Machine Learning, 6, 251-276 (1991)
© 1991 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

A Nearest Hyperrectangle Learning Method

STEVEN SALZBERG (SALZBERG@CS.JHU.EDU)
Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218

Editor: Tom Dietterich

Abstract. This paper presents a theory of learning called nested generalized exemplar (NGE) theory, in which learning is accomplished by storing objects in Euclidean n-space, E^n, as hyperrectangles. The hyperrectangles may be nested inside one another to arbitrary depth. In contrast to generalization processes that replace symbolic formulae by more general formulae, the NGE algorithm modifies hyperrectangles by growing and reshaping them in a well-defined fashion. The axes of these hyperrectangles are defined by the variables measured for each example. Each variable can have any range on the real line; thus the theory is not restricted to symbolic or binary values.

This paper describes some advantages and disadvantages of NGE theory, positions it as a form of exemplar-based learning, and compares it to other inductive learning theories. An implementation has been tested in three different domains, for which results are presented below: prediction of breast cancer, classification of iris flowers, and prediction of survival times for heart attack patients. The results in these domains support the claim that NGE theory can be used to create compact representations with excellent predictive accuracy.

Keywords. Exemplar, induction, generalization, prediction, incremental learning, exceptions

1. Introduction

This paper presents a theory of learning from examples called Nested Generalized Exemplar (NGE) theory. NGE theory is derived from a learning model called *exemplar-based learning* that was proposed originally as a model of human learning by Medin and Schaffer (1978). In the Medin and Schaffer theory, examples are stored in memory verbatim, with no change of representation. The set of examples that accumulate over time form category definitions; for example, the set of all chairs that a person has seen forms that person's definition of "chair" (see also (Osherson & Smith, 1981; Medin, 1983; Smith & Osherson, 1984; Barr & Caplan, 1987)). An example is defined within the NGE model as a vector of features values plus a label that represents the category of the example.

NGE theory makes several significant modifications to the exemplar-based learning model.

1. It retains the notion that examples should be stored verbatim in memory, but once it stores them, it allows examples to be *generalized*. In NGE theory, generalizations take the form of hyperrectangles in a Euclidean n-space, where the space is defined by the variables measured for each example. The hyperrectangles may be nested one inside another to arbitrary depth, and inner rectangles serve as *exceptions* to surrounding rectangles. (See Thornton (1987) for a hypercuboid learning model.)
2. Another significant change that NGE makes to exemplar-based learning is that of attaching weights to each hyperrectangle. These weights, which are modified extensively during the learning process, are used by the algorithm as described below. With respect to other

machine learning theories, these nested, weighted, generalized exemplars are a novel way to represent concepts.

3. NGE dynamically adjusts its distance function, which gives it a greater tolerance for noise in some situations. (Some algorithms use a *similarity function*, which is basically the opposite of a distance function.) The distance function determines the distance between a new example and all exemplars stored in memory.
4. NGE combines the uses of hyperrectangles (generalizations) with specific instances, in contrast to other models that use either one or the other form of representation.

The test of this theory, as with all empirical learning theories, is its performance on real data. Until we can develop provably correct learning algorithms, all machine learning algorithms must be subject to empirical verification. In particular, a learning theory should be compared to other theories by testing it on the same data sets. This paper presents results of the NGE algorithm using three different data sets, each of which was previously used in experiments with other learning theories. The application domains of the three data sets are (1) predicting the recurrence of breast cancer, (2) classifying iris flowers, and (3) predicting survival times for heart attack patients.

The program that implements NGE theory in this paper is called EACH, for Exemplar-Aided Constructor of Hyperrectangles. EACH makes predictions and classifications based on the examples it has seen in the past. The precise details of the EACH algorithm will be explained further below, but briefly: the learner compares new examples to those it has seen before, and finds the closest example in memory. To determine what is closest, a *distance measure* is used. This distance measure is modified by the program during the learning process. Once an example has been stored, it is termed an *exemplar*; this term is used henceforth specifically to refer to objects stored in computer memory. Over time, exemplars may be changed from points to hyperrectangles, as explained later. Stored with each exemplar is the value of the variable that the system is trying to predict. In the simplest case, the system predicts that the dependent variable for a new example will have the same value as that stored on the closest exemplar. Exemplars have properties such as weights, shapes, and sizes—all of which can be adjusted based on the results of the prediction. The learning itself takes place only after EACH receives feedback on its prediction. If the prediction was correct, EACH strengthens the exemplar used to make it. Strengthening can occur by increasing weight or size. If the prediction was incorrect, EACH weakens the exemplar. These processes, too, will be explained in detail later.

The experimental comparisons given below support the following claims:

1. The classification accuracy of the NGE model compares favorably to that of other machine learning algorithms.
2. EACH is superior to a "greedy" version of the program (described below) in memory requirements and the two versions are equivalent in terms of classification accuracy.
3. Feature weights are an important component of EACH's success.
4. EACH achieves substantial compression of the data sets.

These claims will be considered again in the experimental results section.

2. NGE theory as a kind of exemplar-based learning

Exemplar-based learning is a recent addition to the AI literature. However, based on the existing work, one can construct a hierarchy of different exemplar-based learning models. The position of Nested Generalized Exemplar theory in that hierarchy is shown in Figure 1.

Every theory in the family shown in Figure 1 shares the property that it uses verbatim examples as the basis of learning; in other words, these theories store examples in memory without any change in representation. From this initial theoretical position, however, several divergent models have already appeared. A model called "instance-based learning" retains examples in memory as points, and never changes them. The only decisions to be made are what points to store, where to put them, and how to measure distance. Kibler and Aha (1987, 1989) have created several variants of this model, and they are experimenting with how far they can go with strict point-storage models. All of their models use some kind of nearest neighbor technique to classify new inputs. Nearest neighbor algorithms, though a recent addition to the arsenal of machine learning techniques, have been explored in many forms in the pattern recognition literature, going back as far as the early work of Cover and Hart (1967).

Once a theory moves from a symbolic space to a Euclidean space, it becomes possible to nest generalizations one inside the other. The capability to do this nesting is a unique contribution of Nested Generalized Exemplar theory. Its generalizations, which take the form of hyperrectangles in Euclidean n-space E^n, can be nested to an arbitrary depth, where inner rectangles act as exceptions to outer ones. NGE theory is intended to encompass more than just hyperrectangles: it includes all theories that create nested generalizations of any shape from exemplars. Other shapes that might be the subject of future research are spheres, ellipses, and convex hulls. However, the nested hyperrectangles created by EACH are the only current implementation of NGE.

Another research track that fits into the exemplar-based learning paradigm is *case-based reasoning*. One of the first such systems was CYRUS (Kolodner, 1980; Kolodner & Simpson, 1984), a system that learned by reading (short) stories about Cyrus Vance, the former U.S. Secretary of State. It used stories as examples that it stored in an abstraction hierarchy. CYRUS generalized symbolic slot values, as did the later case-based Protos system (Bareiss, 1988). Learning in CYRUS was measured by improvements in its ability to read stories.

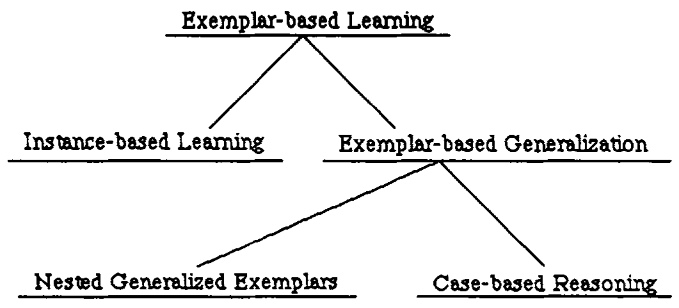

Figure 1. NGE as a type of exemplar-based learning.

Although the task was very different, the basic idea of saving examples for use in later processing is the same as that used by NGE. Exceptions are implicitly handled in CYRUS (and in other systems, such as Protos, that modify abstraction hierarchies) by the use of defaults in the abstraction hierarchy: when a default is overidden on a specific node in the hierarchy, that node can be considered an exception to more abstract nodes.

Case-based reasoning methods, and any other methods that use symbolic abstraction hierarchies (Ashley & Rissland, 1987; Rissland & Ashley, 1987; Bareiss, 1988), might reasonably be called *abstraction-based* methods. EACH clearly shares some features with these methods, especially in the basic approach that stores an example verbatim and later generalizes it. On the other hand, the strategy of using a distance metric to find the nearest points (*or* rectangles) in feature space clearly resembles other exemplar-based methods such as those of Aha and Kibler. These similarities illustrate that the boundary between abstraction-based and exemplar-based systems is not so clear, and EACH falls quite near this boundary. Fisher (1989) illustrates the similarities well by pointing out that case-based reasoning "is most productive when few training observations are available and noise is not present" (p. 829), but that abstractions will be required as more cases are observed. The alternative to abstraction is what Fisher calls "selective retention"—storing selected examples verbatim—which is just the strategy employed by Aha and Kibler (1989). Selective retention improves efficiency and accuracy, but presents new problems when dealing with irrelevant attributes. The general point is that although the learning techniques are quite different, case-based and exemplar-based learning share some fundamental principles and goals.

2.1. Generalizations with exceptions

Perhaps the most important distinguishing feature of NGE learning is its ability to capture generalizations with exceptions. The importance of such an ability has been remarked upon before:

> Programs that can only discover conjunctive characteristic descriptions have limited practical application. In particular, they are inadequate in situations involving noisy data or in which no single conjunctive description can describe the phenomena of interest. Consequently, as one of the evaluation criteria [of the systems under review], we consider the ease with which each method could be extended to . . . discover exceptions. (Dietterich & Michalski, 1983, p. 50)

Exemplar-based learning models do not, in general, create any generalizations, although virtually all other learning programs do. Among those programs that create generalizations, none represent generalizations with exceptions in the same way as NGE theory. Helmbold, Sloan, and Warmuth (1989) have developed an algorithm that uses a similar representation, but this algorithm is currently a theoretical model, not yet applicable to real world problems. Probably the best example of a learning model that explicitly handles exceptions is Vere's (1980) system for constructing multilevel counterfactuals. His system learned generalizations in the blocks world, in which the only legal relations were binary predicates such as (on a b) and (behind b c) and unary predicates such as (green a) and (pyramid c). The program produced a single description, disjunctive if necessary, that covered all positive examples and no negative examples. For example, it might produce a description such as: (on X1 table)$\wedge \neg$((on X2 X3)$\wedge$(on X3 X4)), where the negated clauses are exceptions. According to Vere, a typical expression contained counterfactuals (negated terms) nested several layers deep.

Vere's system differs from EACH in several significant ways: it only allowed symbolic features, it only handled noise-free data, its results applied only to two-category problems, and it was not incremental. An important result that it demonstrated was that by using negated terms (counterfactuals), one could create much more concise concept descriptions. These negated terms are conceptually similar to the exceptions created by EACH.

The generalizations created by any program that learns from examples are associated with a prediction or category. The fact that naturally occuring categories are not always easily characterized—in particular, the fact that natural categories have exceptions—has been a major problem for generalization algorithms. The NGE model, as implemented in EACH, explicitly creates exceptions by creating "holes" in its hyperrectangles. These "holes" are also hyperrectangles, and they can have additional exceptions inside them, nested as deep as the data require.

2.2. Problem domain characteristics

In addition to describing the advantages that NGE offers over other methods, it is also worthwhile to consider the kinds of problem domains it may not handle well. Although NGE (along with other exemplar-based learning models) is domain independent, there are some domains in which the target concepts are very difficult for NGE, and other learning techniques may perform better. In particular, exemplar-based learning is best suited for domains in which the exemplars are clusters (ideally, convex solids) in feature space, and where the behavior of the exemplars in a cluster is relatively constant (i.e., they all fall into the same category). Nested Generalized Exemplar learning works equally well when exemplars in a cluster have different behavior, because it can store clusters within clusters. On the other hand, if the exemplars are strung out along an infinite curve (for example), then the best description of the domain is the equation of that curve (which can be estimated using multiple regression analysis), rather than a set of exemplar objects. Although exemplar-based learning can handle such a domain, it will not create nearly as concise a description as a curve-fitting method will. Figure 2 illustrates this point. In the figure, we see that the exemplars listed in the "bad" plot for category *a* are scattered along a curve which could probably be approximated by a cubic function.

2.3. Related theoretical work

Very recent results in computational complexity dovetail nicely with the hyperrectangle memory model constructed by EACH. In particular, Helmbold, Sloan, and Warmuth (1989) have devised an algorithm that learns by constructing nested, axis-parallel hyperrectangles. Given an admittedly restrictive set of assumptions, Helmbold et al. prove some very strong

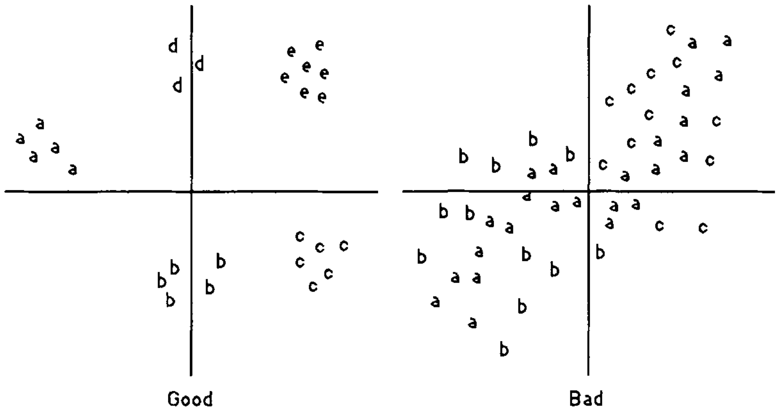

Figure 2. Good and bad structures for exemplar-based learning.

optimality results for their algorithm. In particular, they consider four optimality criteria: (1) the number of examples required to make accurate predictions with high confidence, (2) the probability of making a mistake on the nth instance, (3) the expected number of errors for the first m instances (using an incremental algorithm), and (4) the worst case number of errors for the first m instances. Their algorithm is asymptotically optimal with respect to all these measures.

Their learning algorithm applies to any intersection-closed concept class. Rectangles are intersection-closed, since the intersection of any two rectangles is also a rectangle. Monomials and subspaces of E^n are also intersection-closed. This constraint on their algorithm means that the "true" concepts in the class must be, for example, rectangular (this constraint does not apply to EACH). However, the representation constructed by the algorithm—a set of nested rectangles—is not constrained to be intersection-closed.

3. The NGE learning algorithm

This section describes the details of the nested generalized exemplar learning algorithm used by EACH. For the sake of brevity, descriptions of the data structures and minor subroutines have been omitted. A pseudo-code summary of the algorithm appears as an appendix. Following the description of the algorithm is an analysis of how the NGE algorithm partitions feature space.

3.1. Initialization

In order to make predictions, EACH must have a history of examples on which to base its predictions. Memory is initialized by "seeding" it with a small set of examples (the minimum size of this set is one). The seeding process simply stores each example in memory without attempting to make any predictions. These examples are chosen at random from the training set (as long as the examples are presented in random order, the system can use the first

N instances from the training stream as seeds). The number of seeds was determined by trial and error on a simulated data set (where an unlimited number of examples were available), and performance was not found to be sensitive to the size of the seed set.

An example is a vector of features, where each feature may have any number of values, ranging from 2 (for binary features) to infinity (for real-valued features). For example, the echocardiogram examples were patients who had recently suffered acute heart attacks, each patient being described by a vector of six variables. One of those six variables was binary, one was an integer ranging from 1 to 90, and the others were real-valued. In addition, each exemplar has a slot containing the category associated with that example. This category variable may be binary, discrete, or continuous, and the system will try to predict it accordingly.

When predicting continuous variables, the system uses an error tolerance parameter that indicates how close two values must be in order to be considered equivalent. This parameter is necessary because for real-valued variables, it is usually the case that no two values ever match exactly, and yet the system needs to know if its prediction was close enough to be considered correct. For example, if the user sets the error tolerance to 1%, and an exemplar e makes the prediction 5.0, then any value in the range [4.95, 5.05] will be considered a match. If a new example has the value 5.03 for its prediction variable, and this example matches e, EACH will not store a new point in memory. The result is that continuous predictions are approximated by a discrete set of values.

3.2. *Match and classify*

After initialization, the algorithm operates incrementally, processing one example at a time. Every new example is matched to memory using the matching process described below. Because memory has remained small in all the tests run so far, it has been acceptable to require the system to compare the new example to every object in memory. (For example, the echocardiogram data never required a memory for more than about 20 objects.) The best match is used for classification in the obvious way: the system predicts that the new example will fall into the same category as the best matched exemplar.

The matching process is one of the central features of the algorithm, and it is also a process that allows some customization, if desired. This process computes the distance between a new data point (an example) and an exemplar memory object (a hyperrectangle in E^n). For the remainder of this section, I will refer to the new example as E and the hyperrectangle as H.

The system computes a match score between E and H by measuring the Euclidean distance between the two objects. The simplest equation for this measurement assumes that H is a point. The distance is determined by the usual distance function computed over every dimension in feature space, with a few additions that are explained below. Specifically, the distance D_{EH} from E to H is calculated as:

$$D_{EH} = w_H \sqrt{\sum_{i=1}^{m} \left(w_i \frac{E_{f_i} - H_{f_i}}{max_i - min_i} \right)^2}$$

where w_H is the weight of the exemplar H, w_i is the weight of the feature i, E_{f_i} is the value of the *ith* feature on example E, H_{f_i} is the value of the *ith* feature on exemplar H, min_i, max_i are the minimum and maximum values of that feature, and m is the number of features recognizable on E.

The best match is the one with the smallest distance. A few special characteristics of the computation, which are not evident in the formula above, deserve mention here. First, let us suppose we measure the distance between E and H along the dimension f. Assume for simplicity that f is a real-valued feature. In order to normalize all distances so that one dimension will not overwhelm the others (every feature may have a different unit from every other, e.g., meters, seconds, years), the maximum distance between E and H along any dimension is 1. To maintain this property, the system uses its statistics on the maximum and minimum values of every feature. Suppose that for E, the value of f is 10, and for H, the value of f is 30. The unnormalized distance is therefore 20. Suppose further that the minimum value of f for all exemplars is 3, and the maximum value is 53. Then the total range of f is only 50, and the normalized distance from E to H along this dimension is 20/50, or 0.4. (Note: another way to normalize values is to use the standard deviation as one unit, and to measure distances in terms of that. However, if values for a dimension do not follow a normal distribution, the standard deviation is not an appropriate unit.)

Because the maximum and minimum values of a feature are not given *a priori*, the distance calculation will vary over time as these values change. This variation is a direct consequence of the incremental nature of the algorithm. If the maximum and minimum values are known ahead of time, then the distance calculation will not suffer this variation.

Now consider what happens when the exemplar, H, is not a point but a hyperrectangle, as is usually the case. In this case, the system finds the distance from E to the nearest face or edge of H. There are obvious alternatives to this, such as using the center of H (as with the centroid method in cluster analysis (Everitt, 1980)), but these lead to complications because the algorithm allows the nesting of exemplars inside one another. The formula used above changes because H_{f_i}, the value of the *ith* feature on H, is now a range instead of a point value. If we let H_{lower} be the lower end of the range, and H_{upper} be the upper end, then our equation becomes:

$$D_{EH} = w_H \sqrt{\sum_{i=1}^{m} \left(w_i \frac{dif_i}{max_i - min_i} \right)^2}$$

where

$$dif_i = \begin{cases} E_{f_i} - H_{upper} & \text{when } E_{f_i} > H_{upper} \\ H_{lower} - E_{f_i} & \text{when } E_{f_i} < H_{lower} \\ 0 & \text{otherwise} \end{cases}$$

The distance measured by this formula is equivalent to the length of a line dropped perpendicularly from the point E_{f_i} to the nearest surface, edge, or corner of H. This length is modified by the weighting factors, as described below. Note that points internal to a hyperrectangle have distance 0 to that rectangle. Furthermore, a point belongs only to the *innermost*

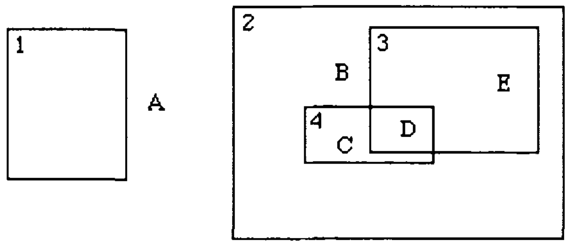

Figure 3. Matching of points to rectangles.

rectangle if it is internal to several nested rectangles. In the case of overlapping rectangles, a point falling in the area of overlap belongs to the *smaller* rectangle (this preference is merely a heuristic, based on the assumption that larger exemplars may have been overgeneralized). Figure 3 illustrates how the matching algorithm matches new points to rectangles. In the figure, point A is mapped to rectangle 1, B to rectangle 2, C to rectangle 4, D to rectangle 4, and E to rectangle 3.

For binary features, the distance computation is much simpler: if the features are equal, the distance is zero, else it is one. The same computation applies to any discrete, non-numeric features.

Notice that there are two weights on the distance metric. w_H is a simple measure of how frequently the exemplar, H, has been used to make a correct prediction. In fact, the use of this weight means that the distance metric measures more than just distance. w_H is a measure of the *reliability*, or the probability of making a correct prediction, of each exemplar. (In fact, w_H is an inverse measure, since the larger it is, the less reliable the exemplar is.) This weight measure says, in effect, "in this region of feature space, the reliability of my prediction is n," and of course EACH should prefer more reliable exemplars. The distance measure accomplishes this as follows. Suppose, in the above example, that H had been used for 15 previous predictions and that it had been correct on 12 of those occasions. The system will multiply the weight of the total match score between E and H by 15/12, or 1.25. Thus weight is a non-decreasing function of the number of times an exemplar has been used. If the exemplar *always* makes the correct prediction, then the weight will remain at 1. (Note that the seed exemplars do not get a weight of zero, because they are treated as if they had predicted themselves correctly; i.e., they are marked as having been used once and having been correct once.) More generally, if the weight of an exemplar is n/c, then when it is used to make an *incorrect* prediction, its weight increases from n/c to $(n + 1)/c$. If it is used to make a *correct* prediction, its weight will decrease to $n + 1/c + 1$. Note that if $n = c$, the weight will remain at 1 after a correct use of the exemplar H.

Also, as the number of correct uses of H increases, the effect of an incorrect use decreases: if H has been correct 100 times, and its weight is n/100, then an incorrect use makes the weight (n + 1)/100. This is an increase of 0.01 in absolute terms. If H has only been correct 10 times, its weight will increase from n/10 to (n + 1)/10, an increase of 0.1. In the former case the effect of an incorrect use is much smaller. This effect is desirable because as we know more about H, we do not want a single new example to change significantly our confidence in it.

Furthermore, noisy exemplars will gradually "disappear" as their weight w_H increases. If a point represents noise, then its prediction will rarely be correct for other points nearby. If such an exemplar is used 10 times, for example, but is only correct once, then its distance to new points will be multiplied by 10. New points will be much more likely to match some other point in memory than a noisy one. Very recently, Aha and Kibler (1989) have used a similar measure to create an instance-based learning system that tolerates noise. Their program saves statistics on the number of correct and incorrect classifications made by an exemplar, and only uses exemplars with a good record of classifications.

The other weight measure, w_i, is the weight of the *ith* feature. These weights are adjusted over time, as described below. Since the features do not normally have equal predictive power, they need to be weighted differently. In practice, the system performed best if these weights were adjusted for a fixed number of examples, and then locked in. When feature weight adjustment was allowed to continue indefinitely (on experiments not included here), the algorithm tended to oscillate, since adjustments in these weights can wipe out previous learning by negating the effects of previous adjustments. On the other hand, if EACH were applied to a domain that was gradually changing over time, weights should not be fixed.

A problem with these attribute weights is that they measure the relative importance of each attribute over the entire domain. If it turns out that an attribute is highly relevant over a small range of values but irrelevant for other values, the weight adjustment in the current algorithm cannot reflect this property. The utility of exemplar-specific attribute weights remains an open issue.

3.3. Feedback and learning

Learning only occurs when EACH gets feedback about its classification. The main feedback is simply whether or not the classification or prediction was correct.

Correct prediction. If EACH makes the correct prediction, it records some statistics about its performance and then makes a generalization. Two objects, E and H (using the same notation as above), are used to form the generalization. H is replaced in memory by a larger object (i.e., an abstraction) that is a generalization of E and H. H may have been a single point, or it may have been a hyperrectangle (after a single generalization, an exemplar becomes a hyperrectangle). If H was a hyperrectangle, then for every feature of E that did not lie within H, H is extended just far enough so that its boundary contains E. If H and E were both points, H is replaced by a new object that has, for each feature of E and H, a range of values defined by E and H. For example, in a simple case with just the two features f_1 and f_2, if E was at (2, 5) and H was a point at (3, 16), then the new object would be a rectangle extending from 2 to 3 in the f_1 dimension and from 5 to 16 in the f_2 dimension.

One consequence of this generalization procedure is that all hyperrectangles created by EACH are *axis-parallel* hyperrectangles, because they are not rotated by the algorithm. Another consequence is that growing an exemplar H1 may cause it to overlap an existing exemplar H2. As mentioned above, a point within the overlapping area belongs to the smaller exemplar.

Incorrect prediction. If the system makes the wrong prediction, it has one more chance to make the right one. This "second chance" heuristic (which is intended to be nothing more than a heuristic) is used by EACH in order to avoid creating more memory objects than necessary. The idea is to try very hard to make a generalization and thus keep down the size of memory. So, before creating a new exemplar, EACH first looks at the *second best match* in memory. Assume here that H_1 was the closest exemplar to E and H_2 was second closest; i.e., H_2 would be the closest if H_1 were removed. If H_2 will give the correct prediction, then the system tries to adjust hyperrectangle shapes to make the second closest exemplar into the closest exemplar. It does this by creating a generalization (using the process outlined in the previous section) from H_2 and E. The goal of this process is to improve the predictive accuracy of the system without increasing the number of exemplars stored in memory.

An interesting side note here is that the problem of creating an optimal number of (possibly overlapping) rectangles to classify a set of points is NP-hard. This proof of this result is via a reduction to a graph searching problem, achieved by laying a grid over all the points (Kasif, 1989).

A very important consequence of this "second chance" heuristic is that it allows the formation of hyperrectangles within other hyperrectangles. If a new point *p1* lies within an existing rectangle, its distance to that rectangle will be zero. Its distance to another point *p2* (a previously stored exception) within the rectangle will be small but positive. Thus EACH will first assume that the new point belongs to the same category as the rectangle. If *p1* is the same category as *p2*, then the second chance heuristic will discover this fact, and form a rectangle from these two points.

If the second best match also makes the wrong prediction, then the system simply stores the new example, E, as a point in memory. Thus E is made into an exemplar that can immediately be used to predict future examples, and can be generalized and specialized if necessary. This new exemplar may be *inside* an existing exemplar H, in which case it acts as an exception to, or "hole" in H.

EACH adjusts the weights w_i on the features f_i after discovering that it has made the wrong prediction. Weight adjustment occurs in a very simple loop: for each f_i, if E_{f_i} matches H_{f_i}, the weight w_i is increased by setting $w_i := w_i(1 + \Delta_f)$, where Δ_f is the global *feature adjustment rate*. A typical value used for Δ_f is 0.2. An increase in weight causes the two objects to seem farther apart, and the idea here is that since EACH made a mistake matching E and H, it should push them apart in space. If E_{f_i} does not match H_{f_i}, then w_i is decreased by setting $w_i := w_i(1 - \Delta_f)$. If EACH makes a correct prediction, feature weights are adjusted in exactly the opposite manner; i.e., weights are decreased for features that matched, which decreases distance, and increased for those that did not. Recall that each weight w_i applies uniformly to the entire feature dimension f_i, so adjusting w_i will move around exemplars everywhere in feature space. Thus this weight must be adjusted gradually, in order to avoid oscillation that would result from cancelling the effects of earlier learning.

3.4. Partitioning feature space

Using a Euclidean distance formula to determine distance essentially partitions the feature space between the hyperrectangles. Some analysis of how rectangular exemplars divide feature space is presented below (for a more detailed analysis, see (Salzberg, 1989)). With many dimensions and many rectangles (as in the experimental domains used for testing EACH), the partitioning is too complex to illustrate here; consequently, the discussion here is restricted to two dimensions and two rectangles. It should become clear that the NGE representation is equivalent to partitioning a space with many hypersurfaces, where the surfaces are not, in general, parallel to the axes—despite the fact that the rectangles themselves are always axis-parallel.

Figure 4 illustrates how two particular rectangles partition an unweighted feature space. The surface S that undulates between the two rectangles is simply the set of all points equidistant to the two rectangles, where distance is measured to the nearest edge or corner of a rectangle. In the center of the figure, for example, the surface is a vertical line segment equidistant from two edges of the rectangles. As we trace out the surface, we find it alternates between straight line segments and parabolic segments (parabolic segments are sets of points equidistant to a corner of one rectangle and an edge of the other). The surface in the figure has a total of nine segments, alternating straight lines and parabolas, with eight junction points where straight segments meet parabolic ones. Every junction is smooth, so S has a continuous derivative. Note, however, that the infinite straight line extensions

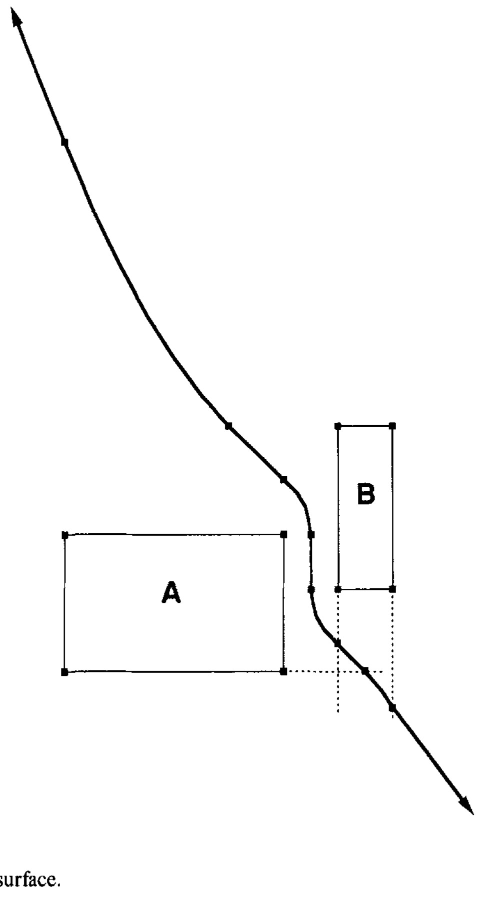

Figure 4. The separating surface.

at either end of the surface are neither collinear nor parallel. This partitioning of the plane is considerably more complex than the partitioning induced by a pair of points or circles.

EACH also attaches weights to each rectangle, which further complicates the shape of the surface. When hyperrectangles are nested, the inner rectangle is simply a sharp-edged hole in the outer one, so the distance measure does not come into play. In a domain for which EACH creates very closely packed hyperrectangles, it is possible that all new examples will fall inside existing hyperrectangles, and the partitioning of feature space will not be so important. (The iris flower domain, for example, was structured in this way.) This illustration should make it easier to compare EACH to learning algorithms that form different types of clusters (e.g., ellipsoids) or that explicitly partition space with hyperplanes.

3.5. Greedy variant of algorithm

A modification of the basic algorithm was developed in order to maximize the *post hoc* success rate on the training data. (*Post hoc* success refers to the success rate of the algorithm on examples it has already seen.) The idea behind this variant was that by storing more exemplars, the program would not only match the training data more closely, but would also exhibit superior classification performance on unseen test examples. One of the interesting results reported below is that storing more examples does not, in general, improve performance. This result corresponds to observations of Breiman et al. (1984), Quinlan (1986), and many others that overfitting a data set generally hurts performance on new data. The modified algorithm differs from EACH in that it always creates a new exemplar after making a mistake, as opposed to checking the second closest match and adjusting the boundaries of existing exemplars. Because it tends to create more exemplars than the original algorithm, and thus to be "greedy" in its use of memory, this version is called Greedy EACH.

The Greedy EACH algorithm was intended to be a compromise between the goal of creating a perfect *post hoc* model and that of creating useful generalizations. The Greedy EACH algorithm only creates (or increases the size of) a hyperrectangle when it makes a correct prediction. When it makes an incorrect prediction, it automatically stores a new point in feature space, without checking the second closest match. A significant implication of this rule is that Greedy EACH *cannot* create nested hyperrectangles: recall that a rectangle is created when two points match and make the same prediction. However, a new example e_i that falls inside an existing rectangle R will always be measured closer to R than to any exception point e_j inside the rectangle, even if it is very close to e_j. The distance measure will find a small positive distance between any two points e_i and e_j, but a zero distance between R and a point e_i within R. (It might be possible to modify the measure to allow the Greedy algorithm to find a match between e_i and e_j, but such modifications were not tested.) So, although the Greedy algorithm can create exceptions—by storing points within a rectangle—it cannot create nested rectangles.

4. Experimental results with EACH

The EACH program has tested using real data from three different problem domains: (1) predicting the recurrence of breast cancer, (2) classifying iris flowers, and (3) predicting survival for heart attack victims. The use of real data in these tests provided a measure of the system's accuracy on noisy and incomplete data sets, and, most importantly, allowed comparisons between EACH and other systems.

Below, each data set is described briefly, followed by a presentation of the experimental results. The main results are (1) summaries of EACH's performance rates and (2) summaries of the memory requirements. *Performance rate* is the performance level of the system during a test run, measured as the percentage of correct predictions or classifications. EACH is also compared to other learning models that were run on the same data. Different measures of performance and experimental designs were necessary on different experiments in order to compare EACH to previously published results in those cases.

For all the experiments, EACH was run with the feature adjustment rate Δ_f at 0.0 and 0.2.[1] Earlier trials using simulated data (Salzberg, 1988) indicate that 0.2 was a good value of Δ_f for small data sets (on the order of a few hundred examples), so that value is used throughout. The results below also include success rates for Greedy EACH. These different results are included to allow comparisons between EACH and Greedy EACH and to test the usefulness of feature weights.

4.1. Breast cancer data

The first problem domain for EACH is predicting the recurrence of breast cancer. The examples consisted of 273 patients who underwent surgery to remove tumors, all of whom were followed up five years later. The task for EACH was to predict whether or not breast cancer would recur during that five year period. The data set contains nine variables that were measured, including both numeric and binary values. The dependent variable was a binary prediction: either the patient did suffer a recurrence of cancer, or she did not.

This data is identical to that used by Michalski, Mozetic, Hong, and Lavrac (1986) in a study of the incremental learning algorithm AQ15. EACH's results are compared with AQ15's results in the discussion below. AQ15 is a concept learning program which uses a logic-based language to represent the rules it learns. Thus the representations learned by these two programs are quite different. (See Buchanan and Mitchell (1978) for a discussion of rule-learning algorithms, and Bundy, Silver, and Plummer (1985) for a comparative study.) This data set has also been used in a more recent study by Weiss and Kapouleas (1989) that compared a large number of different learning and classification algorithms.

To allow for proper comparisons, the experimental design used was the same as that used by Michalski et al. For each trial, the examples were divided into a training set and a test set. 70% of the examples were randomly chosen for each trial to be in the training set. Four different trials were run, and the final results are an average of those trials. Weiss and Kapouleas (1989) also use this methodology, so the methodology here is consistent with both studies.

In order to make a comparison to human performance, Michalski et al. tested five human experts on the same examples. The human prognoses were correct in 64% of the cases. Michalski et al. report that random guessing would produce 50% correct. However, it would be unfair not to note that approximately 70% of the patients fell into the non-recurrence category. Hence a strategy or predicting the more likely category for every example would give a 70% success rate, although it would be incorrect for *all* of the cases in which the other category was the right response. One must assume that Michalski used a strategy of tossing a fair coin to choose a category in order to produce the 50% figure.

4.1.1. Success rates and comparisons

The best performance of EACH occurred when feature adjustment was in effect ($\Delta_f = 0.2$ for this and all subsequent experiments), where the success rate was 77.6%. Table 1 gives a summary of EACH's performance on this data set with different values of Δ_f. AQ15 had success rates of 66%, 66%, and 68%, using three different configurations of that program. If we take 68% as the performance rate of AQ15, a *t* test on EACH's success rate finds that EACH is significantly better, $p < .01$, than AQ15. The improvement over human experts is even more marked. Without using the feature adjustment rate; i.e., setting $\Delta_f = 0$, EACH's success rate on the test sets was 71.5%, still slightly better than AQ15. Table 1 also includes success rates for the decision tree program CART (Breiman et al., 1984), the PVM rule (Weiss & Kapouleas, 1989), and a back-propagation neural net model. These three results were reported in Weiss and Kapouleas (1989).

4.1.2. Memory requirements

The average size of memory after processing the entire data set of 278 patients was just 22 exemplars. When weight adjustment was not used, the final size of memory was 19 exemplars, although performance was not quite as good. As expected, Greedy EACH required far more memory. These results are summarized in Table 2.

Table 1. Success rates for breast cancer data.

Algorithm	Success Rate (%)
EACH, $\Delta_f = 0.2$	77.6
EACH, $\Delta_f = 0$	71.5
Greedy EACH	72.9
AQ15	68
CART	77.1
PVM rule	77.1
Neural net	71.5
Doctors	64

Table 2. Memory requirements for breast cancer data.

Algorithm	Size of Memory
EACH, $\Delta_f = 0.2$	22
EACH, $\Delta_f = 0$	19
Greedy EACH	68

4.1.3. Varying the feature adjustment rate

Some interesting properties of the algorithm were observed in other tests using the same data. In partciular, the effect of the feature adjustment rate, Δ_f, on the success rate was very good as long as Δ_f was kept small. With $\Delta_f = 0.05$, for example, the success rate was 75%, still a very good result. If Δ_f grew too large, though, the weight adjustments evidently got out of hand, cancelling the effects of earlier weight adjustments and biasing the overall model in the wrong direction. With $\Delta_f = 0.3$, the over-adjustment effect was small, but beginning to be noticeable—the system performed at a 72% success rate, better than with $\Delta_f = 0$, but not as well as with $\Delta_f = 0.2$. With higher values of Δ_f, success rates were lower.

4.2. Iris classification

The next task given to EACH was that of classifying a set of 150 iris flowers, using a data set from Fisher (1936). Each of the examples consists of four integer-valued variables—making it the smallest vector used in the tests of EACH—plus a known assignment of the example to a particular species of iris. The data covered three different species: *I. virginica*, *I. setosa*, and *I. versicolor*. The four variables measured were sepal length, sepal width, petal length, and petal width.

The methodology used here was the *leaving-one-out* cross-validation technique. Cross-validation involves removing mutually exclusive test sets of examples from the data. For each test set, the remaining examples serve as a training set, and classification accuracy is measured as the accuracy on all the test sets. The leaving-one-out method involves removing exactly one example from the data and training on the remaining examples. The technique is repeated for every example in the data set, and the accuracy is measured across all examples. For the iris data set, this involved 150 runs through the data. On each run, 149 examples were used as training, and one example was tested. This methodology was the same one used for the identical data set by Weiss and Kapouleas (1989) in their study comparing several different learning techniques.

4.2.1. Success rates

With no feature adjustment, EACH made 139 correct classifications, for a success rate of 92.6%. With feature adjustment set to 0.2, EACH somewhat better, getting 143 classifications correct, for a success rate of 95.3%. These success rates are summarized in Table 3. As in the previous section, the PVM rule and a neural net algorithm (back propagation) from the Weiss and Kapouleas study (1989) have been included for comparisons. The success rate for CART comes from a study by Crawford (1989).

The smaller memory size used by EACH with $\Delta_f = 0$ was due in part to the use of a smaller seed set (five seeds instead of ten) for that test. Smaller seed sets produced less accurate results in the other tests on this data.

Table 3. Success rates for iris flowers.

Algorithm	Success Rate (%)	Average Memory Size
EACH, $\Delta_f = 0.2$	95.3	18.3
EACH, $\Delta_f = 0$	92.6	6.3
Greedy EACH	94.6	14.4
CART	93	—
PVM rule	96.0	—
Neural net	96.7	—

4.3. Echocardiogram tests

The third set of data used to test the EACH program was a set of records from people who had recently suffered acute myocardial infarctions (heart attacks). This data set contained the smallest number of examples of all sets reported here, but it provided an opportunity to compare EACH to another modeling technique, since medical researchers have used a statistical regression algorithm on the same data. In addition, this data (like the breast cancer data) is real, noisy, and incomplete. It is incomplete in the sense that more variables would need to be measured for each patient in order to make perfect predictions. Noise exists in several variables for which measurement accuracy is not very precise. For example, the "wall motion score" described below is a score determined by a specialist looking at echocardiograms and grading them subjectively.

4.3.1. Description of the domain

First, a brief description of the data itself is necessary, since (unlike the breast cancer data and the iris data) it has not been described elsewhere in the literature. Each example is a record for a patient who has had a heart attack. The data set includes several measures taken from echocardiograms, which are ultrasound measurements of the heart itself. The goal of physicians using these measurements is to predict a patient's chances of survival. In particular, experimental work is being performed currently to detemine if the echocardiogram (in conjunction with other measures) can be used to predict whether or not a patient will survive longer than a certain time period; e.g., one year. The data used in these trials were provided by a medical researcher who is using a statistical regression model (Cox regression) to predict whether patients will live more than one year after a heart attack (Kinney, 1988). In addition, an earlier study (Kan et al., 1986) used echocardiograms to predict the same variable, with similar results. (Kan's study used a different data set containing 345 patients.) Six input variables were used in the experiments below. A complete description of the variables used by Kinney (1988) and by EACH can be found in Salzberg (1989).

Because the prediction was binary, there are two results to report: positive predictive accuracy and negative predictive accuracy. These variables are defined as follows: a positive prediction is a prediction that a patient will live for more than one year, and a negative prediction is a prediction that the patient will die. Positive predictive accuracy is the ratio of the successful positive predictions to all positive predictions (successful and unsuccessful), and negative predictive accuracy is the ratio of successful negative predictions to all negative predictions. The most interesting results concern negative successes, because it is with these that doctors have the most difficulty. The best statistical models are only correct about 60% of the time in predicting that a patient will die (Kan et al., 1986; Kinney, 1988).

4.3.2. *Results*

The data in the tests below include 119 patients. Of these, just three were sufficient as a seed set, leaving 116 patients for the learning trials. Table 4 shows, in addition to overall success rates, the negative predictive accuracy and positive predictive accuracy for each method. An important methodological comment about the studies of Kinney and of Kan et al. is that they only recorded how well their statistical models fit the training data. (The experiments here used the same data set as Kinney, while the Kan study used a different data set.) Unlike the experiments testing machine learning techniques, neither of these experiments divided the data into a training and a test set. Since the statistical models have not been applied to unseen data, the accuracies for these models are a *post hoc* measure of how well each model fits the data. As discussed in Crawford (1989) and in numerous places in the statistics literature, such measures systematically overestimate the accuracy of a model. In fact, instance based models such as EACH and IB2 (Aha et al., 1990) can usually achieve perfect accuracy on data that has previously been seen. The success rates reported in Table 4 for EACH, on the other hand, were obtained using the "leaving one out" method described in the previous section.

5. Discussion

The results on both the breast cancer data and the iris data support the claim that EACH compares favorably with other machine learning algorithms. For the breast cancer data, EACH matched the performance of the PVM rule (Weiss & Kapouleas, 1989) and of the CART algorithm (Breiman et al., 1984), and it surpassed the performance of AQ15 (Michalski et al., 1986). As mentioned above, a statistical comparison shows that EACH's performance on this data is significantly better ($p < .01$) than AQ15's. For the iris data, EACH performed approximately the same as CART, PVM, and neural nets, using the results from Crawford (1989) and Weiss and Kapouleas (1989) as standards for comparison.

Table 4. Echocardiogram success rates.

	Negative Predictive Accuracy (%)	Positive Predictive Accuracy (%)	Overall Accuracy (%)	Size of Memory
EACH, $\Delta_f = 0.2$	56	79	75	11
EACH, $\Delta_f = 0$	50	78	71	7
Greedy EACH	56	80	78	28
Kinney	60	—	—	—
Kan et al.	61	97	86	—

The accuracy of EACH on the echocardiogram data compares favorably with statistical models on negative predictive accuracy (the measure of most interest to doctors using these models), but does not perform as well as the model of Kan et al. on positive accuracy. However, as noted above, the Kan study reported only the fit of the model to the training data, a number that is virtually always an overestimate of the accuracy on unseen data.

Feature weights. A second claim supported by the results is that feature weights are an important component of EACH's success. The results on all three data sets show that feature adjustment does, in general, improve EACH's performance. Table 5 summarizes the results with and without feature weight adjustment on the three test domains. Although the effect of feature weights is not large, it appears to be consistent across all the experiments. Experiments on a simulated domain (reported in Salzberg (1989)) indicated that more rapid weight adjustment is useful during the first few hundred examples, but that smaller values of Δ_f perform better when the training set expands to several thousand examples. Additional experiments need to be run to determine a weight adjustment policy that will apply across all domains.

EACH VS. GREEDY EACH. The third claim that these experiments investigated was that EACH would be superior to Greedy EACH on either classification accuracy or memory requirements, or both. The hypothesis was that the inability of Greedy EACH to create *nested* rectangles would reduce its classification accuracy. Furthermore, the nature of the Greedy algorithm leads it to store more examples. The results on this claim are somewhat mixed, but generally support the superiority of EACH to Greedy EACH. Table 6 shows comparisons on both accuracy and memory requirements for the two algorithms.

The table shows that EACH performed slightly better on two of the domains and slightly worse on the third. There does not seem to be a significant performance difference here. In terms of memory requirements, Greedy EACH needed considerably more memory on two of the three domains, although it needed slightly less on the iris flowers problem. Since both algorithms only store new exemplars when they misclassify an example, the difference in memory requirements will be most clear on problems for which classification accuracy is not very high. As expected, this difference was clear for the breast cancer and echocardiogram domains, where overall accuracy was in the 70–80% range.

Table 5. Effect of feature weights on accuracy.

	Breast Cancer	Iris Flowers	Echocardiogram
EACH with weights	77.6	95.3	75
EACH without weights	71.5	92.6	71

Table 6. EACH vs. Greedy EACH.

	Breast Cancer		Iris Flowers		Echocardiogram	
	Accuracy	Memory	Accuracy	Memory	Accuracy	Memory
EACH	77.2	22	95.3	18	75	11
Greedy EACH	72.9	68	94.6	14	78	28

As explained above, Greedy Each is unable to create nested hyperrectangles. Each should prove superior for domains in which nested exceptions are the most accurate representation of the concept class. The results here demonstrate an advantage in storage efficiency for Each, but no clear advantage in accuracy. The Greedy algorithm even performed slightly better (though not significantly) on the echocardiogram problem, which had the fewest examples. One conclusion to draw from this result is that as the size of the training set increases, the normal, conservative memory policy will outperform the greedy policy. Additional experiments are required to further test this hypothesis.

Memory size and structure. The fourth and final claim was that by storing only a small number of hyperrectangles, Each achieves substantial compression of the training set. To evaluate this claim, we can calculate the number of bits required to store the training set and the number of bits Each uses to store its exemplars (plus the number of bits needed to store any training examples that are misclassified by Each).

We begin by calculating the number of bits required to store each training example. For the breast cancer data, each example is described by nine attributes, of which three are binary (requiring only one bit each) and the others are real-valued. For all real-valued quantities, we will assume that 16 bits are required, although this is surely an overestimate. Finally, of course, one bit is required to indicate whether the cancer recurred. Hence, to store a training example, $3 + 6(16) + 1 = 100$ bits are needed. There are 191 training examples, so the total number of bits required to store the training set is 19,100.

Now let us compute the number of bits that Each uses to store its representation of the training set. For the non-binary attributes in an exemplar, Each stores an upper and lower bound. There is also a weight w_H stored with each exemplar, so the storage required is $3 + 12(16) + 1 + 16 = 212$ bits. Since Each used 22 exemplars for the breast cancer data, its storage requirement for these exemplars was 4664 bits. In addition, a weight w_i was stored on each feature, requiring another 144 bits for the nine features. Finally, even after training, Each misclassifies 24 of the training examples, so we must add another 2400 bits to account for the storage that would be needed in order to perform perfectly on the training data. Adding these numbers up, we get a total of 7208 bits for Each.

Comparing these two quantities, we see that the data compression achieved by Each was better than 2.5 to 1. The compression for the other data sets, although not as impressive, was still substantial. These calculations give some indication of how Each compares to models that store every single example (Reed, 1972). (Kibler and Aha (1987, 1989) have used such models as benchmarks against which to compare their own exemplar-based models.)

One useful feature of exemplar based learning is that the user can examine the memory after learning, and glean some useful informtion from it. A user can examine the rectangles visually (two dimensions at a time) to search for additional properties of the data. For example, for the iris classification task, the program needed very few exemplars to achieve its performance. On one trial (Salzberg, 1989), it needed only five exemplars, and still achieved over 90% accuracy. Of those five exemplars, three were used for the category *I. virginica*, and only one for each of the other two categories, indicating that *virginica* was the most difficult category to define. Figure 5 shows one of the two-dimensional views of the rectangles. The figure also illustrates the fact that one of the categories (*I. setosa*) was linearly separable from the from the other two.

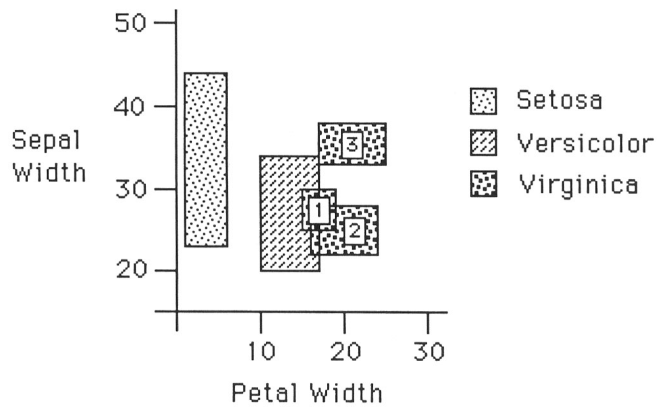

Figure 5. Sepal width vs. petal width for iris flowers.

Examining the model created by EACH for the echocardiogram data reveals other interesting characteristics. Using a trial that required 19 exemplars, five of the exemplars were found to make a positive prediction, which was the correct prediction for approximately three-fourths of the patients. One would expect, then, that these five hyperrectangles would be quite large, as in fact they are—the largest one covers 25% of the entire feature space. In essence, these rectangles define the positive prediction as the system's default.

Of the 14 negative exemplars, seven are simple points; i.e., they were never made into hyperrectangles. Each of the other seven points was used successfully in a prediction, and thus was expanded into a hyperrectangle. The generalization that the system learned could be characterized as a disjunct of these fourteen exemplars, and the exemplars could be shown to doctors as descriptions of patients who are unlikely to survive longer than one year. For example, Figure 6 shows an exemplar created by EACH that predicts that the patient will not survive beyond one year. This exemplar indicates that if the patient's age is between 62 and 85 and pericardial effusion exists, and the other variables fall into the ranges shown, then the prediction is that the patient will die within one year. The weight indicates that this exemplar has made correct predictions in 55.6% of the cases in which it has been used.

```
f1:  Age at time of heart attack: (62 85)
f2:  Pericardial effusion: T
f3:  Fractional shortening (0.07 0.26)
f4:  E-point septal separation (8.5 20)
f5:  Left ventricular end-diastolic dimension (4.65 5.47)
f6:  Wall motion index (1.38 2.25)
Prediction: NIL
Weight: .556
```

Figure 6. Exemplar created from echocardiogram data.

6. Conclusion

The experiments presented above demonstrate that an exemplar-based learning model that constructs hyperrectangles can learn effectively in a diverse set of domains. The EACH system presented here displayed robustness in the face of noise and incomplete data. Comparisons with experts' performance in the breast cancer domain were quite favorable, with the program performing significantly better than the experts.

One of the strengths of the exemplar-based learning model is the simplicity of both the algorithm and the representation it creates. In its barest form, the exemplar model says to store every example as a single point and to predict new points based on simple Euclidean distance to old points. Nested Generalized Exemplar theory retains some elements of this basic model, but it makes some significant modifications. The most important feature of the NGE model is the construction of axis-parallel rectangles—hyperrectangles—and the nesting of exceptions within these rectangles.

The generalization process is also an important component of Nested Generalized Exemplar theory. The generalization rule that EACH uses is to increase the size of a hyperrectangle whenever it makes a correct prediction for a new point lying outside its boundaries. The use of generalization in what is essentially an exemplar-based method points out the strong similarity between exemplar-based learning and DNF-based methods such as those of Quinlan (1986) and Michalski et al. (1986). One of the main distinctions between the methods is that exemplar-based methods can correctly classify an example that is not matched by any exemplar. EACH embodies this ability while still making generalizations whenever possible.

Another important feature of the NGE learning model is the fact that the hyperrectangles can be easily interpreted, when presented in a form such as Figure 5 or Figure 6, by domain experts. This perspicuity is essential for any learning system that might be used by humans as a decision making tool.

Another important issue for NGE is the use of weights in the distance metric. The results above indicate that performance is improved by adjusting the weights on features. The weights on the rectangles themselves are included in the model in order to increase the tolerance for noise. Although the experiments here did not systematically test the usefulness of these weights, Aha and Kibler (1989) have recently created a noise-tolerant version of their instance-based algorithm which uses a factor very similar to EACH's weight factor. Their algorithm keeps track, for each exemplar, of the percentage of the time an exemplar is used to make a correct prediction. EACH tracks the same statistic. If the percentages fall below a certain threshold, Aha and Kibler assume the point represents noise and erase it from memory. Their experiments have yielded positive results with respect to their weight factor. Unlike their program, EACH never erases an exemplar, but the weight factor makes poor exemplars less and less likely to be used by the program.

The theoretical results of Helmbold, Sloan, and Warmuth (1989) nicely complement the experimental model presented here. Although their algorithm includes many restrictions that prevent direct comparisons with NGE, they are working on more general proofs that may remove some of these restrictions. For example, they are working on a proof of an algorithm that allows many distinct sets of nested hyperrectangles, instead of just one (Warmuth, 1989).

As long as we lack proofs of the correctness or optimality of any machine learning algorithm, we will, and should, continue to explore many alternatives. The EACH algorithm represents an alternative that is simple to implement and undemanding of computer memory, yet produces accurate classification models. As with other learning programs, the best support for this one lies in its successful application to real data sets. The results described here provide an encouraging basis for further work and extensions to the NGE algorithm.

Acknowledgments

Thanks to David Aha, Tom Dietterich, Barbara Grosz, William Woods, and two anonymous reviewers for many helpful comments on several careful readings of the manuscript. Thanks to Eve Kinney for providing the echocardiogram data.

Notes

1. Setting Δ_f to 0.0 is equivalent to not using feature weights.

Appendix

The pseudocode shown below presents an overview of the EACH algorithm. Some details described in the text of the paper have been omitted for the sake of brevity. Procedures that simply search data structures have also been omitted.

```
begin
   Number_of_seeds := 10;          /* Size of seed set */
   for i from 1 to number_of_seeds do
      Store_in_memory(read_one_example);
   /* Begin main loop */
   print(''Do you want to process another example?'');
   read(answer)
   if (answer = ''yes'') then
      begin
         e_i := (read_one_example);
         process_next_example(e_i);
      end;
end.

/* the procedure process_next_example does all of the processing
   for a single example */

procedure process_next_example(e_i);
begin
   /* find the two closest matches to the new example */
   M1 := find_closest_exemplar(e_i,*global_memory*);
   M2 := find_second_closest_exemplar(e_i,*global_memory*);
   /* predictions are stored with exemplars */
   P1 := get_prediction(M1);
   P2 := get_prediction(M2);

   /* the new example has its result stored with it */
   result := get_result(e_i);

   if (P1 = result) then
      begin
         adjust_weight_for_success(M1);
         generalize_exemplar(M1,e_i);
      end
   else
      begin
         adjust_weight_for_failure(M1);
         if (P2 = result) then
            begin
               adjust_weight_for_success(M2);
               generalize_exemplar(M2,e_i);
            end;
         else
            begin
               adjust_weight_for_failure(M2);
               /* store the example as a new exemplar */
               store_in_memory(e_i);
               adjust_feature_weights(e_1,M1);
            end;
      end;
end.           /* process_next_example */

/* The procedure generalize_examplar extends a hyperrectangle H just
   far enough to include a new example e, where e is a point. This
   procedure can also handle the case where H is a point.  I have
   omitted details of sub-procedures. */

procedure generalize_exemplar(H,e);
begin
   for i from 1 to *number_of_features* do
      being
      /* feature(i,X) returns the value of feature i on for
         X, where X is either a point or a rectangle.
         The value of feature i may be either a number or an
         interval. lower_end and upper_end return pointers
         to the lower and upper values of an interval.  */
      if feature(i,e) < lower_end(feature(i,H))
      then lower_end(feature(i,H)) := feature(i,e)
      else if feature(i,e) > upper_end(feature(i,H))
          then upper_end(feature(i,H) := feature(i,e));
      end;
end.           /* generalize_exemplar */
```

References

Aha, D. (1989). Incremental, instance-based learning of independent and graded concept descriptions. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 387–391). Ithaca, NY: Morgan Kaufman.

Aha, D., & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 794–799). Detroit, Michigan: Morgan Kaufmann.

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

Ashley, D., & Rissland, E. (1987). But, see, accord: Generating "Blue Book" citations in HYPO. A case-based system for trade secreta law. *Proceedings of the First International Conference on Artificial Intelligence and Law* (pp. 67–74). Boston, MA: ACM Press.

Bareiss, R. (1988). *Protos: A unified approach to concept representation, classification, and learning*. Ph.D. Thesis, University of Texas at Austin. (Technical Report CS-88-10). Nashville, TN: Vanderbilt University, Department of Computer Science.

Bareiss, R., Porter, B., & Murray, K. (1989). *Gaining autonomy during knowledge acquisition*. (Technical Report AI89-96). Austin, TX: University of Texas, Artificial Intelligence Laboratory.

Barr, R., & Caplan, L. (1987). Category representations and their implications for category structure. *Memory and Cognition*, *15*, 397–418.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, *36*, 929–965.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont: Wadsworth.

Buchanan, B., & Mitchell, T. (1978). Model-directed learning of production rules. In D. Waterman, & F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

Bundy, A., Silver, B., & Plummer, D. (1985). An analytical comparison of some rule-learning programs. *Artificial Intelligence*, *27*, 137–181.

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

Crawford, S. (1989). Extensions to the CART algorithm. *The International Journal of Man-Machine Studies*, *31*, 197–217.

Dietterich, T., & Michalski, R. (1983). A comparative review of selected methods of learning from examples. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning*. San Mateo, CA: Morgan Kaufmann.

Everitt, B. (1980). *Cluster analysis*. Hampshire, England: Gower Publishing Co. Ltd.

Fisher, D. (1989). Noise-tolerant conceptual clustering. *Proceedings of IJCAI-89* (pp. 825–830). Detroit, MI: Morgan Kaufmann Publishers.

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

Helmbold, D., Sloan, R., & Warmuth, M. (1989). Learning nested differences of intersection closed concept classes. *Proceedings of the 1989 Workshop on Computational Learning Theory*. San Mateo, CA: Morgan Kaufmann.

Kan, G., Visser, C., Koolen, J., & Dunning, A. (1986). Short and long term predictive value of wall motion score in acute myocardial infarction. *British Heart Journal*, *56*, 422–427.

Kasif, S. (1989). Personal communication.

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

Kinney, E. (1988). Personal communication.

Kolodner, J. (1980). *Retrieval and organizational strategies in conceptual memory: A computer model*. Ph.D. Thesis (Research Report 187). Department of Computer Science, Yale University, New Haven, CT.

Kolodner, J., & Simpson, R. (1984). Problem solving and dynamic memory. *Proceedings of the First Annual Workshop on Theoreteical Issues in Conceptual Information Processing* (pp. 1–10). Atlanta, GA: Georgia Institute of Technology.

Medin, D. (1983). Structural principles in categorizaton. In J. Tighe, & B. Shepp (Eds.), *Perception, cognition, and development*. Hillsdale, NJ: L. Erlbaum Associates.

Medin, D., & Schaffer, M. (1978). Context theory of classification learning. *Psychological Review*, *85*, 207–238.

Michalski, R., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. *Proceedings of AAAI-86* (pp. 1041–1045). Philadelphia, PA: Morgan Kaufmann.

Osherson, D., & Smith, E. (1981). On the adequacy of prototype theory as a theory of concepts. *Cognition*, *9*, 35–58.

Porter, B., Bareiss, R., & Holte, R. (1989). *Knowledge acquisition and heuristic classification in weak-theory domains*. (Technical Report AI89-96). Austin, TX: University of Texas, Artificial Intelligence Laboratory.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Reed, S. (1972). Pattern recognition and categorization. *Cognitive Psychology*, *3*, 382–407.

Rissland, E., & Ashley, K. (1987). A case-based system for trade secrets law. *Proceedings of the First International Conference on Artificial Intelligence and Law* (pp. 60–66). Boston, MA: ACM Press.

Salzberg, S. (1985). Heuristics for inductive learning. *Proceedings of IJCAI-85* (pp. 603–610). Los Angeles, CA: Morgan Kaufmann.

Salzberg, S. (1986). Pinpointing good hypotheses with heuristics. In W. Gale (Ed.), *Artificial Intelligence and Statistics*, Reading, MA: Addison-Wesley Publishing Co.

Salzberg, S. (1989). *Learning with nested generalized exemplars*. Ph.D. Thesis (Technical Report TR-14-89). Cambridge, MA: Harvard University, Department of Computer Science.

Smith, E., & Osherson, D. (1984). Conceptual combination with prototype concepts. *Cognitive Science*, *8*, 337–361.

Thornton, C. (1987). Hypercuboid formation behaviour of two learning algorithms. *Proceedings of IJCAI-87* (pp. 301–303). Milan, Italy: Morgan Kaufmann.

Vere, S. (1980). Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, *14*, 138–164.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.

Warmuth, M. (1989). Personal communication.

Weiss, S., & Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proceedings of IJCAI-89* (pp. 781–787). Detroit, MI: Morgan Kaufmann.