed in [15]. Although D does not define a metric over the input space, it accounts naturally for the presence of don't-cares and missing values.

ILA's learning consists of two phases: execution and adaptation. In the execution phase, the training vector is presented to the network and a winning node is identified as shown in Fig. 4. The training vector is denoted by y and the pairs stored in nodes by x. The details of the Gather() function are given in [16]. Essentially, it consists of having each node compute its activation, and as values are passed up the tree to the parent, identifying a local winner by first minimising the activation (i.e., node whose stored vector is closest, or most similar wins), then resolving ties by giving priority to the more specific node (i.e., larger value of num_asserted), then to the node with higher static priority, and finally to the node that has been matched most often so far

$$D(x,y) = \frac{\displaystyle\sum_{i \neq target} \begin{cases} x_i \neq y_i & \text{if } (x_i, y_i \neq \star, ?) \wedge (\text{ attribute } i \text{ is nominal}) \\ \dfrac{|x_i - y_i|}{range(i)} & \text{if } (x_i, y_i \neq \star, ?) \wedge (\text{ attribute } i \text{ is linear}) \\ 0.5 & \text{if } (x_i = ?) \vee [(x_i \neq *) \wedge ((y_i = *) \vee (y_i = ?))] \\ 0 & \text{otherwise} \end{cases}}{num\_asserted(x)}$$

where $range(i)$ is the range of values of attribute i.

Fig. 3. ILA's Distance Measure

(i.e., larger value of num_covers). Each node is tagged as SELF, RIGHT or LEFT, based on which node wins the local competition (i.e., itself or one of its children). An overall winning node is then easily identified. Giving priority to more specific nodes helps in handling exceptions (see section 4.3). Giving priority to nodes with higher static priority values accounts for certain conflicting rules given a priori. Finally, giving priority to nodes whose num_covers value is greater increases the chances of selecting a "most likely to be correct" winner.

The output of the winning node is the system's output and its current best guess at predicting the outcome of the incoming vector. Hence, the system thus produces an answer at any time (see section 2). Clearly, predictions in the early stages of learning may not be very accurate. However, since the execution phase is followed by an adaptation phase, as more examples are presented, the system's predictions improve (see section 4.2).

Following the execution phase, the network undergoes adaptation. The knowledge base/network is updated based on the relationship between the training vector and its best match. If the network is empty, then there is no winning node and a new node is automatically added for the training vector. In all other cases, the algorithm in Fig. 5 is executed. The winning node is denoted by W, the vector stored in W by x and the training vector by y. As nodes are added to the network, the binary tree structure is kept balanced. Generalisation consists in setting the value on which the two vectors disagree to $\star$. The various conditions on when to generalise have the following purposes. The concordant condition is self-evident. The near_match condition guarantees that vectors that produce a generalisation are, in some sense, close to each other in the input space. The condition on the relative number of asserted inputs (variable diff) controls the level (or sparsity) of generalisation. The larger the value the bigger the inductive leap. The value of 1 is supported by empirical evidence. Finally, the condition on the relative values of num_asserted increases the amount of generalisation, and the condition on the absolute value of num_asserted guarantees that no rule is created that has only $\star$ values and thus covers the entire input space.

```
diff ← |num_asserted($x$) - num_asserted($y$)|

if ($x$ is identical to $y$) then
  increment $x$.counters[$y$.class]
  if ($y$.pty > $x$.pty) then
    $x$.pty ← $y$.pty
    increment $x$.counters[$y$.class]
  else if ($x$.pty > $y$.pty) then
    increment $x$.counters[$x$.class]

else if (($x$ and $y$ are concordant) and
        ($y$ is a subset of $x$)) then
   increment $x$.num_covers

else if (($x$ and $y$ are concordant) and
        ($x$ and $y$ are near_match) and
        (diff $\leq$ 1)) then
   if ((num_asserted($x$) $\leq$ num_asserted($y$)) and
      (num_asserted($x$) > 1)) then
     generalise $x$
   else if (num_asserted($y$) > 1) then
     generalise $y$
     create a node for $y$
     delete $W$

else if (($x$ and $y$ are concordant) and
        ($x$ is a subset of $y$)) then
   increment $y$.num_covers
   create a node for $y$
   delete $W$

else
   create a node for $y$
```

Fig. 5. Adaptation Phase

The inherent symmetry within the “identical” case and across the two “subset” cases is a reflection of ILA’s attempt at reducing the effect of ordering on learning (see section 4.1).

Evaluating ILA for predictive accuracy on a test set consists of presenting every vector in the test set and only running the execution phase. A number of experiments with ILA are reported in [16]. The results on standard classification problems are comparable to those obtained by non-incremental learners. Although a number of useful mechanisms are in place, further experiments are needed to evaluate ILA on genuinely incremental learning tasks.

6. Conclusion

This paper revisits the issue of incrementality in machine learning. It provides formal definitions and a number of examples of both incremental learning tasks and incremental learning algorithms. The paper also argues that incrementality is ubiquitous in learning and that the most natural way of tackling incremental learning task is with incremental learners. Several issues raised by the design of such learners are examined and a system, called ILA, is presented to illustrate how these can be addressed in an implementation.

As computers become an ever-increasing part of people’s lives and their use extends to new and challenging areas, it seems reasonable to expect the need for flexible learning algorithms to increase. If such is the case, incrementality must be taken seriously and embedded in the design of artificial learning systems. Incrementality only adds flexibility and broadens applicability.

Acknowledgements

The author would like to thank the anonymous reviewers for their many detailed and constructive comments.

References

[1] M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge University Press, 1997.

[2] P. Brazdil, Learning in multi-agent environments, In *Proceedings of the European Working Session on Learning*, (1991).

[3] P. Brazdil, M. Gams, S. Sian, L. Torgo and W. van de Velde, Learning in distributed systems and multi-agent environments, In *Proceedings of the Second Workshop on Algorithmic Learning Theory*, (1991), pp. 15–29.

[4] G. Brewka, *Nonmonotonic Reasoning: Logical Foundations of Commonsense*, Cambridge University Press, 1991.

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

[6] A. Cornuéjols, An exploration into incremental learning: the INFLUENCE system, In *Proceedings of the Sixth International Conference on Machine Learning*, (1989), pp. 383–386.

[7] A. Cornuéjols, Getting order independence in incremental learning, In *Proceedings of the Sixth European Conference on Machine Learning*, (1993), pp. 196–212.

[8] A. Cornuéjols, Training issues in incremental learning, *AAAI-93 Spring Symposium*, Report SS-93-06, AAAI Press, 1993.

[9] T.L. Dean and M. Boddy, An analysis of time-dependent planning, In *Proceedings of the National Conference on Artificial Intelligence*, (1988), pp. 49–54.

[10] J.L. Elman, Incremental learning, or the importance of starting small, CRL Technical report 9101, University of California, San Diego, Center for Research in Language, 1991.

[11] D. Fisher, Knowledge acquisition via incremental conceptual clustering, *Machine Learning* **2** (1987), 139–172.

[12] D. Fisher, L. Xu and N. Zard, Ordering effects in clustering, In *Proceedings of the Eighth International Conference on Machine Learning*, (1992).

[13] A. Ginsberg, Theory reduction, theory revision and retranslation, In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (1990), pp. 777–782.

[14] C. Giraud-Carrier and T.R. Martinez, Using precepts to augment training set learning, In *Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, (1993), pp. 46–51.

[15] C. Giraud-Carrier and T. Martinez, An efficient metric for heterogeneous inductive learning applications in the attribute-value language, In E.A. Yfantis (Ed), *Proceedings of the Fourth Golden West International Conference on Intelligent Systems*, Vol. 1, (1994), pp. 341–350.

[16] C. Giraud-Carrier and T. Martinez, ILA: Combining inductive learning with prior knowledge and reasoning, Technical report CSTR-95-03, University of Bristol, Department of Computer Science, 1995.

[17] L.P. Kaelbling, Reinforcement learning, In *MLnet Workshop: Learning in Autonomous Agents*, (invited talk), 1987.

[18] P. Langley, *Personal communication*, 1999.

[19] W. Lukaszewicz, *Non-Monotonic Reasoning: Formalization of Commonsense Reasoning*, Ellis Horwood Limited, 1990.

[20] J. MacGregor, The effects of order in learning classifications by example: Heuristics for finding the optimal order, *Artificial Intelligence* **34** (1988), 361–370.

[21] T.R. Martinez, B. Hughes and D.M. Campbell, Priority ASOCS, *Journal of Artificial Neural Networks* **1**(3) (1994), 403–429.

[22] R.S. Michalski, A theory and methodology of inductive learning, *Artificial Intelligence* **20** (1983), 111–161.

[23] T.M. Mitchell, Generalization as search, *Artificial Intelligence* **18** (1982), 203–226.

[24] T.M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[25] A. Newell and H. Simon, Computer science as empirical enquiry: Symbols and search, *Communications of the ACM* **19**(3) (1976), 113–126.

[26] D. Ourston, and R.J. Mooney, Theory refinement combining analytical and empirical methods, *Artificial Intelligence* **66**(2) (1994), 273–309.

[27] J.R. Quinlan, Inductive learning of decision trees, *Machine Learning* **1** (1986), 81–106.

[28] D. Rumelhart and J. McClelland, *Parallel Distributed Processing*, Vol. 1, MIT Press, 1986.

[29] H. Simon, Why should machines learn?, In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, CA, 1983.

[30] M. Tan, Adaptation-based cooperative learning multiagent systems, In *Proceedings of the Tenth International Conference on Machine Learning*, (1993), pp. 330–337.

[31] G.G. Towell, J.W. Shavlik and M.O. Noordewier, Refinement of approximate domain theories by knowledge-based neural networks, In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (1990), pp. 861–866.

[32] P. Utgoff, ID5: An incremental ID3, In *Proceedings of the Fifth International Workshop on Machine Learning*, (1988), pp. 107–120.