Comparing Instance-Averaging with Instance-Saving Learning Algorithms

Dennis Kibler
David W. Aha
Irvine Computational Intelligence Project
Department of Information and Computer Science
University of California, Irvine, CA 92717

Technical Report 88-06

23 March 1988

Comparing Instance-Averaging with Instance-Saving Learning Algorithms

Dennis Kibler David W. Aha

Department of Information & Computer Science
University of California, Irvine
Irvine, CA 92717 U.S.A.
(714) 856-8779

Problem Area:	Concept Learning
General Approach:	Instance-Based Algorithms
Evaluation Criteria:	Empirical and Theoretical

Abstract

The goal of our research is to understand the power and appropriateness of instance-based representations and their associated acquisition methods. This paper concerns two methods for reducing storage requirements for instance-based learning algorithms. The first method, termed instance-saving, represents concept descriptions by selecting and storing a representative subset of the given training instances. We provide an analysis for instance-saving techniques and specify one general class of concepts that instance-saving algorithms are capable of learning. The second method, termed instance-averaging, represents concept descriptions by averaging together some training instances while simply saving others. We describe why analyses for instance-averaging algorithms are difficult to produce. Our empirical results indicate that storage requirements for these two methods are roughly equivalent. We outline the assumptions of instance-averaging algorithms and describe how their violation might degrade performance. To mitigate the effects of non-convex concepts, a dynamic thresholding technique is introduced and applied in both the averaging and non-averaging learning algorithms. Thresholding increases the storage requirements but also increases the quality of the resulting concept descriptions.

1 Introduction

This paper addresses representational issues for the *learning from examples* task. The object of this task is, given a set of training examples annotated with concept membership information, to yield a description for each concept represented by at least one of the training examples. Representations based on rules, first order logic, and decision trees have received significantly more attention in the machine learning literature than have other, even simpler methods for describing concepts. This paper explores the use of instances, either observed or constructed, to represent concepts.

Instance-based learning (IBL) algorithms are given a *training set* of instances and derive a *concept set* from them. The concept set can then be used, along with the chosen *classification metric* (similarity function), to describe concepts.

Classification metrics also determine how each training instance influences the representation of the concept. While the learning algorithms' efficiencies can degrade when employing a poor classification metric, it is not obvious that the choice of the metric influences which concepts can be learned. For this reason, we employ only the simple nearest-neighbor classification algorithm in our experiments. Nearest-neighbor classifications yield concepts which have a piecewise-linear boundary, enabling simpler analysis. Distinguishing the strengths and weaknesses of different classification metrics is a future research problem.

Metrics for measuring the performance of IBL algorithms include *generality* (the class of concepts which are learnable by the algorithm) and the resulting concept set's *accuracy* (in mapping instance space to concept space), number of misclassifications (measured by its number of false positives and false negatives), and *storage requirements* (size).

IBL techniques have not received a great deal of attention in the ML literature, in part because they are presumed to be storage intensive. Two approaches have been used in response to this problem. Stanfill and Waltz (1986) suggest using large numbers of multiprocessors to significantly reduce memory limitation problems. Others have researched methods for reducing storage requirements through either *instance-*

averaging techniques (Sebestyen, 1962; Kohonen, 1986; Bradshaw, 1987) or by selectively saving some of the training instances (Kurtzberg, 1987; Kibler & Aha, 1987; Connell & Utgoff, 1987). Several natural domains require surprisingly few instances to be saved in order to attain high classification accuracies on new instances. Section 2 describes a family of IBL algorithms that attempt to reduce storage requirements without sacrificing predictive abilities. Section 2 also introduces the notion of *thresholding*, which suggests that instances which are far from previously observed instances should be saved, even if they would be correctly classified.

Section 3 describes an analysis of some of the simpler IBL algorithms, showing when they are guaranteed to succeed. Section 4 presents an empirical comparison of the IBL algorithms introduced in Section 2 and discusses the advantages and problems involved when using instance-averaging and thresholding techniques.

2 A Family of Instance-Based Learning Algorithms

In this section we discuss a family of incremental IBL algorithms. They all assume a representation in which each instance is defined by a finite set of real-valued attributes. All the algorithms also assume that a new disjunct is formed when a training instance is misclassified by the current concept set. Consequently, the misclassified training instance is added to the concept set. Thus we will refer to them as error-driven, disjunct-learning (EDDL) algorithms.

Two pairs of EDDL algorithms are described in this section. The algorithms differ in terms of whether they employ instance-averaging and/or adaptive thresholding techniques. Algorithms that *don't* average instances simply discard correctly classified training instances. Instance-averaging algorithms replace the correctly classified instance's nearest neighbor in the current concept set with a weighted-average of the two instances. Algorithms that *don't* employ thresholding techniques classify new training instances in terms of whether they are in the same concept as their nearest neighbor in the current concept set. Thresholding algorithms additionally require that the nearest neighbor be within a specified distance (threshold) of the new instance. The general

```
1     LEARN GUESSES FOR THRESHOLDS
2     ∀ training instances $t \in T$:
      2.1  Find $nn$, the nearest neighbor of $t$ in $C$
      2.2  IF ($t$ is classified correctly by $nn$)
      2.3  AND (DISTANCE($t$,$nn$) < THRESHOLD(CLASS($t$)))
           2.4  THEN REPLACE $nn$ IN $C$ WITH WEIGHTED-AVERAGE($t$,$nn$)
           2.5  ELSE add $t$ to $C$
      2.6  UPDATE THRESHOLD GUESSES
```

Figure 1: EDDL learning algorithms: deriving concept set C from training set T. Lines 1, 2.3, and 2.6 refer only to thresholding algorithms. Line 2.4 refers only to averaging algorithms.

Learning Algorithm	Options Employed		Corresponding Lines
	Averaging	Thresholding	in Figure 1
Growth			2,2.1-2,2.5
Growth + AT		✓	all but 2.4
Disjunctive Spanning (DS)	✓		2,2.1-2,2.4-5
DS + AT	✓	✓	all

Table 1: Distinguishing the behaviors of the four EDDL algorithms.

skeleton of the four EDDL algorithms is given in Figure 1. Operations involving thresholding and instance-averaging techniques are distinguished (preceded by bold-faced line numbers) in the figure. The two pairs of EDDL algorithms to be discussed are summarized in Table 1 with respect to whether they employ instance-averaging and/or adaptive thresholding (AT) techniques.

The simplest algorithm presented here, which we refer to as the *growth* algorithm, discards correctly classified training instances and saves misclassified training instances in the current concept set. The *disjunctive spanning* algorithm, introduced by Bradshaw (1987), averages correctly classified training instances. Each instance is associated with a weight (initialized to 1). When training instance t is classified correctly by concept instance nn, the disjunctive spanning algorithm replaces nn with the weighted average of t and nn. The weight of the averaged instance is defined as the sum of the weights of nn and t. Thus instances correctly classified by concept instance nn affect the relocation of nn in C as an inverse function of nn's weight and as a function of their distance from nn.

Bradshaw (1987) reported that instance-averaging in the disjunctive spanning algorithm is unsafe in the following sense. An averaged "instance" not only might be non-prototypical, but might not even be an instance of concept it was to represent. Sebestyen (1962) also recognized this problem and attempted to solve it by introducing an explicit, predefined threshold. Sebestyen's algorithm averaged a new instance only if it was correctly classified and it was within the threshold distance of some instance in the current concept set with the same classification.

While Sebestyen's algorithm describes how to use a threshold, no principled method for setting the threshold was suggested. The choice for Sebestyen's threshold significantly affects his algorithm's capability to learn concepts. Therefore we have developed an *adaptive* thresholding technique for both the growth and disjunctive spanning algorithms, denoted by growth+AT and DS+AT respectively.

The thresholding algorithms learn one threshold for *each concept* in the instance space.[1] A threshold for concept C is defined as the shortest distance between a pair of instances in C that appear to lie in different disjuncts of C (i.e. they are intervened by at least one instance in another concept). All thresholds are initialized to unreasonably large values and are recomputed after each training instance is incorporated. Training instances are simply stored until the thresholds for each concept have stabilized. A threshold for concept C stabilizes when it has not changed by more than $P\%$ during the incorporation of the last T training instances. (In the experiments with the thresholding algorithms, P and T were assigned the values 5 and 10 respectively.) The learning algorithm is applied to the entire training set after the thresholds have stabilized. The intuition here is to determine a good guess of the thresholds by sampling the training set until the guesses tend to stabilize.

The DS+AT algorithm further reduces the probability of yielding misclassifications in the concept set by employing an instance-averaging function which weights old instances more heavily than new ones. For each saved concept instance, Bradshaw's (1987) algorithm first averages liberally and quickly becomes extremely conservative.

[1]Ideally, a thresholding algorithm should employ a *set* of "local" thresholds for each concept. The intuition here is that a concept's threshold is not the same throughout the instance space; it would be large in "core" areas and low at "boundary" areas of concept disjuncts. The simpler method is used in this paper because this ideal approach adds undue complexity for our present purposes.

In contrast, DS+AT uses an averaging algorithm (Kohonen, 1986) that gradually decreases the effects of averaging. Kohonen's algorithm ensures that concept instances are never *quickly* averaged far away from their initial location in instance space and that all correctly classified training instances have a non-trivial impact on the formation of the concept set.

3 Convergence of Instance-Saving Algorithms

In this section we will examine the issue of generality to determine which assumptions guarantee the correctness of the learning algorithms introduced in Section 2. Clearly none of the algorithms is guaranteed to always work. In particular none of these algorithms would learn the concept of even numbers given positive and negative instances of numbers. One basic assumption that IBL algorithms share is that if x is close to y and x is a member of some concept C then y is a member of C. IBL algorithms that average additionally assume that the underlying concept is convex. A violation of this assumption suggests that the very process of averaging could result in an instance located outside the concept class.

Cover and Hart (1967) demonstrated that, under very general statistical assumptions, the nearest neighborhood decision policy has a Bayes Risk of at most twice the optimal decision policy. This result is partially weakened by the fact that it requires an unbounded number of samples. In contrast to statistical assumptions, we make *geometric* assumptions about the shape of the concept to be learned. We are then able to show that the nearest neighbor (proximity) and growth algorithms converge, in reasonable time, if the concept has a "nice" boundary (i.e. the boundary has a finite size and separates the concept's interior from its exterior).

We need a few definitions for the analysis. For any $\epsilon > 0$, let the *ϵ-core* of a set C be all those points of C which have an ϵ-ball about them contained in C. Similarly we define the *ϵ-neighborhood* of C to be all those points which are within ϵ of some point of C. If the set of points C' contains the ϵ-core of C and is contained in the ϵ-neighborhood of C, then C' is an *ϵ-approximation* of C. Finally, if the ϵ-neighborhood

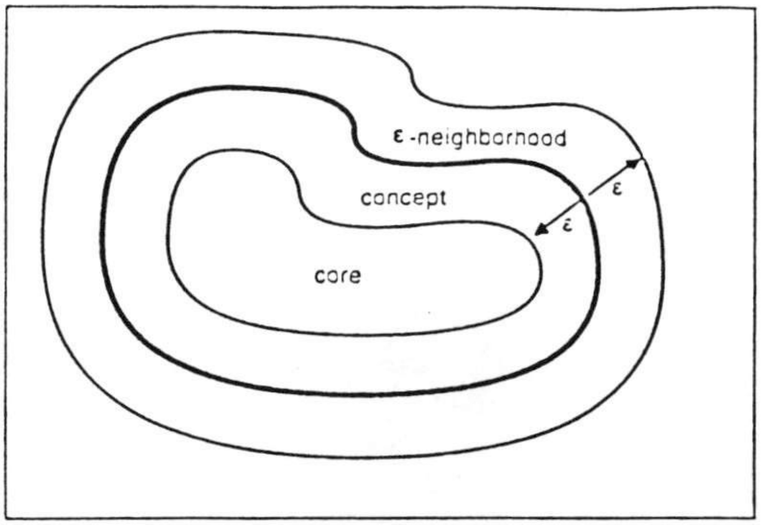

Figure 2: Parts of a concept.

of a finite set of points F contains the entire space, then that set is an *ϵ-net* for the space. The diagram in Figure 2 illustrates some of these concepts.

We first establish that the proximity algorithm, which saves every instance, nearly always converges to an approximately correct definition of a concept when the boundary of the concept is sufficiently "nice". "Nearly always" means with probability greater than $1 - \delta$, where δ is an arbitrarily small positive number. "Approximately correct" means that the generated concept is an ϵ-approximation of the actual concept, where ϵ is an arbitrarily small positive number.

For simplicity we will establish the theorem for any finite polygon in a bounded region in the plane.

Theorem 1. *Let C be any finite polygon with boundary length L in the unit square. Given an $\epsilon > 0$, then the* proximity *algorithm will (approximately) learn C (with confidence $1 - \delta$).*

Proof: Let ϵ and δ be arbitrary positive numbers. A mild extension of our previous result (Kibler and Aha, 1988) yields: if $N > (1/\epsilon^2)ln(1/\delta)$, then any N randomly-selected samples will form an ϵ-net (with confidence $1 - \delta$) for C. Now let p be any point of the unit square and let s be the closest point to p in the sample set. By construction, we are guaranteed that s is within ϵ of p.

There are three cases to consider.

1. s is in the ϵ-core of C. Since the distance between s and p is less than ϵ and s is in the ϵ-core, p is also in C. Consequently s correctly predicts that p is a member of C with confidence $1 - \epsilon$.
2. s is outside the ϵ-neighborhood of C. Since s is within ϵ of p, then p is also outside of C. In this case we also have that s correctly predicts that p is not a member of C.
3. s is within the ϵ-neighborhood but outside the ϵ-core of C. This is the only case in which s may incorrectly predict whether p is a member of C or not. Then we can bound the size of the set of values on which this algorithm makes errors with $2\epsilon L$.

Since L is a fixed number, this shows that we can get arbitrarily close approximations to the concept C. ■

This proof allows us to conclude a number of qualitatively important statements.

- The set of false positives is contained in the outer ribbon (the ϵ-neighborhood of C excluding C). Similarly, the set of false negatives is contained in the inner ribbon.
- The algorithm will not distinguish a concept from anything containing the ϵ-core and contained in its ϵ-neighborhood. Consequently small perturbations in the shape of a concept are not captured by this approach.
- No assumptions about the convexity of the concept, the connectedness (number of components or disjuncts) of the concept, nor the relative positions of the various components of the concepts need be made.
- The argument did not depend on the unit square, but rather on the "niceness" of the boundary. The proof could be generalized to finite-sized polyhedron in arbitrary Euclidean space.

We now show that the growth algorithm will also learn concepts with "nice" boundaries.

Theorem 2. *Let C be any finite polygon with boundary length L in the unit square. Given an $\epsilon > 0$, then the* **growth** *algorithm will (approximately) learn C (with confidence $1 - \delta$).*

Proof: As before choose N so large that we are guaranteed (with confidence $1-\delta$) to have an ϵ-net. Note that the proximity algorithm would store N instances to represent the concept and its complement. The growth algorithm stores some subset of these instances. If the resulting concept description is an ϵ-approximation of C, then we are done. If it is not, then another pass through the *same* N instances will add at least one instance to our representation. If we repeat this process at most N times, then we will be guaranteed that it converges to the concept. ∎

The reader may note that we have, perhaps unfairly, assumed that the algorithm can re-examine the same N instances. We believe that the proof would hold without this assumption, but so far have been unable to produce the appropriate argument.

In practice, the growth algorithm tends to add points that are near the boundary. As one can see, if enough border points are selected, the core points become extraneous (in the sense that any point in the core would be correctly classified). In fact, as we will demonstrate in Section 4, the number of instances saved in the concept set is linearly proportional to the concept's boundary length.

The growth algorithm with thresholding converges by the same proof.

The above results give a good characterization of the classes of concepts learnable by instance-saving algorithms. The situation for instance-averaging algorithms appears to be more complex. Bradshaw (1987), Sebestyen (1962), Kohonen (1986), and several others have demonstrated that instance-averaging techniques work in real domains. We have been unable, however, to find any reasonable constraints on the concept shape that would guarantee such convergence.

Let us consider a few examples. Suppose that the concept to be learned was a

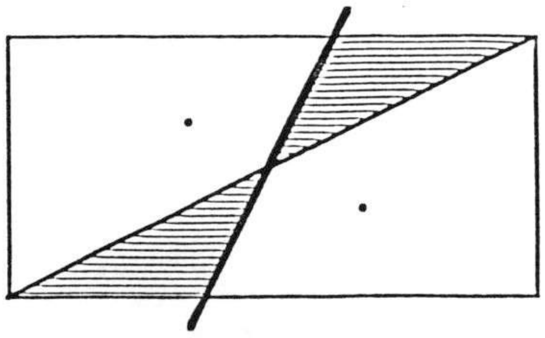

Figure 3: An instance-averaging example: error sets for 2 convex concepts.

ring and suppose we were given only positive examples. Then instance-averaging approaches would converge to the center of mass (centroid) of the ring. Thus they would converge to a point which was not even a member of the concept! Although this is an extreme example, whenever a concept is not convex there is the possibility that an instance-averaging algorithm will converge to a point not in the concept. Therefore one reasonable constraint might be that instance-averaging algorithms will converge if the concept is convex. Even this strong constraint, however, is not sufficient.

Consider the rectangular instance space in Figure 3. The space consists of two triangularly-shaped concepts. If the training set is ordered so that all positive instances precede all negative instances, then instance-averaging algorithms will converge to the centroids (shown in the figure). The shaded area would then represent the error set for instance-averaging. Examples like these leave us in a quandary. Instance-averaging algorithms work; when they do remains a mystery to us.

4 Performance of EDDL Algorithms

Instance-averaging algorithms can yield erroneously classified "instances" in the concept set when one or more of the algorithm's assumptions are violated. These algorithms are sensitive to the degree of convexity of the concepts, the distribution of instances across disjuncts in the training set, the ratio of a concept's area to its disjuncts' boundary lengths, and the ordering of instances in the training set.

We define the *degree of convexity* of a concept C as the probability that, given any two points of C, the line segment joining the two points is also contained in C. The

Instance Space	Instance Space Convexity	Total Errors in the Concept Set/Average Accuracy/Average Storage Requirements			
		Algorithm			
		Growth	Growth+AT	Disjunctive Spanning	DS+AT
1	largest	0/91%/7	0/93%/11	0/93%/6	0/94%/11
2	large	0/86%/11	0/88%/17	12/86%/11	1/89%/17
3	smaller	0/86%/11	0/88%/17	15/87%/10	0/89%/16
4	least	0/86%/11	0/88%/18	20/88%/11	0/89%/17

Table 2: Empirical results: varying instance space convexity.

probability that an averaging algorithm will yield concept set misclassifications varies inversely with the degree of convexity of the concept. For example, if all the concepts are convex, then all instance-averagings will be safe. Applying an averaging algorithm to an instance space with a low degree of convexity, however, might result with several concept set misclassifications, depending upon the particular training instances and their ordering of presentation.

Table 2 summarizes the application of the four EDDL algorithms to a set of four 2-dimensional, square instance spaces. Each instance space consists of two concepts, A and B, where A consists of a single, horizontal, rectangular disjunct and B consists of the remainder of the space. A's disjunct extends across the entire width of the space and its width is one-fourth the height of the space. The disjunct is located at the north boundary of the first instance space and is gradually shifted southwards until it is centered in the last (fourth) space. Thus the degree of convexity of B decreases dramatically from the first to last space. The results summarize 100 applications of the algorithms to each space, where each application used a 50-instance training and a (disjoint) 100-instance test set. All instances were randomly selected. As expected, the number of misclassifications yielded by the disjunctive spanning (DS) algorithm increased as the degree of convexity of the instance space decreased. The results show that the DS+AT algorithm resulted with far fewer erroneously classified concept set instances than did the DS algorithm. This can be attributed both to its thresholding and conservative averaging algorithms. Interestingly, all four algorithms had about the same accuracies on the test sets. The thresholding algorithms understandably saved more instances than their counterparts since their thresholds add conditions to correct

Percentage of Instances in Disjunct	Total Errors in the Concept Set/Average Accuracy/Average Storage Requirements			
	Algorithm			
	Growth	Growth+AT	Disjunctive Spanning	DS+AT
10	0/88%/9	0/89%/14	36/88%/9	0/90%/14
25	0/86%/11	0/88%/18	20/88%/11	0/89%/17
50	0/91%/12	0/92%/18	7/91%/11	0/92%/17
90	0/72%/6	0/70%/10	1/74%/6	0/72%/9

Table 3: Empirical results: varying the distribution of training set instances.

classifications and all misclassifications result in additions of training instances to the concept set. The growth algorithm's simplicity suggests that it is the best algorithm to use for these instance spaces.

Table 3 summarizes applications to the fourth (centered disjunct) instance space during which the distribution of 50 training instances among concepts was varied. The percentage of training instances in A was set at 10%, 25%, 50%, and 90%. Each algorithm was applied 100 times to each distribution setting. Again, the non-thresholding averaging algorithm resulted with several erroneously classified concept set instances and their number decreased as the distribution favored A. The latter is expected since A is a convex concept and B is not. Therefore as the percentage of training instances in B decreases, there will be fewer opportunities to erroneously average two of B's instances to yield a location in A's disjunct. Note that the accuracies (on 100 randomly selected, disjoint sets of test instances) are again relatively equal across the algorithms and the thresholding algorithms have higher storage requirements.

The ratio of the boundary length of a concept disjunct to its area, as explained in Section 3, also affects the accuracy and number of instances saved by the growth algorithm. We experimented with four 2-dimensional instance spaces. Each had a single, centered, disjunct whose area remained constant across the four spaces. The shape of the disjunct, however, was varied so that its boundary length/area ratio increased monotonically in the four spaces. Each algorithm was applied to each space 25 times. The experiments employed 100-instance training sets with 25% of the instances in the lone disjunct. (The disjunct's area was 4.5% of the instance space.) Test sets contained 100 (disjoint) randomly selected instances. Figure 4 plots the average number of in-

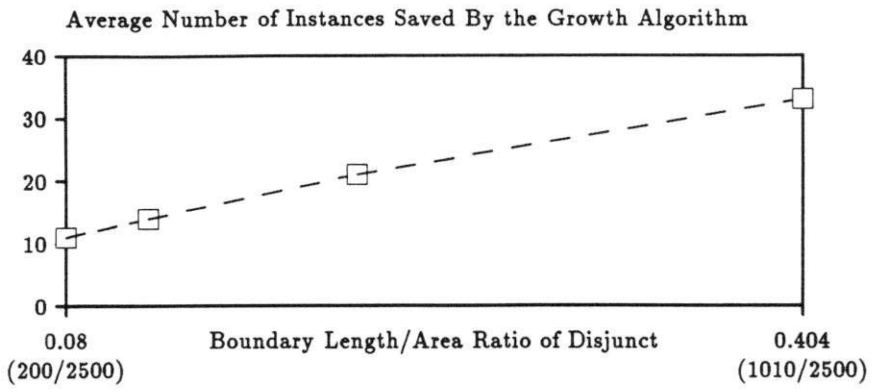

Figure 4: Storage requirements as a function of the boundary length/area ratio of a single disjunct.

Instance Space	Disjunct		Total Errors in Concept Set/Average Accuracy/Average Storage Requirements			
	Boundary Length	Area	Algorithm			
			Growth	Growth+AT	Disjunctive Spanning	DS+AT
1	200	2500	0/98%/11	0/98%/27	1/96%/10	1/98%/23
2	290	2500	0/96%/14	0/98%/33	1/97%/14	0/98%/30
3	520	2500	0/93%/21	0/96%/43	11/93%/23	2/96%/41
4	1010	2500	0/88%/33	0/93%/51	10/87%/34	2/94%/52

Table 4: Empirical results: varying the boundary length/area ratio of a single disjunct.

stances saved by the growth algorithm as a function of the boundary length/area ratio of each space's disjunct. As anticipated in Section 3, the number of instances saved increases as a linear, monotonically increasing function of the boundary length/area ratio of the disjunct being learned. Thus the number of instances saved by the growth algorithm increases as the ratio of core/boundary length decreases.

In fact, this behavior occurred for all four algorithms. The results are summarized in Table 4. Note that the thresholding technique "pays off" for the fourth (thinnest disjunct) instance space in that the average accuracies for the thresholding algorithms are higher than those of the non-thresholding algorithms. Thresholding algorithms are usefully applied to instance spaces that contain narrow disjuncts and/or parts of disjuncts.

It is useful to experiment with the EDDL algorithms in artificial instance spaces in order to observe their behaviors resulting from the variance of a domain-dependent variable. Unfortunately, the information gained from these experiments may not apply to real-world databases. We plan to observe and report on real-world applications of these algorithms in the future while addressing problems such as noise, incomplete information, and irrelevant attributes.

5 Conclusions

We have shown that instance-saving algorithms can learn a large class of reasonable concepts, namely those with "nice" boundaries. We are unable to produce a similar statement about instance-averaging algorithms. The growth algorithm, in both our experiments with and without thresholding, stored about the same number of instances as did the corresponding averaging algorithm. Furthermore, this number is proportional to the ratio of concept boundary length to concept area. Both techniques achieved about the same accuracy on random test sets although the instance-averaging algorithms can yield false positives and false negatives in their concept set (a failing that instance-saving algorithms do not have). The experiments also showed that our adaptive thresholding techniques lessen the likelihood of storing misclassified instances but increase the number of instances saved. Superior thresholding algorithms, however, may not require significant increases in storage requirements. Finally, we have not addressed a number of important concerns. In particular, we have not considered weighting the strengths of different attributes, either singularly or in concert, nor discussed noise issues. We hope to address these issues in our future research.

References

Bradshaw, G. (1987). Learning about speech sounds: The NEXUS project. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 1–11). Irvine, CA: Morgan Kaufmann.

Connell, M. E., & Utgoff, P. E. (1987). Learning to control a dynamic physical system. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 456–460). Seattle, WA: Morgan Kaufmann.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *I.E.E.E. Transactions on Information Theory*, *13*, 21–27.

Kibler, D., & Aha, D. W. (1987). Learning representative exemplars of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 24–30). Irvine, CA: Morgan Kaufmann.

Kibler, D., & Aha, D. W. (1988). Instance-based prediction of real-valued attributes. To appear in *Proceedings of the Canadian Artificial Intelligence Conference*. Edmonton, Alberta.

Kohonen, T. (1986). *Learning vector quantization for pattern recognition* (Technical Report TKK-F-A601). Espoo, Finland: Helsinki University of Technology, Department of Technical Physics.

Kurtzberg, J. M. (1987). Feature analysis for symbol recognition by elastic matching. *I.B.M. Journal of Research and Development*, *31*, 91–95.

Sebestyen, G. S. (1962). *Decision-making processes in pattern recognition.* New York: The Macmillan Company.

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the Association for Computing Machinery*, *29*, 1213–1228.