Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence, 392–397
Washington, DC, November 5-8, 1995

A Method for Partial-Memory Incremental Learning and its Application to Computer Intrusion Detection

Marcus A. Maloof Ryszard S. Michalski*

Machine Learning and Inference Laboratory
422 Science & Technology II
George Mason University
Fairfax, VA 22030-4444
{maloof, michalski}@aic.gmu.edu
*Also: Institute of Computer Science, Polish Academy of Science

Abstract

This paper describes a partial-memory incremental learning method based on the AQ15c inductive learning system. The method maintains a representative set of past training examples that are used together with new examples to appropriately modify the currently held hypotheses. Incremental learning is evoked by feedback from the environment or from the user. Such a method is useful in applications involving intelligent agents acting in a changing environment, active vision, and dynamic knowledge-bases. For this study, the method is applied to the problem of computer intrusion detection in which symbolic profiles are learned for a computer system's users. In the experiments, the proposed method yielded significant gains in terms of learning time and memory requirements at the expense of slightly lower predictive accuracy and higher concept complexity, when compared to batch learning, in which all examples are given at once.

1 Introduction

This paper describes a partial-memory incremental learning method. The method is based on the AQ inductive learning algorithm and uses Variable-Valued Logic (VL_1) as a representation language [1, 2, 3, 4]. The proposed method is incremental in that (a) static concepts are learned over time, and (b) concepts that change over time are learned. The proposed method operates using a partial-memory mode [5], in which representative examples are maintained throughout the learning process that maximally expand and constrain learned concepts in the event space. Learned concepts aid in determining the set of representative concepts. This partial-memory scheme is contrasted by no-memory incremental learning (e.g., reinforcement learning), and full-memory incremental learning [6, 5, 7].

New applications, such as intelligent agents (e.g., [8]) and active vision (e.g., [9]), require autonomous or semi-autonomous functioning and adaptation to changes in the domain, the environment, or the user. Such requirements suggest that incremental learning, as opposed to batch learning, is needed. As experimental results presented here demonstrate, when compared to batch learning, partial-memory incremental learning yields faster learning times and reduced memory requirements at the expense of slightly lower predictive accuracy. Although incremental learning is needed in application areas such as intelligent agents and active vision, the application considered here is a dynamic knowledge-based system for computer intrusion detection [10].

Quite a bit of research has been conducted in attempts to statistically model user behavior [11, 12, 13, 14]. Statistical models, or profiles, once acquired, are subsequently used to verify that a user's recent behavior is consistent with past behavior. While a statistical approach to this problem is certainly valid, there are advantages to the machine learning approach taken here. These advantages include using both statistical and logical information when learning user profiles, learning symbolic concepts which can be inspected and understood by humans, and finally, because learned concepts are directly accessible, incremental learning is possible, which allows the system to adapt to changes in a user's behavior over time. See [15] for a more complete literature review.

The organization of this paper is as follows. The next section introduces the incremental learning architecture and method based on VL_1 and the AQ algorithm. Section 3 describes experimental results. The paper concludes with a discussion of the results and directions for future work.

2 Methodology

Viewing an intrusion detection application as a concept learning problem gives rise to two distinct phases (see Figure 1). The first phase, or the *start-up* phase, involves collecting an initial set of training examples such that a sufficient concept can be learned and will provide the system with enough inferential capability to be useful in its intended environment. This phase equates to the historical and traditional paradigm of concept learning or learning from examples.

The second phase, or the update phase, involves installing the system in its environment where it

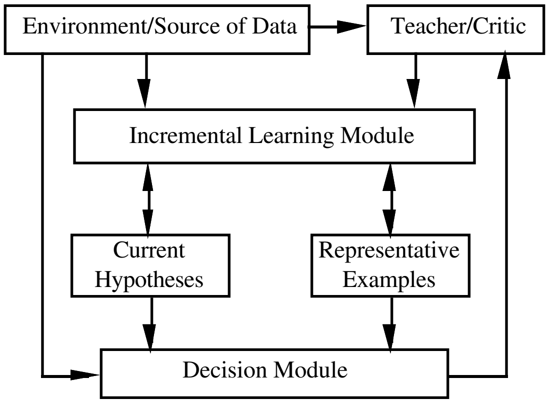

Figure 1. Partial-memory incremental learning architecture and methodology.

should function in a semi-autonomous fashion. During the update phase, the system must incrementally learn and adapt to changes in the environment or in the behaviors of users. Incremental learning is required when an observation is misclassified, which is determined by feedback from the teacher (or user) or from the environment.

After the initial concepts have been learned in the start-up phase and the system has been deployed, the learned concepts are used for inference. The system will receive reinforcement or criticism from its environment, its user, or both. If the system makes a wrong decision, then this is a signal that the system's concepts require refinement. The misclassified training example is included with the existing representative examples and inductive learning takes place. Once new concepts are learned, the training examples are evaluated and those determined to be representative are selected to form the new set of representative examples. The new concepts are used for inference and the new representative training examples are stored. This process repeats indefinitely.

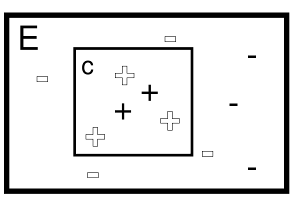

Figure 2. Partial-memory incremental learning.

A partial-memory incremental learning approach uses learned concepts to determine which training examples establish the outer bounds of a concept. Referring to Figure 2, assume that for some event space or representation space *E*, we have a collection of positive and negative examples and that we have learned some concept *c* that is complete and consistent. That is, the concept covers all of the positive examples and none of the negative examples. The representative positive examples of a concept are those examples that lie at the boundaries of the concept in the event space (i.e., the outlined plus signs), while the representative negative examples are those that constrain the concept in the event space (i.e., the outlined minus signs). All other training examples can be discarded.

Several issues must be addressed when applying a partial-memory incremental learning method to a problem. The first concerns how representative examples are chosen. The second involves how these representative examples are maintained throughout the incremental learning process. Regarding the first issue of how representative examples are chosen, for this study maximally general examples were used that either expanded or constrained concepts in the representation space. Specifically, the attribute values for the attributes appearing in learned rules were retained, while other attribute values were changed to unknown or don't care values. The set of representative examples was the set union of these maximally general examples, to eliminate redundancies.

The second issue involves how representative examples are maintained throughout the incremental learning phase. This is an important consideration since examples are generalized and what is considered representative early in incremental learning may not remain representative throughout the incremental learning phase. Consequently, attributes whose values are discarded early may prove to be better in later stages. On the other hand, fewer generalized examples are likely to be kept than specialized examples. One method is to eliminate old representative examples as what is considered 'representative' changes. Another method, the method used here, is to keep all past representative examples regardless of how the representative examples change. The first method would probably be suitable for rapidly changing domains, whereas the second method would yield better results in more stable domains.

This method is summarized in the following algorithm:

Given data partitions DATA_i, for $i = 1..10$

0. $i = 1$
1. $\text{TRAIN}_i = \text{DATA}_i$
2. $\text{CONCEPTS}_i = \text{Learn}(\text{TRAIN}_i)$
3. $\text{REPRESENTATIVE}_i = \text{FindRepresentativeExamples}(\text{CONCEPTS}_i, \text{TRAIN}_i)$
4. $\text{MISSED}_i = \text{FindMissedNewExamples}(\text{CONCEPTS}_i, \text{DATA}_{i+1})$
5. $\text{TRAIN}_{i+1} = \text{REPRESENTATIVE}_i \cup \text{MISSED}_i$
6. $i = i + 1$
7. go to step 2

The counter *i* represents a temporal counter that signifies the passage of time in the system's environment. When *i* = 1, steps 1–3 relate to the start-up phase. The remaining steps of the algorithm for *i* = 1 and under different values of *i* relate to the update phase. Representative examples are found in step 3 using the method described previously. In step 4, missed examples are found by testing new observations, which are represented by DATA_{i+1}, using the current set of learned concepts $\mathrm{CONCEPTS}_i$. Those missed examples are identified by a user and are communicated to the system as criticism. No feedback is viewed as reinforcement. The set of representative examples and the set of misclassified examples are given back to the learning algorithm that incrementally learns new concepts.

2.1 Application to Intrusion Detection

The proposed incremental learning method was applied to the problem of computer system intrusion detection. Typically, an intruder masquerades as one of the legitimate system users. If machine learning could be used to learn *use patterns* for the users of the computer system, then these patterns could be used to detect intruders. This research is most similar to the work of Teng et al. [17] in the sense that we are inductively learning symbolic rules. It differs in that we do not learn from temporal sequences of actions.

In a traditional concept learning scenario, we divide training examples into classes, express the training examples in a representation space that facilitates learning and assert that the examples themselves are sufficient for learning the intended concept. For this application, we might be tempted to divide patterns into classes relating to a "legitimate user" and "intruder", but collecting examples of an intruder's behavior would be a difficult task, since intrusions are a relatively infrequent events. Rather, we should learn use patterns for classes of users or for individual users on the system. In all legitimate uses, the user's login name should match the decision given by the system. If the system's decision does not match the user's login name, then the user is possibly an intruder and appropriate security actions can be taken. These would include making a entry in a systems log file or even forcing the suspect user off the system.

Because of AQ's flexible matching algorithm [16], the intrusion detection system not only produces a decision or classification (i.e., a user's identity), but it also provides a measure of certainty using the degree of match. The degree of match functions similarly to the abnormality measure in NIDES [18].

3 Experimental Results

A series of experiments was conducted using Unix *acctcom* audit data. Accounting data was collecting for a period of three weeks yielding over 11,200 audit records. A set of experiments involved partial-memory incremental learning from two of the system's active users. Only two users were selected, since the bulk of the experimentation was carried out manually. Performance comparisons are made between AQ15c batch learning and partial-memory incremental learning. Further details regarding batch learning experimental results and the specific learning parameters used can be found in [15].

3.1 Data Preparation

The first task involved extracting training examples for each user. A *session* is defined as a contiguous period of activity bounded by a gap in activity of 20 minutes or more. This includes idle time and logouts. Audit data produced by the Unix *acctcom* command was parsed into sessions by user. Attributes were then computed from the various data fields in the audit file. Each numeric metric in an audit file is a time series. Davis [19] characterized time series data for symbolic learning by taking the minimum, maximum, and average values of a time series over a window. For this application, a window is a session of activity. Average, maximum, and minimum computations were made for seven metrics in the audit file: the real time, CPU time, user time, characters transferred, blocks read and written, the CPU factor, and the hog factor. Consequently, each training example, which was derived from a single user session, consisted of 21 continuous or real-valued attributes. Totally, there were 239 training examples distributed over 9 classes, which correspond to 9 selected users.

AQ15c requires discrete-valued attributes, so the SCALE implementation [20] of the ChiMerge algorithm [21] was used. The ChiMerge algorithm merges real-valued attributes into discrete intervals using the chi-square statistic to correlate intervals to classes. For example, the `minchar` attribute, which is the minimum number of characters transferred during a session, ranged from 0.0 to 18747.28. ChiMerge determined that only seven discrete levels were needed for this attribute. After ChiMerge scaling, attribute levels for the training data ranged between 5 and 76.

The final step in data preparation was to select the most relevant attributes. The entropy measure [22] and the PROMISE score [23] were computed for each discrete attribute. Those attributes with a low score (below 0.9) were discarded, leaving 13 attributes, as follows: average and maximum real time, average and maximum system time, average and maximum user time, average and maximum characters transferred, average blocks transferred, average and maximum CPU factor, and average and maximum hog factor.

3.2 Incremental Learning Experimental Method

Two classes were selected from the original training data. Batch learning experiments on the entire data set are reported by Maloof and Michalski [15]. The training data for these two classes were partitioned into 10 sets and used for partial-memory incremental learning experiments, which were compared to AQ15c batch learning. The batch learning experiment involved accumulating the training examples from the 10 data partitions for learning. Partial-memory incremental learning was carried out using the algorithm described above. For these experiments, the incremental algorithm was executed manually for the 10 data partitions.

3.3 Incremental Learning Experimental Results

Several experimental comparisons were made between AQ15c batch learning and partial-memory incremental learning using a variety of metrics, namely, predictive accuracy, learning time, rule complexity, and the number of examples maintained during learning. Figure 3 illustrates how AQ15c's predictive accuracy varies with respect to the portion of training data under batch and partial-memory incremental learning. Although the difference in predictive accuracy is large early in the learning process, toward the end of learning, predictive accuracy differs by only 2%.

Figure 4 provides a learning time comparison between AQ15c batch and partial-memory incremental learning. After the first learning step, incremental learning time was consistently less than 0.1 CPU seconds.

Figure 5 demonstrates the number of examples each approach, partial-memory and batch learning, required. Although batch learning required a linearly increasing quantity examples produced only a moderate increase in predictive accuracy of 2% over partial-memory incremental learning.

The final comparison, illustrated by Figure 6, shows how batch learning and partial-memory incremental learning compare with respect to rule complexity, or the number of conditions in the learned concepts. Batch learning produced less complex rules than partial-memory learning, but this is possibly an artifact of how partial-memory learning was carried out, which was manually. There are probably ways to optimize rules based on the representative examples that will yield simpler rules. This notion will be investigated during implementation.

Note that these results are consistent with the full-memory incremental learning results reported by Reinke and Michalski [5]. Future work will compare partial-memory incremental learning with AQ15 full-memory incremental learning.

4 Discussion and Future Work

Figure 7 shows two AQ15c rules induced for users daffy and elmer. These rules were taken from the

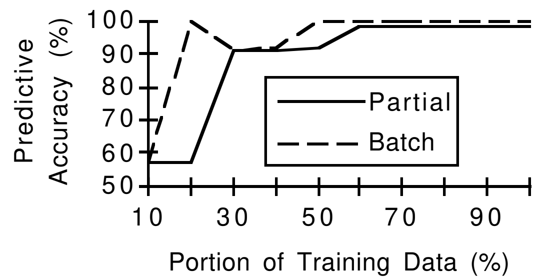

Figure 3. Learning curve comparison.

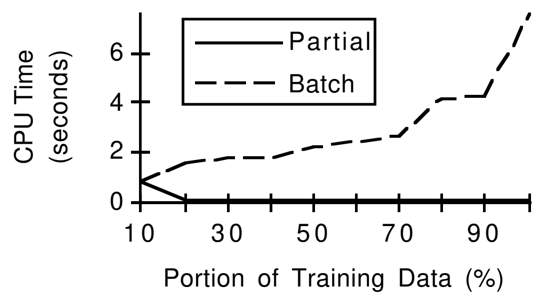

Figure 4. Learning times comparison.

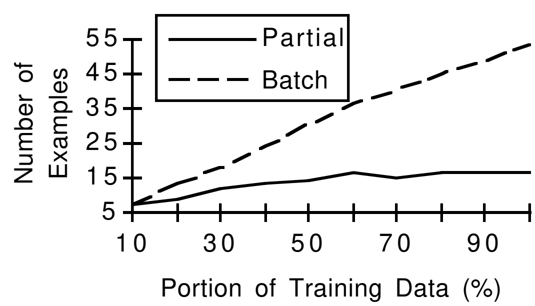

Figure 5. Memory requirements comparison.

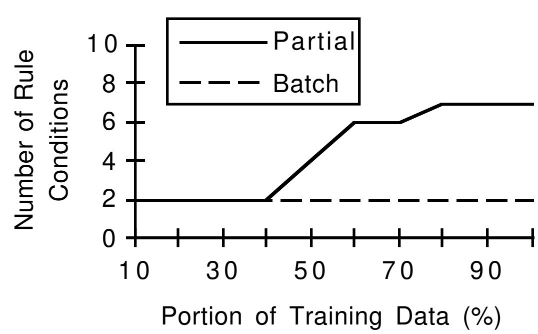

Figure 6. Rule complexity comparison.

best performing rule set that achieved a predictive accuracy of 96% on testing data. The rule that characterizes daffy's computer use is simple, consisting of one complex and one condition. The rule says that if the maximum real time for a user is equal to 16, then this user is daffy. Note that the value 16 represents a range of values because of ChiMerge scaling. The t-weight and u-weight describe the strength of the complex. The t-weight indicates the total number of training examples this rule covers, while the u-weight indicates how many of those examples are unique (i.e., not covered by other complexes).

```
daffy-rule
#  complex
1  [maxreal=16]    (t-weight:16, u-weight:16)

elmer-rule
#  complex
1  [maxsys=8..18] & [maxchar=10..20]
   (t-weight:7, u-weight:2)
2  [maxreal=0..11] & [avgblks=14..48] &
   [avgcpu=1..26] & [avghog=13..35]
   (t-weight:6, u-weight:1)
```

Figure 7. AQ15c rules for daffy and elmer.

The rule for elmer's use, on the other hand, is more complex, consisting of 2 complexes and a total of 7 conditions. The first complex, for example, covers a total of 7 training examples, but only two of those examples are unique. The second complex covers a total of 6 examples, but only one of these is unique. This implies that there is great overlap among these two complexes. Referring again to the first complex of elmer's rule, in order for this complex to be true, under strict matching conventions, the maximum system time must be inclusively between 8 and 18, and the maximum number of characters transferred must be inclusively between 10 and 20. Again note that these discrete intervals relate to much larger real ranges, but were abstracted by the ChiMerge algorithm. If these two conditions are true, then the symbol "elmer" will be assigned as the decision class. Otherwise, the second complex will be tested in much the same manner.

Experimental results suggest that partial-memory incremental learning is beneficial. Although these experiments demonstrate that partial-memory incremental learning resulted in slightly higher rule complexity and slightly lower predictive accuracy than batch learning, significant decreases were seen in the CPU time spent learning and in the number of examples maintained over time. Future work will be on the implementation of these ideas, which will allow larger experiments to determine how this approach scales up to more complex problems.

As an intrusion detection system, clearly more behavioral factors need to be included into this system to increase viability and predictive accuracy. Future work will involve incorporating symbolic data (e.g., terminal names, command names) and structured data (e.g., command hierarchies), and if temporal trends exist (e.g., daffy always works in the afternoon and evening, but not in the morning), investigating how they be exploited to yield higher predictive accuracy.

5 Conclusions

This paper describes a partial-memory incremental learning method and applies it to the problem of computer intrusion detection. The proposed incremental learning architecture and method is especially useful in such applications as intelligent agents working in dynamic environments, active vision, or computer intrusion detection. This is because of the need for the system to interact with users and the environment, and adapt their behavior to the changing conditions. The described method for partial-memory incremental learning yielded significant improvements over batch learning in terms of the number of examples maintained and the learning time, at the cost of slightly lower predictive accuracy and higher rule complexity.

Acknowledgments

The authors thank Eric Bloedorn for many discussions on this work, reading drafts of the paper, and recommending Davis's work on characterizing time series. Thanks also goes to Dorothy Denning for providing pointers to SRI's work on intrusion detection. The authors greatly appreciate the support of a Doctoral Fellowship from the School of Information Technology and Engineering at George Mason University awarded to the first author. This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research is supported in part by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, administered by the Office of Naval Research, and the Grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, in part by the Office of Naval Research under Grant No. N00014-91-J-1351, and in part by the National Science Foundation under Grants No. IRI-9020266 and DMI-9496192.

References

[1] R. S. Michalski, "On the quasi-minimal solution of the general covering problem," *Fifth International Symposium on Information Processing*, vol. A3, pp. 125–128, 1969.

[2] R. S. Michalski, "A variable-valued logic system as applied to picture description and recognition," *IFIP Working Conference on Graphic Languages*, pp. 21–47, 1972.

[3] R. S. Michalski, "AQVAL/1 — computer implementation of a variable-valued logic system VL_1 and examples of its application to pattern recognition," *First International Joint Conference on Pattern Recognition*, 3–17, 1973.

[4] R. S. Michalski, "Pattern recognition as rule-guided inductive inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, no. 4, pp. 349–361, 1980.

[5] R. E. Reinke and R. S. Michalski, "Incremental learning of concept descriptions: a method and experimental results," *Machine Intelligence 11*, J. E. Hayes, D. Michie and J. Richards (Eds), Clarendon Press, Oxford, pp. 263–288, 1988.

[6] J. Hong, I. Mozetic and R. S. Michalski, "AQ15: incremental learning of attribute-based descriptions from examples, the method and user's guide," *UIUCDCS-F-86-949*, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

[7] M. K. Bhandaru and M. N. Murty, "Incremental learning from examples using HC-expressions," *Pattern Recognition,* vol. 24, no. 4, pp. 273–282, 1991.

[8] P. Maes, "Agents that reduce work and information overload," *Communications of the ACM,* vol. 37, no. 7, pp. 31–40, 1994.

[9] D. Ballard and C. Brown, "Principles of animate vision," *Active Perception*, Y. Aloimonos (Ed), Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 245–282, 1993.

[10] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, 1987.

[11] S. E. Smaha, "Haystack: an intrusion detection system," *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pp. 37–44, 1988.

[12] T. F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, and S. Listgarten, "Knowledge-based intrusion detection," *Proceedings of the Annual Artificial Intelligence Systems in Government Conference*, pp. 102–107, 1989.

[13] H. S. Vaccaro, "Detection of anomalous computer session activity," *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, pp. 280–289, 1989.

[14] D. Anderson, T. Frivold and A. Valdes, "Next Generation Intrusion Detection Expert System (NIDES): final technical report," *SRI International Final Technical Report A008*, SRI International, Menlo Park, CA, 1994.

[15] M. A. Maloof and R. S. Michalski, "A partial memory incremental learning methodology and its application to intrusion detection," *Reports of the Machine Learning and Inference Laboratory*, *MLI 95–2*, Department of Computer Science, George Mason University, Fairfax, VA, 1995.

[16] J. Wnek, K. Kaufman, E. Bloedorn and R. S. Michalski, "Selective Induction Learning System AQ15c: the method and user's guide," *Reports of the Machine Learning and Inference Laboratory, MLI 95–4*, Machine Learning and Inference Laboratory, Department of Computer Science, George Mason University, Fairfax, VA, 1995.

[17] H. S. Teng, K. Chen, and S. C-Y. Lu, "Security audit trail analysis using inductively generated predictive rules," *Proceedings of the Sixth Conference on Artificial Intelligence Applications*, pp. 24–29, 1990.

[18] H. S. Javitz and A. Valdes, "The NIDES Statistical Component: description and justification," *SRI International Annual Report A010*, SRI International, Menlo Park, CA, 1994.

[19] J. H. Davis, "CONVART: a program for constructive induction on time dependent data," *Master's Thesis*, Department of Computer Science, University of Illinois, Urbana, IL, 1981.

[20] E. Bloedorn, J. Wnek, R. S. Michalski and K. Kaufman, "AQ17 — A multistrategy learning system: the method and user's guide," *Reports of the Machine Learning and Inference Laboratory, MLI 93–12*, Machine Learning and Inference Laboratory, Department of Computer Science, George Mason University, Fairfax, VA, 1993.

[21] R. Kerber, "ChiMerge: discretization of numeric attributes," *AAAI-92*, pp. 123–128, 1992.

[22] J. R. Quinlan, "Learning efficient classification procedures and their application to chess end games," *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds), Tioga Publishing, Palo Alto, CA, vol. 1, pp. 463–482, 1983.

[23] P. W. Baim, "Automated acquisition of decision rules: the problems of attribute construction and selection," *Master's Thesis*, Department of Computer Science, University of Illinois, Urbana, IL, 1984.