Intelligent Information Systems VIII
Proceedings of the Workshop held in
Ustroń, Poland, June 14–18, 1999
pp. 70–79

AQ-PM: A System for Partial Memory Learning

Marcus A. Maloof[1] and Ryszard S. Michalski[2]

[1]Department of Computer Science
Georgetown University
Washington, DC 20057
maloof@cs.georgetown.edu

[2]Machine Learning and Inference Laboratory
George Mason University, Fairfax, VA 22030
and Institute of Computer Science
Polish Academy of Sciences, Warsaw
michalski@gmu.edu

Abstract. This paper describes AQ-PM, a system for partial memory learning, which determines and memorizes representative concept examples, and then uses them with new training examples to induce new concept descriptions. Our approach uses "extreme" examples that lie at the boundaries of current concept descriptions. We evaluated the system by applying it to synthetic and real-world learning problems. In the experiments, the partial memory learner notably reduced memory requirements for storing examples at the slight expense of predictive accuracy. The system also performed well when tracking concept drift.

Keywords: concept learning, rule induction, partial instance memory models

1 Introduction

Partial memory learners are on-line learning systems that select and store a portion of the past training examples. They can use these examples in different ways. One approach is to use them together with new, incoming examples for generating new concept descriptions. Another is to use them together with the current hypothesis and the new incoming examples to improve the current hypothesis.

The central issues in designing a partial memory learning system are how to determine representative examples from the input stream, how to maintain them, and how to use them in future learning episodes. These decisions affect the system's capabilities, such as accuracy, memory requirements, and ability to cope with concept drift.

After briefly 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). We then present results from a lesion study 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) and to a computer intrusion detection problem (Maloof & Michalski, 1995), experimental results showed a significant reduction in the number of examples maintained during learning at the slight expense of predictive accuracy on unseen test cases. AQ-PM also tracked drifting concepts comparably to STAGGER (Schlimmer & Granger, 1986) and the FLORA systems (Widmer & Kubat, 1996).

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

IB2 (Aha et al., 1991), an instance-based learning method, uses a scheme that, like AQ-PM, keeps a nonconsecutive sequence of training examples from the input stream 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.

The FLORA2 system (Widmer & Kubat, 1996) selects a consecutive sequence of training examples from the input stream and forgets 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 accuracy, 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.

Other examples of partial memory learning systems include HILLARY (Iba, Woogulis, & Langley, 1988), DARLING (Salganicoff, 1993), and MetaL(B) (Widmer, 1997).

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

Fig. 1. Algorithm for partial memory learning.

3 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 figure 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.

Initially, the learner begins with no concepts and no training examples in partial memory (step 1), 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 5). Subsequently, the system must determine which of the training examples to retain in partial memory (steps 6 and 7).

In subsequent time steps ($t > 1$), the system begins by determining which of the new training examples it misclassifies (step 3). The system uses these examples and the examples in partial instance memory to learn new concept descriptions (step 5).

The precise way in which a particular learner determines misclassified examples (step 3), learns (step 5), selects examples to retain (step 6), and maintains those examples (step 7) depends on the concept description language, the learning methods employed, and the task at hand. Therefore, to ground further discussion, we will now describe the AQ-PM learning system.

4 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, Kaufman, Bloedorn, & Michalski, 1995), so we will begin by describing this system before delving into the details of AQ-PM.

In AQ-15c, 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. 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 view training instances as 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 decision rules. This involves computing the *degree of match*, which is the proportion of rule conditions an instance matches. This computation yields a number in the range [0, 1] with a value of zero meaning there is no match, and a value of one meaning there is a complete match. The flexible matching routine returns as the decision the label of the class with the highest degree of 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 figure 1. AQ-PM finds misclassified training examples by flexibly matching the current set of decision rules with the examples in Data_t (step 3). These "missed" examples are grouped with the examples currently held in partial memory (step 4) and passed to the learning algorithm (step 5). 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 6), 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 4.1. Finally, AQ-PM may use a variety of maintenance policies (step 7), like time-based forgetting, aging, and inductive support, which are activated by setting parameters.

4.1 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 training examples that lie on the boundaries of generalized concept descriptions. We will call these examples *extreme*

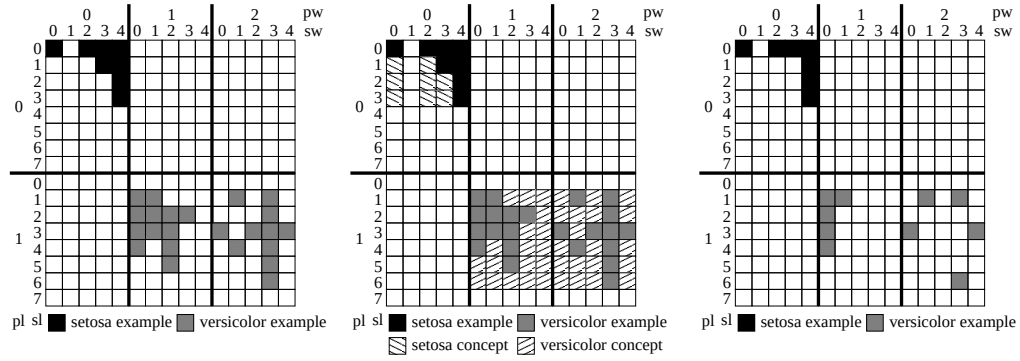

Fig. 2. Visualization of the setosa and versicolor classes from the iris data set. Left: Original training examples. Middle: Characteristic concept descriptions with overlain training examples. Right: Derived extreme examples.

examples. Each AQ-PM decision rule is an axis-parallel hyper-rectangle in a 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 other schemes (Maloof, 1996).

Referring to the left graphic in figure 2, we see a portion of a discrete version of the iris data set. We took the original data set from the UCI Machine Learning Repository (Blake, Keogh, & Merz, 1998) and produced a discrete version using 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 values for the objects represented in the training set, and form the tightest possible hyper-rectangle around a cluster of examples. The middle graphic in figure 2 shows the characteristic rules induced from the training examples.

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. The right graphic in figure 2 shows the examples retained by the selection algorithm, which are those examples that lie on the edges of the hyper-rectangles expressed by the characteristic decision rules.

5 Experimental Results

In this section, we present a series of experimental results from a lesion study. 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 that retains all of training examples in the input stream. We will refer to this learner as AQ-Baseline (AQ-BL). We also included IB2 (Aha et al., 1991) for the sake of comparison. The first problem is referred to as the "STAGGER Concepts" (Schlimmer & Granger, 1986), and the second is a real-world problem that involves learning profiles of computing behavior for intrusion detection (Maloof & Michalski, 1995).

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).

5.1 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].

At each time step, a single training example and 100 testing examples were generated randomly.[1] 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 3, 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$\pm$2.8%) and worse on the final two (53$\pm$2.7% and 62$\pm$4.0%, respectively). Conversely, AQ-PM and AQ-BL achieved very high predictive accuracies for the first target concept (99$\pm$1.0% and 100$\pm$0.0%, respectively). However, once the target concept changed at time step 40, AQ-BL was never able to match the partial memory learner's predictive accuracy because the former was burdened with examples irrelevant to the new target concept. This experiment illustrates the importance of forgetting mechanisms. AQ-PM was less burdened by past examples because it kept fewer examples in memory and forgot those held in memory after a fixed period of time, an issue we discuss further in the following paragraphs. AQ-PM's predictive accuracy on the second target concept was 89$\pm$3.38%, while AQ-BL's was 69$\pm$3.0%. For the third target concept, AQ-PM achieved 96$\pm$1.8% predictive accuracy, while AQ-BL achieved 71$\pm$3.82%.

[1] For the first time step, we generated two random examples, one for each class.

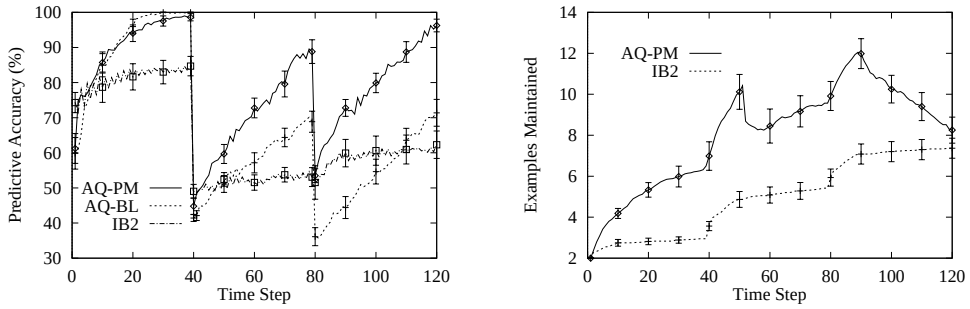

Fig. 3. Experimental results for the STAGGER Concepts. Left: Predictive accuracy for AQ-PM, AQ-BL, and IB2. Right: Memory requirements for AQ-PM and IB2.

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 the memory requirements for AQ-PM and IB2, shown on the right of figure 3, we see that the number of examples each learner maintained increases because of example selection mechanisms.[2] 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 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-

[2] We do not show the memory requirements for AQ-BL since it simply memorized each example, thus producing a rather uninteresting straight line.

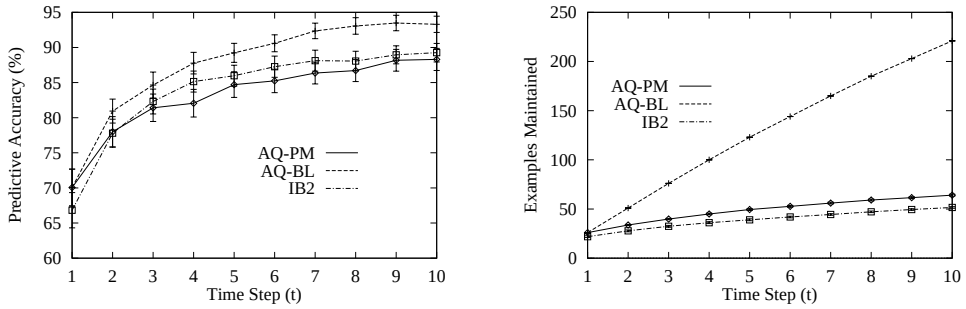

Fig. 4. Experimental results for the computer intrusion detection problem. Left: Predictive accuracy for AQ-PM, AQ-BL, and IB2. Right: Memory requirements for AQ-PM, AQ-BL, and IB2.

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.

5.2 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. 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 from the output of the UNIX acctcom command 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. These 21 real and integer attributes were scaled and discretized using the ChiMerge algorithm (Kerber, 1992). Finally, using the PROMISE measure (Baim, 1988), we selected the 13 most relevant attributes.

We randomly selected 10% of the original data for testing, while the remaining 90% was partitioned randomly and evenly into 10 sets (i.e., Data_t, for $t = 1 \ldots 10$). For each data set in succession, AQ-PM, AQ-BL, and IB2 induced concept descriptions that were evaluated on the test set, recording predictive accuracy and the number of examples maintained. We conducted 100 of these learning runs, averaging the performance metrics over these runs.

Referring to figure 4, 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 the previous one. 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 ($p < .05$).

The right chart of figure 4 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, 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 less than the number held by AQ-PM. Again, this difference was not statistically significant ($p < .05$).

6 Conclusion

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 and is currently incapable of constructive induction. Although it cannot learn contextual cues, and we did not implement explicit mechanisms to handle noise, there has been work on such mechanisms in contexts similar to these (Widmer & Kubat, 1996).

For future work, we are interested in combining constructive induction with techniques for tracking concept drift. Much of the current research assumes that the representation space in which concepts drift or contexts change is adequate for learning. Yet if an environment is nonstationary, then the representation space itself could also change. If the learner is unable to achieve acceptable performance when concepts change, 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).

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. Experimental results from a lesion study suggested that the method notably reduces memory requirements with small decreases in predictive accuracy for the computer intrusion detection problem. 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.

Acknowledgements. We would like to thank Eric Bloedorn, Renée Elio, Doug Fisher, Wayne Iba, and Pat Langley, whose suggestions improved 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. An expanded version of this paper has been accepted for publication in *Machine Learning*.

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 Defense 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.

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 PAMI*, *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.

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

Iba, W., Woogulis, J., & Langley, P. (1988). Trading simplicity and coverage in incremental concept learning. In *ICML '88*, pp. 73–79.

Kerber, R. (1992). ChiMerge: discretization of numeric attributes. In *AAAI-92*, pp. 123–128.

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

Maloof, M., & Michalski, R. (1995). A method for partial-memory incremental learning and its application to computer intrusion detection. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, pp. 392–397.

Michalski, R. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the 5th International Symposium on Information Processing*, Vol. A3, pp. 125–128.

Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. *IEEE PAMI*, *2*(4), 349–361.

Salganicoff, M. (1993). Density-adaptive learning and forgetting. In *ICML '93*, pp. 276–283.

Schlimmer, J., & Granger, R. (1986). Beyond incremental processing: tracking concept drift. In *AAAI-86*, pp. 502–507.

Watanabe, L., & Elio, R. (1987). Guiding constructive induction for incremental learning from examples. In *IJCAI-87*, pp. 293–296.

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. Reports of the Machine Learning and Inference Laboratory MLI 95-4, George Mason University, Fairfax, VA.