Machine Learning, ?, 1–28 (1999)
© 1999 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Selecting Examples for Partial Memory Learning

MARCUS A. MALOOF maloof@cs.georgetown.edu
Department of Computer Science, Georgetown University, Washington, DC 20057

RYSZARD S. MICHALSKI michalski@gmu.edu
Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA 22030
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Received 25 February 1998 ; Revised 8 February 1999

Editor: Douglas H. Fisher

Abstract. This paper describes a method for selecting training examples for a partial memory learning system. The method selects extreme examples that lie at the boundaries of concept descriptions and uses these examples with new training examples to induce new concept descriptions. Forgetting mechanisms also may be active to remove examples from partial memory that are irrelevant or outdated for the learning task. Using an implementation of the method, we conducted a lesion study and a direct comparison to examine the effects of partial memory learning on predictive accuracy and on the number of training examples maintained during learning. These experiments involved the STAGGER Concepts, a synthetic problem, and two real-world problems: a blasting cap detection problem and a computer intrusion detection problem. Experimental results suggest that the partial memory learner notably reduced memory requirements at the slight expense of predictive accuracy, and tracked concept drift as well as other learners designed for this task.

Keywords: on-line concept learning, partial memory models, concept drift

1. Introduction

Partial memory learners are on-line systems that select and maintain a portion of the past training examples, which they use together with new examples in subsequent training episodes. Such systems can learn by memorizing selected new facts, or by using selected facts to improve the current concept descriptions or to derive new concept descriptions. Researchers have developed partial memory systems because they can be less susceptible to overtraining when learning concepts that change or drift, as compared to learners that use other memory models (Salganicoff, 1993; Maloof, 1996; Widmer & Kubat, 1996; Widmer, 1997).

The key issues for partial memory learning systems are how they select the most relevant examples from the input stream, maintain them, and use them in future learning episodes. These decisions affect the system's predictive accuracy, memory requirements, and ability to cope with changing concepts. A selection policy might keep each training example that arrives, while the maintenance policy forgets examples after a fixed period of time. These policies strongly bias the learner toward recent events, and, as a consequence, the system may forget about important but rarely occurring events. Alternatively, the system may attempt to select prototypical examples and keep these indefinitely. In this case, the learner is strongly anchored to the past and may perform poorly if concepts change or drift.

This paper presents a method for selecting training examples for a partial memory learner. Our approach extends previous work by using induced concept descriptions to select training examples that lie at the extremities of concept boundaries, thus enforcing these boundaries. The system retains and uses these examples during subsequent learning episodes. This approach stores a nonconsecutive collection of past training examples, which is needed for situations in which important training events occur but do not necessarily reoccur in the input stream. Forgetting mechanisms may be active to remove examples from partial memory that no longer enforce concept boundaries or that become irrelevant for the learning task. As new training examples arrive, the boundaries of the current concept descriptions may change, in which case the training examples that lie on those boundaries will change. As a result, the contents of partial memory will change. This continues throughout the learning process.

After surveying relevant work, we present a general framework for partial memory learning and describe an implementation of such a learner, called AQ-Partial Memory (AQ-PM), which is based on the AQ-15c inductive learning system (Wnek, Kaufman, Bloedorn, & Michalski, 1995). We then present results from a *lesion study* (Kibler & Langley, 1990) that examined the effects of partial memory learning on predictive accuracy and on memory requirements. We also make a direct comparison to IB2 (Aha, Kibler, & Albert, 1991), since it is similar in spirit to AQ-PM. In applying the method to the STAGGER Concepts (Schlimmer & Granger, 1986), a synthetic problem, and two real-world problems—the problems of blasting cap detection in X-ray images (Maloof & Michalski, 1997) and computer intrusion detection (Maloof & Michalski, 1995)—experimental results showed a significant reduction in the number of examples maintained during learning at the expense of predictive accuracy on unseen test cases. AQ-PM also tracks drifting concepts comparably to STAGGER (Schlimmer & Granger, 1986) and the FLORA systems (Widmer & Kubat, 1996).

2. Partial Memory Learning

On-line learning systems must have a memory model that dictates how to treat past training examples. Three possibilities exist (Reinke & Michalski, 1988):

1. *full instance memory*, in which the learner retains all past training examples,
2. *partial instance memory*, in which it retains some of the past training examples, and
3. *no instance memory*, in which it retains none.

Researchers have studied and described learning systems with each type of memory model. For example, STAGGER (Schlimmer & Granger, 1986) and Winnow (Littlestone, 1991) are learning systems with no instance memory, while GEM (Reinke

& Michalski, 1988) and IB1 (Aha et al., 1991) are learners with full instance memory. Systems with partial instance memory appear to be the least studied, but examples include LAIR (Elio & Watanabe, 1991), HILLARY (Iba, Woogulis, & Langley, 1988), IB2 (Aha et al., 1991), DARLING (Salganicoff, 1993), AQ-PM (Maloof & Michalski, 1995), FLORA2 (Widmer & Kubat, 1996), and MetaL(B) (Widmer, 1997).

On-line learning systems must also have policies that deal with *concept memory*, which refers to the store of concept descriptions. Researchers have investigated a variety of strategies in conjunction with different models of instance memory. For example, IB1 (Aha et al., 1991) maintains all past training examples but does not store generalized concept descriptions. In contrast, GEM (Reinke & Michalski, 1988) keeps all past training examples in addition to a set of concept descriptions in the form of rules. ID5 (Utgoff, 1988) and ITI (Utgoff, Berkman, & Clouse, 1997) store training examples at the leaf nodes of decision trees, so they are also examples of systems with full instance memory. Actually, ID5 stores a subset of an example's attribute values at the leaves and is an interesting special case of full instance memory. Finally, as an example of a system with no instance memory, ID4 (Schlimmer & Fisher, 1986) uses a new training example to incrementally refine a decision tree before discarding the instance.

For systems with concept memory, learning can occur either in an incremental mode or in a temporal batch mode. Incremental learners modify or adjust their current concept descriptions using new examples in the input stream. If the learner also maintains instance memory, then it uses these examples to augment those arriving from the environment. FLORA2, FLORA3, and FLORA4 (Widmer & Kubat, 1996) are examples of systems that learn incrementally with the aid of partial instance memory.

Temporal batch learners, on the other hand, replace concept descriptions with new ones induced from new training examples in the input stream and any held in instance memory. DARLING (Salganicoff, 1993) and AQ-PM are examples of temporal batch learners with partial instance memory. Any batch learning algorithm, such as C4.5 (Quinlan, 1993) or CN2 (Clark & Niblett, 1989), can be used in conjunction with full or no instance memory. However, this choice depends greatly on the problem at hand. Figure 1 displays a classification of selected learning systems based on concept memory and the various types of instance memory.

Having described the role of instance and concept memory in learning, we will now discuss partial instance memory learning systems that have appeared in the literature. In the following sections, we will focus on learning systems with instance memory. Thus, for the sake of brevity, we will drop the term *instance* when referring to such systems. For example, we will use the term *partial memory* to mean "partial instance memory."

2.1. A Survey of Partial Memory Learning Systems

LAIR (Watanabe & Elio, 1987; Elio & Watanabe, 1991) appears to be one of the first partial memory systems. In some sense, it has a minimal partial memory model

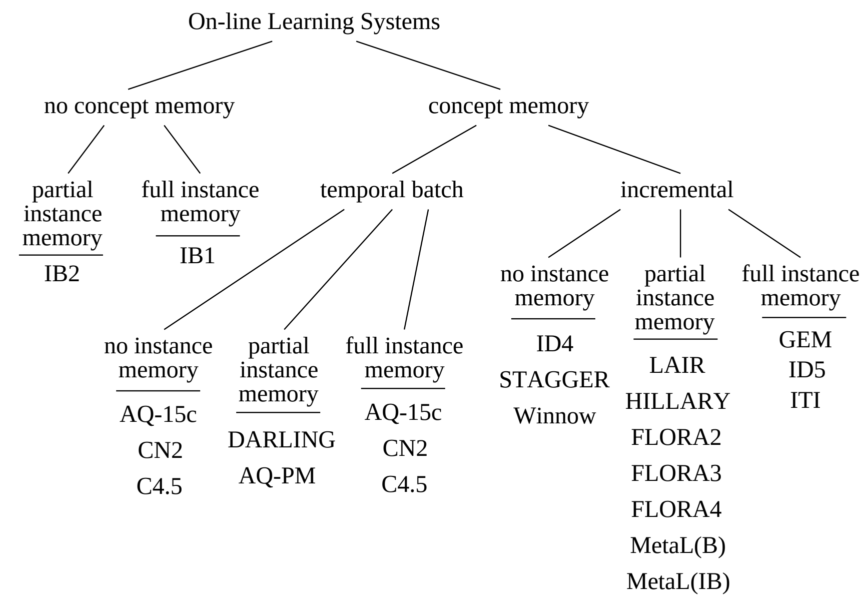


Figure 1. Learning systems classified by concept and instance memory.

because the system keeps only the first positive example. Consequently, it always learns from the one positive example in partial memory and the arriving training examples.

HILLARY (Iba et al., 1988) maintains a collection of recent negative examples in partial instance memory. Positive examples may be added to a concept description as disjuncts but are generalized in subsequent learning steps. HILLARY retains negative examples if no concept description covers them; otherwise, it specializes the concept description. Negative examples that are retained may be forgotten later if they are covered by a positive concept description.

IB2 (Aha et al., 1991), an instance-based learning method, uses a scheme that, like AQ-PM, keeps a nonconsecutive sequence of training examples in memory. When IB2 receives a new instance, it classifies the instance using the examples currently held in memory. If the classification is correct, the instance is discarded. Conversely, if the classification is incorrect, the instance is retained. The intuition behind this is that if an instance is correctly classified, then we gain nothing by keeping it. This scheme tends to retain training examples that lie at the boundaries of concepts. IB3 extends IB2 by adding mechanisms that cope with noise.

DARLING (Salganicoff, 1993) uses a proximity-based forgetting function, as opposed to a time-based or frequency-based function, in which the algorithm initializes the weight of a new example to one and decays the weights of examples within a neighborhood of the new example. When an example's weight falls below a threshold, it is removed. DARLING is also an example of a partial memory learning system, since it forgets examples and maintains only a portion of the past training examples.

The FLORA2 system (Widmer & Kubat, 1996) selects a consecutive sequence of training examples from the input stream and and uses a time-based scheme to forget those examples in partial memory that are older than a threshold, which is set adaptively. This system was designed to track drifting concepts, so during periods when the system is performing well, it increases the size of the window and keeps more examples. If there is a change in performance, presumably due to some change in the target concepts, the system reduces the size of the window and forgets the old examples to accommodate the new examples from the new target concept. As the system's concept descriptions begin to converge toward the target concepts, the size of the window increases, as does the number of training examples maintained in partial memory.

FLORA3 extends FLORA2 by adding mechanisms to cope with changes in context. The change of seasons, for instance, is a changing context, and the concept of warm is different for summer and for winter. Temperature is the contextual variable that governs which warm concept is appropriate. FLORA4 extends FLORA3 by adding mechanisms for coping with noise, similar to those used in IB3 (Aha et al., 1991).

Finally, the MetaL(B) and MetaL(IB) systems (Widmer, 1997) are based on the naive Bayes and instance-based learning algorithms, respectively. These systems, like FLORA3 (Widmer & Kubat, 1996), can cope with changes in context and use partial memory mechanisms that maintain a linear sequence of training examples, but over a fixed window of time. When the algorithm identifies the context, it uses only those examples in the window relevant for that context. MetaL(IB) uses additional mechanisms, such as exemplar selection and exemplar weighting, to further concentrate on the relevant examples in the window.

The FAVORIT system (Krizakova & Kubat, 1992; Kubat & Krizakova, 1992), which extends UNIMEM (Lebowitz, 1987), uses mechanisms for aging and forgetting nodes in a decision tree. Although FAVORIT has no instance memory, we include this discussion because aging and forgetting mechanisms are important for partial memory learners, and because this system uses a third type of forgetting: frequency-based forgetting.

FAVORIT uses incoming training examples to add nodes and to strengthen existing nodes in a decision tree. Over time, aging mechanisms gradually weaken the strengths of nodes. If incoming training examples do not reinforce a node's presence in the tree, then the node's score decays until it falls below a threshold. At this point, the algorithm forgets, or removes, the node. Conversely, if incoming training examples continue to reinforce and revise the node, its score increases. If the score surpasses an upper threshold, then the node's score is fixed and remains so.

2.2. A General Framework for Partial Memory Learning

Based on an analysis of these systems and on our design of AQ-PM, we developed a general algorithm for inductive learning with partial instance memory, presented in

```
1.  Learn-Partial-Memory(Data$_t$, for $t = 1 \ldots n$);
2.      Concepts$_0$ = $\emptyset$;
3.      PartialMemory$_0$ = $\emptyset$;
4.      for $t = 1$ to $n$ do
5.          Missed$_t$ = Find-Missed-Examples(Concepts$_{t-1}$, Data$_t$);
6.          TrainingSet$_t$ = PartialMemory$_{t-1}$ $\cup$ Missed$_t$;
7.          Concepts$_t$ = Learn(TrainingSet$_t$, &optional Concepts$_{t-1}$);
8.          TrainingSet$'_t$ = Select-Partial-Memory (TrainingSet$_t$, Concepts$_t$);
9.          PartialMemory$_t$ = Maintain-Partial-Memory (TrainingSet$'_t$, Concepts$_t$);
10.     end; /* for */
11. end. /* Learn-Partial-Memory */
```

Table 1. Algorithm for partial memory learning.

table 1. The algorithm begins with a data source that supplies training examples distributed over time, represented by Data$_t$, where t is a temporal counter. We generalize the usual assumption that a single instance arrives at a time by placing no restrictions on the cardinality of Data$_t$, allowing it to consist of zero or more training examples. This criterion is important because it ultimately determines the structure of time for the learner.[1] By allowing Data$_t$ to be empty, the learner can track the passage of time, since the passage of time is no longer associated with the explicit arrival of training examples. By allowing Data$_t$ to consist of one or more training examples, the learner can model arbitrary periods of time (e.g., days and weeks) without requiring that a specific number of training examples arrive during that interval. Intuitively, there may be a day when the system learns one thing, but simply because it learns something else does not mean that another day passed.

Initially, the learner begins with no concepts and no training examples in partial memory (steps 2 and 3), although it may possess an arbitrary amount of background knowledge. For the first learning step ($t = 1$), the partial memory learner behaves like a batch learning system. Since it has no concepts and no examples in partial memory, the training set consists of all examples in Data$_1$. It uses this set to induce the initial concept descriptions (step 7). Subsequently, the system must determine which of the training examples to retain in partial memory (steps 8 and 9).

In subsequent time steps ($t > 1$), the system begins by determining which of the new training examples it misclassifies (step 5). The system uses these examples and the examples in partial instance memory to learn new concept descriptions (step 7).

As we have seen in the review of related systems, there are several ways to accomplish this. The system could simply memorize the new examples in the training set. It could also induce new concept descriptions from these examples. And finally, it could use the examples in the training set to modify or alter its existing concept descriptions to form new concept descriptions.

The precise way in which a particular learner determines misclassified examples (step 5), learns (step 7), selects examples to retain (step 8), and maintains those examples (step 9) depends on the concept description language, the learning methods employed, and the task at hand. Therefore, to ground further discussion, we will describe the AQ-PM learning system.

3. Description of the AQ-PM Learning System

AQ-PM is an on-line learning system that maintains a partial memory of past training examples. To implement AQ-PM, we extended the AQ-15c inductive learning system (Wnek et al., 1995), so we will begin by describing this system before delving into the details of AQ-PM.

AQ-15c represents training examples using a restricted version of the attributional language VL_1 (Michalski, 1980). Rule conditions are of the form

'[' <*attribute*> '=' <*reference*> ']',

where <*attribute*> is an attribute used to represent domain objects, and <*reference*> is a list of attribute values. A rule condition is true if the attribute value of the instance to which the condition is matched is in the <*reference*>. Decision rules are of the form

$D \Leftarrow C,$

where D is an expression in the form of a rule condition that assigns a decision to the decision variable, $\Leftarrow$ is an implication operator, and C is a conjunction of rule conditions. If all of the conditions in the conjunction are true, then the expression D is evaluated and returned as the decision. We can also represent training instances in VL_1 by restricting the cardinality of each reference to one. That is, we can view training instances as VL_1 rules in which all conditions have references consisting of single values.

The performance element of AQ-15c consists of a routine that flexibly matches instances with VL_1 decision rules. Decision rules carve out regions in the representation space, leaving some of the space uncovered. If an instance falls into an uncovered region of the space, then, using *strict matching* technique, the system would classify the instance as unknown, which is important for some applications. *Flexible matching* involves computing the *degree of match* between the instance and each decision rule. We can compute this metric in a variety of ways, but, for the experiments discussed here, we computed the degree of match as follows.[2] For each decision class D_i, consisting of n conjunctions C_j, the degree of match, δ_i, for an instance is given by

$$\delta_i = \begin{cases} \frac{\alpha_{ij}}{\beta_{ij}}, & \text{for } j = 1; \\ \delta_{i-1} + \frac{\alpha_{ij}}{\beta_{ij}} - \delta_{i-1}\frac{\alpha_{ij}}{\beta_{ij}}, & \text{for } j = 2 \ldots n, \end{cases} \tag{1}$$

where α_{ij} is the number of conditions in C_j satisfied by the instance, and β_{ij} is the total number of conditions in C_j.

Formula 1 yields a number in the range [0, 1] and expresses the proportion of the conditions of a rule an instance matches. A value of zero means there is no match, and a value of one means there is a complete match. The flexible matching routine returns as the decision the label of the class with the highest degree of match. If the degree of match falls below a certain threshold, then the routine may report "unknown" or "no match".

To learn a set of decision rules, AQ-15c uses the AQ algorithm (Michalski, 1969), a covering algorithm. Briefly, AQ randomly selects a positive training example, known as the *seed*. The algorithm generalizes the seed as much as possible, given the constraints imposed by the negative examples, producing a decision rule. In the default mode of operation, the positive training examples *covered* by the rule are removed from further consideration, and this process repeats using the remaining positive examples until all are covered.

To implement AQ-PM, we extended AQ-15c by incorporating the features outlined in the partial memory algorithm in table 1. AQ-PM finds misclassified training examples by flexibly matching the current set of decision rules with the examples in Data_t (step 5). These "missed" examples are grouped with the examples currently held in partial memory (step 6) and passed to the learning algorithm (step 7). Like AQ-15c, AQ-PM uses the AQ algorithm to induce a set of decision rules from training examples, meaning that AQ-PM operates in a temporal batch mode. To form the new contents of partial memory (step 8), AQ-PM selects examples from the current training set using syntactically modified *characteristic decision rules* derived from the new concept descriptions, which we discuss further in section 3.1. Finally, AQ-PM may use a variety of maintenance policies (step 9), like time-based forgetting, aging, and inductive support, which are activated by setting parameters.

3.1. Selecting Examples

One of the key issues for partial memory learners is deciding which of the new training examples to select and retain. Mechanisms that maintain these examples are also important because some of the examples held in partial memory may no longer be useful. This could be due to the fact that concepts changed or drifted, or that what the system initially thought was crucial about a concept is no longer important to represent explicitly, since the current concept descriptions implicitly capture this information.

Returning to AQ-PM, we used a scheme that selects the training examples that lie on the boundaries of generalized concept descriptions. We will call these examples *extreme examples.* Each AQ-PM decision rule is an axis-parallel hyper-rectangle in discrete n-dimensional space, where n is the number of attributes used to represent domain objects. Therefore, the extreme examples could be those that lie on the surfaces, the edges, or the corners of the hyper-rectangle covering them. For this study, we chose the middle ground and retained those examples that lay on the edges of the hyper-rectangle, although we have considered and implemented the other schemes for retaining examples (Maloof, 1996).

Referring to figure 2, we see a portion of a discrete version of the iris data set (Fisher, 1936). We took the original data set from the UCI Machine Learning Repository (Blake, Keogh, & Merz, 1998) and produced a discrete version using the SCALE implementation (Bloedorn, Wnek, Michalski, & Kaufman, 1993) of

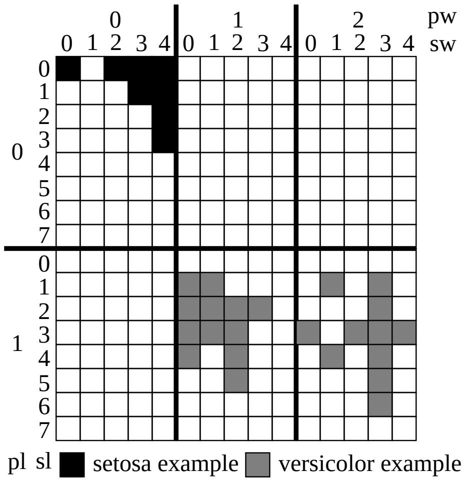


Figure 2. Visualization of the setosa and versicolor training examples.

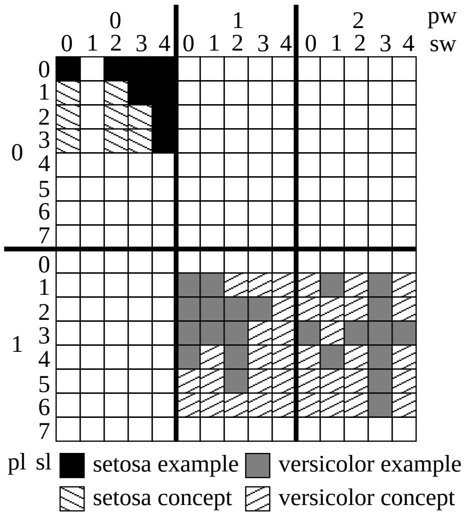


Figure 3. Visualization of the setosa and versicolor concept descriptions with overlain training examples.

the ChiMerge algorithm (Kerber, 1992). Shown are examples of the versicolor and setosa classes with each example represented by four attributes: petal length (pl), petal width (pw), sepal length (sl), and sepal width (sw).

To find extreme or boundary training examples, AQ-PM uses characteristic decision rules, which specify the common attributes of domain objects from the same class (Michalski, 1980). These rules consist of all the domain attributes and their

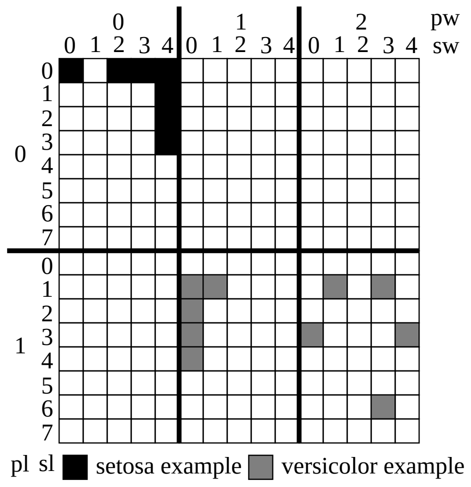


Figure 4. Visualization of the setosa and versicolor extreme examples.

values for the objects represented in the training set, and form the tightest possible hyper-rectangle around a cluster of examples. Returning to our example, figure 3 shows the characteristic rules induced from the training examples pictured in figure 2.

AQ-PM syntactically modifies the set of characteristic rules so they will match examples that lie on their boundaries and then uses a strict matching technique to select the extreme examples. Although AQ-PM uses characteristic rules to select extreme examples, it can use other types of decision rules (e.g., discriminant rules) for classification. Figure 4 shows the examples retained by the selection algorithm, which are those examples that lie on the edges of the hyper-rectangles expressed by characteristic decision rules.

Theorem 1 states the upper bound for the number of examples retained by AQ-PM and its lesioned counterpart. The lesioned version of AQ-PM, which we describe formally in the next section, is equivalent to a temporal batch learning system with full instance memory. For the best case, the partial memory learner will retain fewer training examples than the lesioned counterpart by a multiplicative factor. For the worst case, or the lower bound, the number of examples maintained by the partial memory learner will be equal to that of the lesioned learner. This follows from the proof of Theorem 1 and occurs when the training set consists only of examples that lie on the edges of a characteristic concept description.

THEOREM 1 *For the characteristic decision rule $D \Leftarrow C$ induced from training examples drawn from an* n*-dimensional discrete representation space, the number of training examples retained by the partial memory learner is*

$$\sum_{k=1}^{n} \left[2^{n-1}(|\mathit{reference}_k| - 2)\right] + 2^n,$$

while its lesioned counterpart will retain

$$\prod_{k=1}^{n} |\mathit{reference}_k|.$$

Proof: Let D $\Leftarrow$ C be a characteristic decision rule induced from training examples drawn from an n-dimensional discrete representation space $\Re$. Let c_k be the kth condition in C. By definition, the following three are numerically equivalent:

1. The dimensionality n of $\Re$.
2. The number of conditions $c \in$ C.
3. The number of attributes forming $\Re$.

For the partial memory learner, c_k expresses the kth dimension in the hyper-rectangle and will match $|\mathit{reference}_k|$ training examples along each edge of the kth dimension. Furthermore, c_k corresponds to 2^{n-1} edges in the kth dimension of the hyper-rectangle realized by D $\Leftarrow$ C. Therefore, the number of training examples matched by c_k is

$$2^{n-1}|\mathit{reference}_k|.$$

If we were to compute this number for $k = 1 \ldots n$, then we would overcount the training examples that lie at the corners of the hyper-rectangle. Therefore, we must subtract the two training examples that lie at the endpoints of each edge of the hyper-rectangle, yielding

$$\sum_{k=1}^{n} \left[2^{n-1}(|\mathit{reference}_k| - 2)\right].$$

But now this undercounts the number of training examples because it excludes all of the training examples that lie at the corners. Since there are 2^n corners in an n-dimensional hyper-rectangle, the total number of examples matched is

$$\sum_{k=1}^{n} \left[2^{n-1}(|\mathit{reference}_k| - 2)\right] + 2^n.$$

For the lesioned learner, each attribute value of a training example will map to a corresponding value in a condition c_k, by definition of a characteristic concept description. For a set of training examples, each attribute will result in a condition c_k such that the number of attribute values in the condition's reference is equal to the number of unique values the attribute takes. Therefore, the number of training examples maintained by the lesioned learner is equal to

$$\prod_{k=1}^{n} |\mathit{reference}_k|.$$

$\square$

```
1. AQ-BL(Data_t, for t = 1...n);
2.     Concepts_0 = ∅;
3.     TrainingSet_0 = ∅;
4.     for t = 1 to n do
5.         TrainingSet_t = TrainingSet_{t-1} ∪ Data_t;
6.         Concepts_t = AQ-Learn(TrainingSet_t);
7.     end; /* for */
8. end. /* AQ-BL */
```

Table 2. Algorithm for the lesioned version of AQ-PM, AQ-Baseline (AQ-BL).

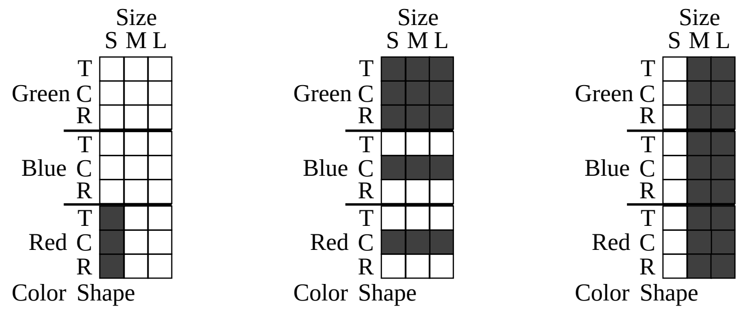
3.2. Forgetting Mechanisms

Forgetting mechanisms are important for partial memory learners for two reasons. First, if the learner selects examples that lie on the boundaries of concept descriptions, as AQ-PM does, and these boundaries change, then there is no reason to retain the old boundary examples. The new extreme examples do the important work of enforcing the concept boundary, so the learner can forget the old ones.

Second, if the learner must deal with concept drift, then forgetting mechanisms are crucial for removing irrelevant and outdated examples held in partial memory. As we will see in the experimental section, when concepts change suddenly, the learner must cope with the examples held in partial memory from the previous target concept. In the context of the new target concept, many of these examples will be contradictory, and forgetting them is imperative.

In AQ-PM, there are two types of forgetting: explicit and implicit. Explicit forgetting occurs when examples in partial memory meet specific, user-defined criteria. In the current implementation, AQ-PM uses a time-based forgetting function to remove examples from partial memory that are older than a certain age.

Implicit forgetting occurs when examples in partial memory are evaluated and deemed useless because they no longer enforce concept boundaries. When computing partial memory, the basic algorithm (table 1) evaluates the training examples currently held in partial memory and those misclassified by the current concept descriptions (step 8). Consequently, it repeatedly evaluates the extreme examples and determines if they still fall on a concept boundary, which gives rise to an *implicit forgetting process.* That is, if the learning algorithm generalizes a concept description such that a particular extreme example no longer lies on the concept boundary, then it forgets the example. We call this an implicit forgetting process because there is no explicit criterion for removing examples (e.g., remove examples older than fifty time steps).



a. Target concept for time steps 1–39.

b. Target concept for time steps 40–79.

c. Target concept for time steps 80–120.

Figure 5. Visualization of the STAGGER Concepts.

4. Experimental Results

In this section, we present a series of experimental results from a lesion study (Kibler & Langley, 1990), in which we used AQ-PM for three problems. To produce the lesioned version of AQ-PM, we simply disabled its partial memory mechanisms, resulting in a system equivalent to a temporal batch learner with full instance memory. We present this learner formally in table 2 and will refer to it as AQ-Baseline (AQ-BL). We also included IB2 (Aha et al., 1991) for the sake of comparison, which is a instance-based learner with a partial memory model.

The first problem, a synthetic problem, is referred to as the "STAGGER Concepts" (Schlimmer & Granger, 1986). It has become a standard benchmark for testing learning algorithms that track concept drift. We derived the remaining two data sets from real-world problems. The first problem entails detecting blasting caps in X-ray images of airport luggage (Maloof & Michalski, 1997), and the second involves using learned profiles of computing behavior for intrusion detection (Maloof & Michalski, 1995). We chose these real-world problems because they require on-line learning and likely involve concepts that change over time. For example, computing behavior changes as individuals move from project to project or from semester to semester. The appearance of visual objects can also change due to deformations of the objects or to changes in the environment. For these experiments, the independent variable is the learning algorithm, and the dependent variables are predictive accuracy and the number of training examples maintained. For both of these measures, we computed 95% confidence intervals, which are also presented. Detailed results for learning time and concept complexity for these and other problems can be found elsewhere (Maloof, 1996).

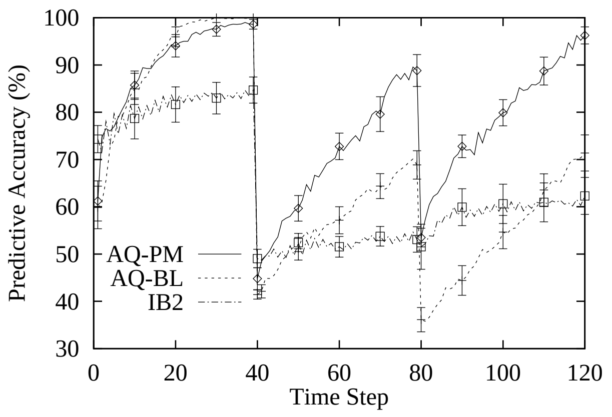


Figure 6. Predictive accuracy for AQ-PM, AQ-BL, and IB2 for the STAGGER Concepts.

4.1. The STAGGER Concepts

The STAGGER Concepts (Schlimmer & Granger, 1986) is a synthetic problem in which the target concept changes over time. Three attributes describe domain objects: size, taking on values small, medium, and large; color, taking on values red, green, and blue; and, shape, taking on values circle, triangle, and rectangle. Consequently, there are 27 possible object descriptions (i.e., events) in the representation space. The presentation of training examples lasted for 120 time steps with the target concept changing every 40 steps. The target concept for the first 39 steps was [size = small] & [color = red]. For the next 40 time steps, the target concept was [color = green] $\vee$ [shape = circular]. And for the final 40 time steps, the target concept was [size = medium $\vee$ large]. The visualization of these target concepts appears in figure 5.

At each time step, a single training example and 100 testing examples were generated randomly.[3] For the results presented, we conducted 60 learning runs using IB2, AQ-PM, and AQ-BL, the lesioned version of AQ-PM.

Referring to figure 6, we see the predictive accuracy results for IB2, AQ-PM, and AQ-BL for the STAGGER Concepts. IB2 performed poorly on the first target concept (85±2.8%) and worse on the final two (53±2.7% and 62±4.0%, respectively). Conversely, AQ-PM and AQ-BL achieved high predictive accuracies for the first target concept (99±1.0% and 100±0.0%, respectively). However, once the target concept changed at time step 40, AQ-BL was never able to match the partial memory learner's predictive accuracy because the former was burdened with examples irrelevant to the new target concept. This experiment illustrates the importance of forgetting mechanisms. AQ-PM was less burdened by past examples because it kept fewer examples in memory and forgot those held in memory after a fixed period of time. AQ-PM's predictive accuracy on the second target concept was

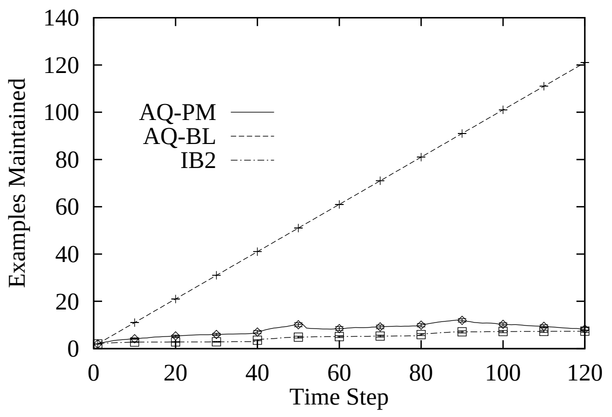


Figure 7. Memory requirements for AQ-PM, AQ-BL, and IB2 for the STAGGER Concepts.

89±3.38%, while AQ-BL's was 69±3.0%. For the third target concept, AQ-PM achieved 96±1.8% predictive accuracy, while AQ-BL achieved 71±3.82%.

The predictive accuracy results for AQ-PM are comparable to those of STAGGER (Schlimmer & Granger, 1986) and of the FLORA systems (Widmer & Kubat, 1996) with the following exceptions. On the first target concept, AQ-PM did not converge as quickly as the FLORA systems but ultimately achieved similar predictive accuracy. On the second target concept, AQ-PM's convergence was like that of the FLORA systems, but it performed about 10% worse on the test cases. Performance (i.e., slope and asymptote) on the third target concept was similar.

Turning to memory requirements, shown in figure 7, we see that the partial memory learners, AQ-PM and IB2, maintained far fewer training examples than AQ-BL. Without the partial memory mechanisms, the baseline learner, AQ-BL, simply accumulated more and more examples. Intuitively, this is an inefficient and inadequate policy when learning changing concepts. Yet, as IB2's predictive accuracy showed, selection mechanisms alone are not enough.

Taking a closer look at the memory requirements for AQ-PM and IB2 (figure 8), we see that the number of examples each learner maintained increases because of example selection mechanisms. Overall, IB2 maintained fewer training examples than AQ-PM, but this savings cannot mitigate its poor predictive accuracy. During the first 40 time steps, for instance, both learners accumulated examples. As each achieved acceptable predictive accuracies, the number of examples maintained stabilized. Once the concept changed at time step 40, both learners increased the number of examples held in partial memory to retain more information about the new concept. The increases in IB2's memory requirements occurred because it adds new examples only if they are misclassified by the examples currently held in memory. When the target concept changed, most of the new examples were misclassified and, consequently, added to memory. Because IB2 kept all of the examples related to the previous target concept, predictive accuracy suffered on this and the final

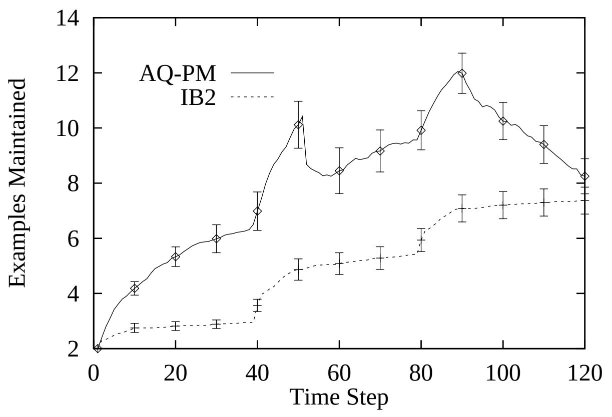


Figure 8. Memory requirements for AQ-PM and IB2 for the STAGGER Concepts.

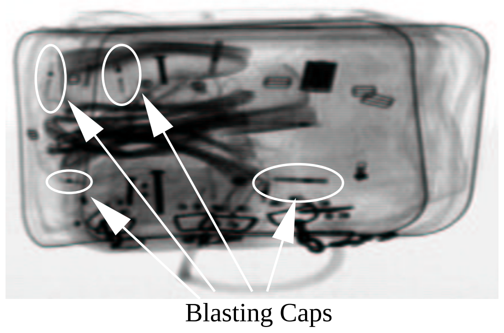


Figure 9. Example of X-ray image used for experimentation.

target concept. Although AQ-PM also increased the number of examples held in memory, it used an explicit forgetting process to remove outdated and irrelevant training examples after a fixed period of fifty time steps, which proved crucial for learning these concepts.

We cannot compare AQ-PM's memory requirements to STAGGER's, since the latter does not maintain past training examples, but we can indirectly compare it to one run of FLORA2 (Widmer & Kubat, 1996). Recall that the size of the representation space for the STAGGER problem is only 27 examples. At time step 50, FLORA2 maintained about 24 examples, which is 89% of the representation space. At the same time step, AQ-PM maintained only 10.11 examples, on average, which is only 37% of the representation space. Over the entire learning run, FLORA2 kept an average of 15 examples, which is 56% of representation space. AQ-PM, on the other hand, maintained only 6.6 examples, on average, which is only 24% of the space.

4.2. Blasting Cap Detection Problem

The blasting cap detection problem involves detecting blasting caps in X-ray images of airport luggage (Maloof & Michalski, 1997). The 66 training examples for this experiment were derived from 5 images that varied in the amount of clutter in the luggage and in the position of the bag relative to the X-ray source. Figure 9 shows a typical X-ray image from the collection. Positive and negative examples of blasting caps were represented using 27 intensity, shape, and positional attributes (Maloof, Duric, Michalski, & Rosenfeld, 1996). We computed eleven attributes for the blob produced by the heavy metal explosive near the center of the blasting cap. We also computed these same eleven attributes for the rectangular region produced by the blasting cap's metal tube. Finally, we used five attributes to capture the spatial relationship between the blob and the rectangular region. These real-valued attributes were scaled and discretized using the SCALE implementation (Bloedorn et al., 1993) of the ChiMerge algorithm (Kerber, 1992).[4] The 15 most relevant attributes were then selected using the PROMISE measure (Baim, 1988). The resulting attributes for the blob were maximum intensity, average intensity, length of a bounding rectangle, and three measures of compactness. For the rectangular region, the selected attributes were length, width, area, standard deviation of the intensity, and three measures of compactness. And finally, the remaining spatial attributes were the distance between the centroids of the blob and rectangle, and the component of this distance that was parallel to the major axis of a fitted ellipse.

We randomly set aside 10% of the original data as a testing set. The remaining 90% was partitioned randomly and evenly into 10 data sets (i.e., Data_t, for $t = 1 \ldots 10$). We then conducted an experimental comparison between IB2, AQ-PM, and the lesioned version of the system, AQ-BL. For each learning run, we presented the learners with Data_t and tested the resulting concept descriptions on the testing set, making note of predictive accuracy and memory requirements. We conducted 100 learning runs, in which we randomly generated a new test set and new data sets Data_t, for $t = 1 \ldots 10$, averaging the performance metrics over these 100 runs.

Figure 10 shows the predictive accuracy results for the blasting cap detection problem. AQ-PM's predictive accuracy was consistently lower than AQ-BL's, which learned from all of the available training data at each time step. When learning stops at time step 10, AQ-PM's predictive accuracy was 7% less than that of the lesioned learner, AQ-BL (81$\pm$3.4% vs. 88$\pm$2.8%). IB2 did not perform well on this task and ultimately achieved a predictive accuracy of 73$\pm$3.8%.

A notable decrease in memory requirements has to be measured against AQ-PM's loss in predictive accuracy, as shown by figure 11. When learning ceased at time step 10, the baseline learner maintained the entire training set of 61$\pm$0.0 examples, while the partial memory learner kept 18$\pm$0.5 training examples, on average, which is roughly 30% of the total number of examples. IB2 retained slightly more examples in partial memory than AQ-PM: 25$\pm$0.6.

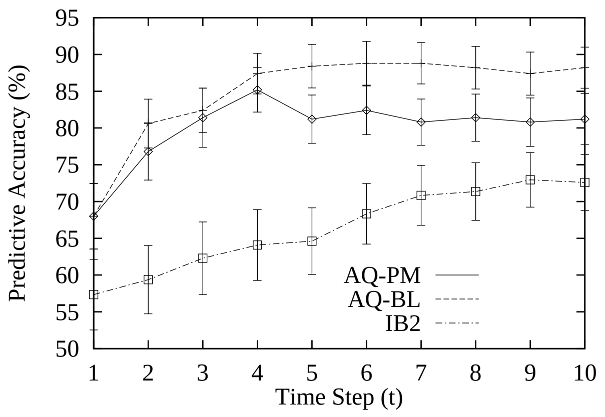


Figure 10. Predictive accuracy for AQ-PM, AQ-BL, and IB2 for the blasting cap detection problem.

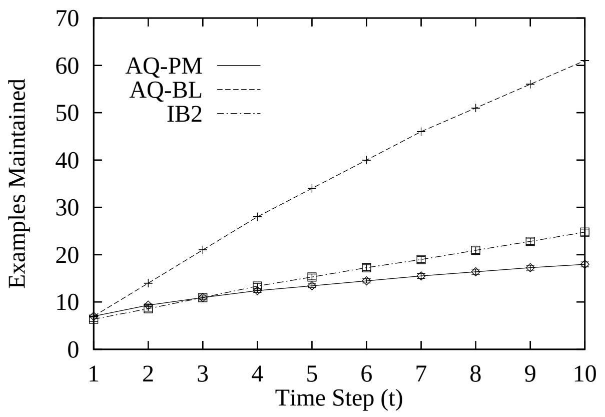


Figure 11. Memory requirements for AQ-PM, AQ-BL, and IB2 for the blasting caps detection problem.

4.3. Computer Intrusion Detection Problem

For the computer intrusion detection problem, we must learn profiles of users' computing behavior and use these profiles to authenticate future behavior. Learning descriptions of intrusion behavior is problematic, since adequate training data is difficult, if not impossible, to collect. Consequently, we chose to learn profiles for each user, assuming that misclassification means that a user's profile is inadequate or that an unauthorized person is masquerading as the user in question. Most existing intrusion detection systems make this assumption.

The data for this experiment were derived from over 11,200 audit records collected for 9 users over a 3 week period. We first parsed each user's computing activity

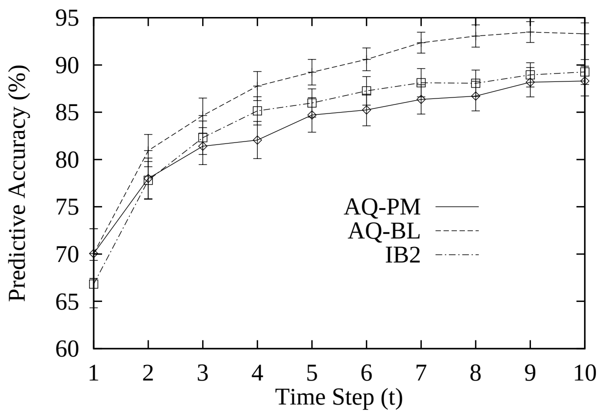


Figure 12. Predictive accuracy for AQ-PM, AQ-BL, and IB2 for the intrusion detection problem.

from the output of the UNIX acctcom command (Frisch, 1995) into *sessions* by segmenting at logouts and at periods of idle time of twenty minutes or longer. This resulted in 239 training examples. We then selected seven numeric audit metrics: CPU time, real time, user time, characters transferred, blocks read and written, CPU factor, and hog factor. Next, we represented each of the seven numeric measures for a session, which is a time series, using the maximum, minimum, and average values, following Davis (Davis, 1981). These 21 real and integer attributes were scaled and discretized using the SCALE implementation (Bloedorn et al., 1993) of the ChiMerge algorithm (Kerber, 1992). Finally, using the PROMISE measure (Baim, 1988), we selected the 13 most relevant attributes: average and maximum real time, average and maximum system time, average and maximum user time, minimum and average characters transferred, average blocks transferred, average and maximum CPU factor, and average and maximum hog factor.

The experimental design for this problem was identical to the one we used for the blasting cap problem. Referring to figure 12, we can see the predictive accuracy results for AQ-PM, AQ-BL, and IB2 for the intrusion detection problem. AQ-PM's predictive accuracy was again slightly lower than AQ-BL's. When learning stopped at time step 10, AQ-PM's accuracy was 88±1.6%, while AQ-BL's was 93±1.2%, a difference of 5%. IB2 fared much better on this problem than on previous ones. When learning ceased, IB2's predictive accuracy was a slightly better than AQ-PM's: 89±1.3%, although this result was not statistically significant ($p < .05$).

Figure 13 shows the memory requirements for each learner for this problem. AQ-PM maintained notably fewer training examples than its lesioned counterpart. When learning ceased at time step 10, the baseline learner, AQ-BL, maintained 221±0.0 examples, while AQ-PM maintained 64±1.0 training examples, which is roughly 29% of the total number of examples. IB2 maintained even fewer examples than AQ-PM. When learning stopped, IB2 held roughly 52±0.8 examples in partial memory, which was slightly fewer than the number held by AQ-PM.

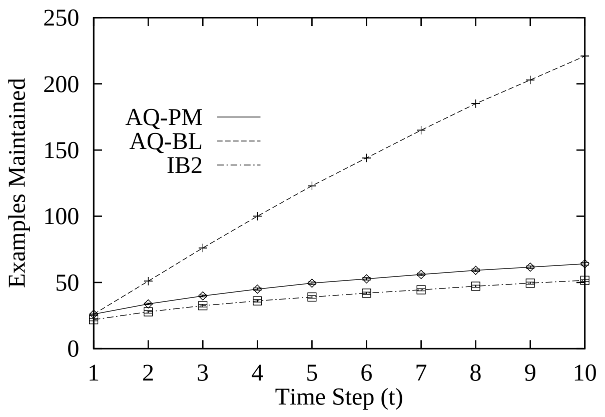


Figure 13. Memory requirements for AQ-PM, AQ-BL, and IB2 for the intrusion detection problem.

4.4. Summary

The lesion study comparing AQ-PM and AQ-BL suggested that the mechanisms for selecting extreme examples notably reduced the number of instances maintained in partial memory at the expense of predictive accuracy. When concepts changed, AQ-PM relied on forgetting mechanisms to remove outdated and irrelevant examples held in memory. Recall that AQ-PM can use two types of forgetting: implicit and explicit. Explicit mechanisms proved crucial for the STAGGER Concepts, but the implicit forgetting mechanisms, in general, had little effect, an issue we explore further in the next section.

The direct comparison to IB2 using the STAGGER Concepts further illustrated the importance of forgetting policies, as it was apparent that the example selection mechanisms alone did not guarantee acceptable predictive accuracy when concepts changed. On the other hand, when concepts were stable, as was the case with the computer intrusion detection and blasting cap detection problems, forgetting mechanisms played a less important role than the selection mechanisms. Moreover, we predict that the differences in performance between AQ-PM and IB2 on these problems were due to inductive bias rather than a limitation of IB2's example selection method. This would explain why IB2 performed well on the intrusion detection problem but performed poorly on the blasting cap detection problem. Indeed, AQ-PM and IB2 used similar selection methods, and experimental results showed that each maintained roughly the same number of examples in memory.

Regarding the indirect comparison to the FLORA systems, AQ-PM performed as well on two of the three STAGGER Concepts, and it appears to have maintained fewer training examples in partial memory. The difference in memory requirements is due to how the learners selected examples from the input stream. The FLORA systems kept a sequence of examples of varying length from the input stream, and, as a result, partial memory likely contained duplicate examples. This would be especially true for a problem like the STAGGER Concepts in which we randomly draw 120 examples from a representation space consisting of 27 domain objects. Conversely, AQ-PM retained only those examples that lay on the boundaries of concept descriptions and, consequently, would not retain duplicate examples or examples from the interior of the concept.

We claim that AQ-PM was able to achieve comparable accuracy while maintaining fewer examples in partial memory because the selected examples enforced concept boundaries and, hence, were of high utility. The two systems do use different concept description languages: AQ-PM uses VL_1, which is capable of representing DNF concepts, whereas the FLORA systems use a conjunctive description language. However, upon analyzing the STAGGER Concepts, we concluded that it is unlikely that representational bias accounted for the differences or similarities in predictive accuracy.

As we noted, AQ-PM did not fare as well as the FLORA systems on the second of the three STAGGER Concepts. Transitioning concept descriptions from the first target concept to the second is the most difficult because it is here that there is the most change in the representation space. It is here that there is the most overlap between the old negative concept and the new positive concept, as depicted in figure 5.

AQ-PM should have had an advantage over an incremental learning system in this situation because it operates in a temporal batch mode. Since AQ-PM replaces old concept descriptions with new ones, it would not be burdened by the information about the old concepts encoded in the concept descriptions. But, because AQ-PM operates in a temporal batch mode, the only cause for its fair performance on the second concept is the examples held in partial instance memory.

As we have discussed, AQ-PM used a simple forgetting policy that removed examples older than fifty time steps. The FLORA systems, on the other hand, used an adaptive forgetting window, which, in this case, more efficiently discarded examples after the concept changed and may account for the difference in performance on the second concept. Making the transition between the second and third STAGGER Concepts is easier than transitioning between the first and second because there is more overlap between the old positive concept description and the new positive concept description (see figure 5). AQ-PM's static forgetting policy worked better during this transition than during the previous one, and the learner achieved predictive accuracies that were comparable to the FLORA systems.

5. Discussion

Intelligent systems need induced hypotheses for reasoning because they generalize the system's experiences. We anticipate that manipulating some concept descriptions to cope with changing concepts will slow a system's reactivity. By keeping extreme examples in addition to concept descriptions, the learner maintains a rough approximation of the current concept descriptions and, consequently, is able to both reason and react efficiently.

When learning stable concepts, we expect slight changes in the positions of concept boundaries. The extreme examples, in this case, document the past and provide stability. On the other hand, when examples arrive that radically change concept boundaries, then the examples held in memory that no longer fall on concept boundaries are removed and replaced with examples that do. This process is actually happening in both situations, but to different degrees. The extreme examples provide stability when it is needed. Yet, they do not hinder the learner because forgetting mechanisms ensure that stability does not result in low reactivity. For systems to succeed in nonstationary environments, they must find a balance between stability and reactivity.

In the sections that follow, we examine a variety of issues related to this study and, more globally, to partial memory learning and nonstationary concepts. In particular, we examine experimental results from other aspects of our study (Maloof, 1996), such as learning time, concept complexity, other methods of example selection, and incremental learning. Then, after discussing the some of the current limitations of this work, we consider directions for the future.

5.1. Learning Time

The experimental results from the lesion study showed that the example selection method greatly reduced the number of training examples maintained when compared to the baseline learner. Because the number of training examples affects run time of the algorithms investigated, reducing the number of training examples maintained resulted in notable decreases in learning time. For the intrusion detection problem, as an example, at time step 10, AQ-PM's learning time was 36.7 seconds, while AQ-BL's was 55.6 seconds,[5] meaning that AQ-BL was 52% slower than AQ-PM for this problem.

5.2. Complexity of Concept Descriptions

We also examined complexity of induced concept descriptions in terms of conditions and rules. AQ-PM produced concepts descriptions that were as complex or simpler than those produced by AQ-BL. The degree to which descriptions induced by AQ-PM were simpler was not as notable as other measures, such as learning time and memory requirements.

Table 3 shows decision rules from the intrusion detection problem that AQ-PM induced for two computer users, daffy and coyote.[6] The first rule, for daffy, consists of one condition involving the average system time attribute, which must fall in the high range of [25352.53. . .63914.66].[7] The class label "daffy" is assigned to the decision variable if the average system time for one of daffy's sessions falls into this range. Therefore, daffy's computing use is characterized by a considerable consumption of system time.

The weights appearing at the end of the rules are strength measures. The t-weight indicates how many total training examples the rule covers. The u-weight indicates how many unique training examples the rule covers. Rules may overlap,

Table 3. Examples of AQ-PM rules for daffy and coyote's computing behavior.

[decision = daffy]	$\Leftarrow$	[averageSystemTime = 25352.53..63914.66] (t-weight: 10, u-weight: 10)
[decision = coyote]	$\Leftarrow$	[averageSystemTime = 3676.80..4579.19] & [averageCharactersTransferred = 0.0..0.20] (t-weight: 7, u-weight: 6)
[decision = coyote]	$\Leftarrow$	[averageRealTime = 44.75..404.84] & [averageBlocks = 0.0..0.39] (t-weight: 4, u-weight: 3)

so two rules can cover the same training example. The rule for daffy's computing use is strong, since it alone covers all of the available training examples. The next two rules characterize coyote's behavior, whose use of computing resources is low, especially compared to daffy's.

5.3. Other Example Selection Methods

The selection method used for this study retained the examples that lay on the edges of the hyper-rectangle expressed by a decision rule. We alluded to similar methods that keep the examples that lie on the corners and surfaces of these hyper-rectangles. Experimental results from a previous study (Maloof, 1996) for the blasting caps and intrusion detection problems showed that keeping the examples that lie on the corners of the hyper-rectangle, as opposed to those on the edges, resulted in slightly lower predictive accuracy and slightly reduced memory requirements. We anticipate that a method retaining the examples that lie on surfaces of the hyper-rectangle will slightly improve predictive accuracy and slightly increase memory requirements when compared to the edges method. From these results, we can conclude that as AQ-PM keeps more and more examples in partial memory, its predictive accuracy will converge to that of the full memory learner.

5.4. Adding Examples to Partial Memory

In this paper, we have discussed a reevaluation strategy for maintaining examples in partial memory. Using this scheme, AQ-PM uses new concept descriptions to test if the misclassified examples and all of the examples in partial memory lie on concept boundaries. It retains those examples that do and removes those that do not. And, as we discussed, this gives rise to an implicit forgetting process.

An alternative scheme is to accumulate examples by computing partial memory using only the misclassified examples and by adding the resulting extreme examples to those already in partial memory. Therefore, once an example is placed in partial memory, it remains there until removed by an explicit forgetting process. For the problems discussed here and elsewhere (Maloof, 1996), we did not see notable differences in performance between the reevaluation policy and the accumulation policy. For example, one would expect that the reevaluation policy would work best for dynamic problems, like the STAGGER Concepts, and the accumulation policy would work best for more stable problems, like the blasting cap problem. To date, our experimental results have not supported this intuition.

5.5. Incremental Learning

In the basic algorithm, we used a temporal batch learning method (table 1, step 7). We have also examined variants using incremental learning algorithms (Maloof, 1996), meaning that the system learns new concept descriptions by modifying the current set of descriptions using new training examples and those examples in partial memory. We have investigated this notion using two incremental algorithms: the GEM algorithm (Reinke & Michalski, 1988), a full instance memory technique, and the AQ-11 algorithm (Michalski & Larson, 1983), a no instance memory technique. We chose these algorithms because their inductive biases are most similar to that of AQ-PM: both use the VL_1 representation language (Michalski, 1980) and the AQ induction algorithm (Michalski, 1969).

Experimental results for the computer intrusion detection and the blasting cap detection problems show evidence that the incremental learning variants of AQ-PM lose less predictive accuracy than AQ-PM using a temporal batch learning method. We can infer that the incremental learning variants perform better because the concepts themselves encode information that is lost when using a temporal batch method. The incremental learning methods take advantage of this information, whereas the temporal batch method does not. Moreover, we intend to evaluate these incremental learning methods using the STAGGER problem to determine how they perform. We may find that the incremental methods perform worse because they encode too much information about the past and reduce the learner's ability to react to changing environments.

5.6. Current Limitations

Many of the current limitations of the approach stem from assumptions the system makes. For example, the system assumes the given representation space is adequate for learning. That is, it is currently incapable of constructive induction (Michalski, 1983). Also, it assumes that the context in which training examples are presented is stationary. Hence, it cannot learn contextual cues (Widmer, 1997). Although we did not implement explicit mechanisms to handle noise, there has been work on such mechanisms in contexts similar to these (Widmer & Kubat, 1996). In general, the selection methodology works best for ordered attributes, taking advantage of their inherent structure. Consequently, for purely nominal domains, the method selects all training examples, since, for each training example, there exists a projection of the representation space in which the example lies on a concept boundary.

5.7. Future Work

Much of the current research assumes that the representation space in which concepts drift or contexts change is adequate for learning. If an environment is nonstationary, then the representation space itself could also change. Learners typically detect concept change by a sudden drop in predictive accuracy. If the learner is subsequently unable to achieve acceptable performance, then it may need to apply constructive induction operators in an effort to improve the representation space for learning. To this end, one may use a program that automatically invokes constructive induction, like AQ-18 (Bloedorn & Michalski, 1998; Kaufman & Michalski, 1998).

Another interesting problem for future research is how to detect good and bad types of change. Consider the problem of intrusion detection. We need systems that are flexible enough to track changes in a user's behavior; otherwise, when changes do occur, the system's false negative rate will increase. Yet, if intrusion detection systems are too flexible, then they may perceive a cracker's behavior as a change in the true user's behavior and adapt accordingly. We envision a two-layer system that learns a historical profile of a user's computing behavior and learns how that historical profile has changed over time. If a user's computing behavior no longer matches the historical profile, then the system would determine if the type of change that occurred is plausible for that user. If it is not, then the system would issue an alert. Such systems should prove to be more robust and should perform with lower false negative rates.

From the standpoint of the methodology, we would like to investigate policies that let the learner function when instances arrive without feedback. Producing decisions without feedback is not necessarily problematic, but, when feedback does arrive after a period of time, the system may realize that many of its past decisions were wrong. Naturally, the simplest policy is to forget past decisions, in which case the learner would never realize that it had made mistakes. Certain applications, like intrusion detection, require systems to be more accountable. However, even though a system may remember past decisions, when it realizes that some were wrong, perhaps it should only issue an alert. Alternatively, the system may seek feedback for the events that led to the incorrect decisions and relearn from them.

From the perspective of the implementation, a fruitful exercise would be to implement the example selection method using another concept representation, like decision trees. There is nothing inherent to the method that limits it to decision rules. We could apply the method to any symbolic representation that uses linear attributes. We could also implement other example selection and maintenance schemes as well as mechanisms for coping with noise and contextual changes, but these latter areas, as we have commented, have been well-studied elsewhere.

Finally, there are several opportunities for additional experimental studies. Here, we investigated concepts that change suddenly. Changes in concepts could also occur more gradually. If we think of concepts as geometric objects in a space, then they could change in shape, position, and size. Consequently, concepts could grow (i.e., change in size, but not in position and shape), move (i.e., change in position, but not in shape and size), deform (i.e., change in shape, but not in position and size), and so on. Although synthetic data sets like these provide opportunities to investigate specific research hypotheses, we are also interested in concept drift in real-world applications like intrusion detection and agent applications (e.g., an agent for prioritizing e-mail).

6. Conclusion

Partial memory learning systems select and maintain a portion of the past training examples and use these examples for future learning episodes. In this paper, we presented a selection method that uses *extreme examples* to enforce concept boundaries. The method extends previous work by using induced concept descriptions to select a nonconsecutive sequence of examples from the input stream. Reevaluating examples held in partial memory and removing them if they no longer enforce concept boundaries results in an implicit forgetting process. This can be used in conjunction with explicit forgetting mechanisms that remove examples satisfying user-defined criteria. Experimental results from a lesion study suggested that the method notably reduces memory requirements with small decreases in predictive accuracy for two real-world problems, those of computer intrusion detection and blasting cap detection in X-ray images. For the STAGGER problem, AQ-PM performed comparably to STAGGER and the FLORA systems. Finally, a direct comparison to IB2 revealed that AQ-PM provided comparable memory requirements and often higher predictive accuracy for the problems considered.

Acknowledgments

We would like to thank Eric Bloedorn, Renée Elio, Doug Fisher, Wayne Iba, Pat Langley, and the anonymous reviewers, all of whom provided suggestions that improved this work and earlier versions of this paper. We would also like to thank the Department of Computer Science at Georgetown University, the Institute for the Study of Learning and Expertise, and the Center for the Study of Language and Information at Stanford University for their support of this work.

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The laboratory's research has been supported in part by the National Science Foundation under grants IIS-9904078 and IRI-9510644, in part by the Advanced Research Projects Agency under grant N00014-91-J-1854, administered by the Office of Naval Research, under grant F49620-92-J-0549, administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under grant N00014-91-J-1351.

Notes

1. The structure of time is not crucially important for this paper, but we do feel that this issue warrants a more sophisticated treatment.

2. AQ-15c has other methods for computing the degree of match, but, based on empirical analysis, we found that this method worked best for the problems in this study.
3. For the first time step, we generated two random examples, one for each class.
4. We ran IB2 using the unscaled, continuous data.
5. We conducted these experiments using a C implementation of AQ-PM running on a Sun Sparc 2.
6. Attribute values have been expressed using their original real ranges.
7. The units in this case are seconds.

References

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, *6*, 37–66.

Baim, P. (1988). A method for attribute selection in inductive learning systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *10*(6), 888–896.

Blake, C., Keogh, E., & Merz, C. (1998). *UCI repository of machine learning databases* ([http://www.ics.uci.edu/~mlearn/MLRepository.html]). University of California, Irvine, Department of Information and Computer Sciences.

Bloedorn, E., & Michalski, R. (1998). Data-driven constructive induction. *IEEE Intelligent Systems*, *13*(2), 30–37.

Bloedorn, E., Wnek, J., Michalski, R., & Kaufman, K. (1993). *AQ17 — A multistrategy learning system: the method and user's guide* (Reports of the Machine Learning and Inference Laboratory No. MLI 93-12). Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–284.

Davis, J. (1981). *CONVART: a program for constructive induction on time dependent data.* Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Elio, R., & Watanabe, L. (1991). An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*, *7*, 7–44.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, *7*, 179–188.

Frisch, A. (1995). *Essential system administration* (Second ed.). Sebastopol, CA: O'Reilly & Associates.

Iba, W., Woogulis, J., & Langley, P. (1988). Trading simplicity and coverage in incremental concept learning. In *Proceedings of the Fifth International Conference on Machine Learning* (pp. 73–79). San Francisco, CA: Morgan Kaufmann.

Kaufman, K., & Michalski, R. (1998). *AQ-18: An environment for learning and natural induction* (Reports of the Machine Learning and Inference Laboratory No. MLI 98-12). Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA.

Kerber, R. (1992). ChiMerge: discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 123–128). Menlo Park, CA: AAAI Press.

Kibler, D., & Langley, P. (1990). Machine learning as an experimental science. In T. Dietterich & J. Shavlik (Eds.), *Readings in machine learning.* San Francisco, CA: Morgan Kaufmann.

Krizakova, I., & Kubat, M. (1992). FAVORIT: concept formation with ageing of knowledge. *Pattern Recognition Letters*, *13*, 19–25.

Kubat, M., & Krizakova, I. (1992). Forgetting and aging of knowledge in concept formation. *Applied Artificial Intelligence*, *6*, 195–206.

Lebowitz, M. (1987). Experiments with incremental concept formation: UNIMEM. *Machine Learning*, *2*, 103-138.

Littlestone, N. (1991). Redundant noisy attributes, attribute errors, and linear-threshold learning using Winnow. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory* (pp. 147–156). San Francisco, CA: Morgan Kaufmann.

Maloof, M. (1996). *Progressive partial memory learning.* Doctoral dissertation, School of Information Technology and Engineering, George Mason University, Fairfax, VA.

Maloof, M., Duric, Z., Michalski, R., & Rosenfeld, A. (1996). Recognizing blasting caps in X-ray images. In *Proceedings of the Image Understanding Workshop* (pp. 1257–1261). San Francisco, CA: Morgan Kaufmann.

Maloof, M., & Michalski, R. (1995). A method for partial-memory incremental learning and its application to computer intrusion detection. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence* (pp. 392–397).

Maloof, M., & Michalski, R. (1997). Learning symbolic descriptions of shape for object recognition in X-ray images. *Expert Systems with Applications*, *12*(1), 11–20.

Michalski, R. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing* (Vol. A3, pp. 125–128).

Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*(4), 349–361.

Michalski, R. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Francisco, CA: Morgan Kaufmann.

Michalski, R., & Larson, J. (1983). *Incremental generation of VL_1 hypotheses: the underlying methodology and the description of program AQ11* (Technical Report No. UIUCDCS-F-83-905). Department of Computer Science, University of Illinois, Urbana.

Quinlan, J. (1993). *C4.5: Programs for machine learning.* San Francisco, CA: Morgan Kaufmann.

Reinke, R., & Michalski, R. (1988). Incremental learning of concept descriptions: a method and experimental results. In J. Hayes, D. Michie, & J. Richards (Eds.), *Machine intelligence 11.* Oxford: Clarendon Press.

Salganicoff, M. (1993). Density-adaptive learning and forgetting. In *Proceedings of the Tenth International Conference on Machine Learning* (pp. 276–283). San Francisco, CA: Morgan Kaufmann.

Schlimmer, J., & Fisher, D. (1986). A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 496–501). Menlo Park, CA: AAAI Press.

Schlimmer, J., & Granger, R. (1986). Beyond incremental processing: tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 502–507). Menlo Park, CA: AAAI Press.

Utgoff, P. (1988). ID5: an incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning* (pp. 107–120). San Francisco, CA: Morgan Kaufmann.

Utgoff, P., Berkman, N., & Clouse, J. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning*, *29*, 5–44.

Watanabe, L., & Elio, R. (1987). Guiding constructive induction for incremental learning from examples. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 293–296). San Francisco, CA: Morgan Kaufmann.

Widmer, G. (1997). Tracking context changes through meta-learning. *Machine Learning*, *27*, 259–286.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*, 69–101.

Wnek, J., Kaufman, K., Bloedorn, E., & Michalski, R. (1995). *Selective induction learning system AQ15c: the method and user's guide* (Reports of the Machine Learning and Inference Laboratory No. MLI 95-4). Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA.