Improving the Design of Induction Methods by Analyzing Algorithm Functionality and Data-Based Concept Complexity*

Larry Rendell Harish Ragavan
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
405 N. Mathews Avenue, Urbana, IL 61801 U.S.A.

Abstract

Although empirical machine learning has seen many algorithms, one of its most important goals has been neglected. Important real-world problems often have just a primitive representation, to which the target concept bears only a remote, obscure relationship. This consideration leads to a class of measures that may be applied to data to estimate difficulty for standard algorithms. As the concept becomes harder, current decision tree and decision list methods give increasingly poor accuracy, though back-propagation does better. A new system for feature construction scales up best. The fundamental limitation of standard algorithms is caused by two problems: greedy search and representational inadequacy. Critical analysis and empirical results show that lookahead alleviates the greedy hill-climbing problem at high cost, but even this is insufficient. Combining lookahead with feature construction alleviates the "complex global replication" problem with hard concepts. For principled algorithm development and good progress, researchers need to study hard concepts and system behavior using them.

1 Introduction

Even in non-incremental, attribute-based learning from examples, many algorithms have been created. To improve their design, researchers often rely on established tactics. One is to identify shortcomings of current algorithms, such as the *replication problem* [Pagallo, 1990]. This sometimes leads to a new algorithm, in this case *FRINGE,* which is later found to have limited value, such as applicability to relatively few concepts [Yang et al., 1991]. Another tactic is to alter the representation. Replacing decision trees with decision lists, Clark and Niblett [1989] showed that *CN2* performed better for certain concepts. These and other tactics amount to a strategy of parallel hill-climbing in a space of designs.

•This work was supported in part by NSF grant IRI-92-04473.

Bradshaw [1993] argues that science advances more quickly when we guide design-space search using a careful functional analysis and suitable decomposition of the underlying problem. Although researchers always use functionality to guide design, algorithms can be evaluated and developed properly only if we decompose the whole problem well.

In this paper we address an undeveloped or neglected dimension of algorithm functionality: concept learning from "primitive" attributes, when experts cannot readily specify a favorable abstract representation. We contend that evaluating learning systems using data bases generated from well understood domains is weak because most of the work is already done. Problems that are simplified through expert knowledge of appropriate abstract attributes are easy to solve [Holte, 1993; Rendell & Seshu, 1990]. In contrast, primitive representation generally degrades accuracy to the point that current algorithms become useless [Rendell & Seshu]. Full assessment of learning algorithms and adequate progress on their design require that we address behavior and phenomena using harder problems.

Section 2 presents a view of concept difficulty based on qualities of real-world representation and typical algorithms. In Section 3 we survey measures and adopt a particular one for the assessment of algorithm performance. Section 4 uses the difficulty measure to assess typical data bases and several algorithms found in learning. These results show the incompleteness of studies that indicate comparable accuracies using various algorithms [Mooney et al., 1989; Weiss & Kapouleas, 1989]. A new algorithm *LFC,* [Ragavan & Rendell, 1993; Ragavan et al., 1993, this volume] scales up particularly well with harder concepts. Section 5 analyzes these results to explain algorithm behavior and to promote research directions.

2 Representation, Algorithms, and Difficulty

In an attribute-based representation, each *training example* is described by an n-tuple of attribute values and a class value (positive or negative in the two-class case). A *concept* is an intensional description of the class, for which a learning algorithm constructs a *hypothesis.*

Numerous extant algorithms perform well on familjar concepts and data found in machine learning work Mooney et al., 1989; Weiss *k* Kapouleas, 1989]. For Many of the Irvine data bases, even a very simple algorithm attains accuracies within 5% of the best method [Holte, 1993]. The basic reason for the high accuracies is that the representation for the underlying concept matches the bias of the SBL algorithm. Typical systems use the similarity *(SBL)* bias: neighboring points in instance space are likely in the same class [Rendell, 1986].[1]

Standard learning systems are accurate if the concept is localized in instance space [Rendell *k* Cho, 1990], so they are good at this function of describing single locations. An area of uniform class membership values, especially a peak or valley in the function, is a contiguous, constant-valued *region.* A region is describable using a few disjuncts (to capture irregular shapes), and is easy to find with simple algorithms that use greedy methods (to view one attribute at a time). Although the details depend on the approach [Rendell, 1986], this function could be called *region description* (RD). Concepts learned accurately by an SBL system designed for RD are "easy" for the algorithm.

In contrast, *hard* concepts are "spread out;" their class membership functions have a high degree of variation [Devroye *k* Gyorfi, 1985; Rendell *k* Seshu, 1990]. Function variation is correlated with other phenomena, such as entropy [Watanabe, 1985] and attribute interaction [Devijver *k* Kittler, 1982], which we collectively call *(concept) scattering.* Hard concepts are learned very poorly by standard algorithms, even when many data are available [Rendell *k* Seshu, 1990].

Concept scattering is not just an academic consideration; it relates directly to representations for real-world domains. Easier concepts are associated with good abstract representations, harder concepts with primitive information closer to direct observation. A good representation for checkers involves piece advantage and center control, whereas primitive attributes are the contents of board squares. A high-level representation for symbol recognition includes definitions of lines and circles, while a low-level representation uses pixel gray-levels. No satisfactory abstract representation is yet known for protein structure prediction, but a hard starting point is the primary sequence of amino acids (each position being an attribute).

In such examples, abstract representations simplify the relationship (reduce the complexity or variation) between the concept and its attributes [Drastal et al., 1989; Rendell *k* Seshu, 1990]. For example, the likelihood of winning checkers increases monotonically with piece advantage and center control [Samuel, 1959]. In contrast, the likelihood of winning changes drastically with a small change in the contents of one or two board squares [Rendell, 1985]. Primitive representation causes concept scattering.

Good abstract representation is based on knowledge. Concepts that have had the benefit of expert knowledge to hone good attributes may be learned accurately and quickly [Rendell *k* Seshu, 1990; Samuel, 1959]. But even moderately complex concepts are difficult if the representation is primitive and little knowledge is available [Quinlan, 1983; Rendell, 1985]. The relationship of primitive attributes to the target concept is often obscure, making the problem hard and standard algorithms slow and inaccurate [Devijver *k* Kittler, 1982; Rendell *k* Seshu, 1990]. For some important real-world problems, the best known representation is poor, and existing induction systems perform little better than guessing based on the prior class probability [Seshu et al., 1989; Towell et al., 1990].

In addition to the region description function RD when concepts are easy, we need to do something about poor representation. Since primitive attributes cause a proliferation of concept regions, another function might be *coalescing abstraction* (CA): change of representation [feature construction; Matheus *k* Rendell, 1989] to diminish the number of regions [variation reduction; Rendell *k* Seshu, 1990]. CA allows RD to manage its task [Drastal et al., 1989].

Algorithm limitations for different amounts of abstraction are largely unexplored, though theory has shown that as concept complexity increases, more data are required to maintain accuracy [Devroye *k* Gyorfi, 1985; Ehrenfeucht et al., 1988]. To better understand the relationships between algorithm capability and representation quality, we can analyze hard concepts, characterize concept difficulty, locate associated deficiencies of algorithms, then reorient approaches to system design.

3 Capturing Concept Difficulty

3.1 Measures of Difficulty

If we quantify poor representation, we can determine the capabilities of algorithms in a more principled way. To approximate concept difficulty, various formalisms have been used.

Measures can be based on the form of the hypothesis, as in theoretical research. Although they provide guiding principles, hypothesis-based measures are impractical for assessing and developing algorithms, since the form of the concept is often unknown. The quantities we examine here are based on the data.

One view is that hard concepts have a class-membership function that is either close to monotonic or else singly-peaked, whereas hard concepts (functions) have many peaks and valleys [regions of uniform class membership; Rendell, 1985; Rendell *k* Cho, 1990]. A related view is Quinlan's [1983] description of hard problems as having large numbers of disjuncts (more peaks require more distincts to describe). Multiple peaks or disjuncts generally cause attribute interaction, which can be simplified in terms of statistical correlation [Devijver *k* Kittler, 1982]. A generalized measure of scattered peaks or disjuncts is function variation [Devroye *k* Gyorfi, 1985; Rendell *k* Seshu, 1990]. More variation for boolean concepts leads to greater complexity [much conjunction and disjunction; Ehrenfeucht et al., 1988]. Finally, concept difficulty also may be described using entropy [Saxena, 1991; Watanabe, 1985], which is commonly used to distinguish attribute relevance in splitting

[1]This geometric view is clearest if the attributes are numeric, but it can be extended to other types.

algorithms [Quinlan, 1983] and transformation schemes [Devyver & Kittler, ch. 5].

Here we advocate an entropy measure called the *blurring.* The blurring of a concept is the average information in the concept over all relevant attributes, conditioned on each attribute in turn. This estimates scattering and interaction because each term in the conditional expression corresponds to a one-dimensional projection in instance space. One attribute alone provides little information about the concept class when scattering and interaction are high. For difficult concepts, any one-dimensional projection is a "blur" comprising many highs and lows of concept class-membership (positive and negative examples). Consequently, such a projection shows a large degree of uncertainty about the concept. In general, the greater the difficulty of the concept, the more blurred these projections become. (Instead of averaging attribute contributions, variants of this definition might be used, as discussed later.)

Formally, the entropy of a binary concept y with class values 0 and 1 is $H(y) = -[p(y)\log_2 p(y) + p(\bar{y})\log_2 p(\bar{y})]$ where $p(y)$ and $p(\bar{y})$ are the prior probabilities that $y = 1$ and $y = 0$, respectively. Let $x_1, x_2, \ldots x_n$ be the relevant attributes for a concept y (relevant attributes—the variables on which the class variable depends, are discussed further below). The *blurring* of y is $\Delta = \sum_{i=1}^{n} H(y|x_i)/n$, where $H(y|x_i)$, the entropy of y conditioned on x_i, is $-\sum_v p(x_i = v)[p(y|x_i = v)\log_2 p(y|x_i = v) + p(\bar{y}|x_i = v)\log_2 p(\bar{y}|x_i = v)]$ over all values v of x_i.

Blurring is a property of the concept and its instance representation. A high value of Δ indicates high SBL learning difficulty; the minimum (zero) can occur only if one of the features is the concept itself

3.2 Measurement Issues

A measure to characterize concept difficulty should have certain qualities: It might correspond to a human perspective. The measure should track concept difficulty as experienced by some standard algorithms. To be useful in detailed studies, a measure should apply to a wide variety of domains and capture fine differences in concept difficulty. The measure should be defined for and computable from both the concept (usually for synthetic cases) and its training data (for real-world cases, when the concept is unknown).

According to these desiderata, some of the candidate measures seem particularly appropriate, and others less so. The number of peaks in the class-membership function is useful for controlled experiments with synthetic concepts [Rendell & Cho, 1990], but is coarse, since peaks vary in extent, height, and shape. Moreover, the number of peaks is hard to measure in real data. In contrast, function variation and concept blurring are fine-grained, and may be computed indirectly from the concept definition (calculating all values) or estimated directly from available data when the concept is unknown (ignoring missing instances).

Like some other measures, blurring captures human preference for compact spatial representations and against scattered primitive representations. As detailed in Section 4, A also estimates the difficulty experienced by many learning algorithms, in terms of their accuracy. Blurring is general (applicable to any type of feature), and precise (responsive to small variations in feature quality).

Whether we choose blurring, variation, or any other datarbased quantity, our current definitions seem to be limited in one way or another. Once again we focus on blurring. First, the exact form of the definition is questionable. To measure projection blurring accurately, Δ should use only those attributes relevant for learning the concept. Averaging may be good if the relevant attributes are known, but more commonly they are unknown, and too many irrelevant attributes artificially raises the blurring estimate. Nevertheless, this is not a serious problem as long as most of the variables are relevant (which is true of our early studies). Our current goal is to assess existing algorithms, which is somewhat insensitive to this issue.

Another question is the importance of higher dimensional projections for more complete definitions of blurring. To measure blurring, Δ estimates entropy in one-dimensional projections. This simplification may have limited use, because it does not discriminate higher order interactions and therefore compresses estimates of difficulty at the higher end of the scale. However, our experiments suggest that much can be done with the simple definition, because typical Δ values are relatively low and current algorithms fail before a concept becomes very hard.

A final simplification in the present definition of blurring is its omission of instance space dimensionality (others have included dimensionality for different uses than ours) [Devijver & Kittler, 1982; Saxena, 1991]. We factor dimensionality out of the current experiments, by keeping it nearly constant. We explore the effects of concept scattering on various algorithms, which likely retain their ranking independent of dimensionality.

Some of the other measures, such as the variation, avoid the above problems, although they have different drawbacks. As our understanding of algorithm behavior becomes more refined, blurring or related measures should be improved.

4 Using the Blurring Measure

4.1 Blurring Examples and Interpretation

Like Holte [1993], we found that many databases in the Irvine repository are easy. If we measure the entropy for the best attribute, we often find values close to zero, though many of the Irvine databases have Δ values 0.5-0.6, likely because less relevant attributes mixed with one important one raise the average. Δ values of 0.6 or less often seem to indicate easy SBL learning; for example simple algorithms run on Iris versicolor (Δ = 0.56) give around 100% accuracy.

Some of the Irvine data bases have high Δ values, such as Pima Diabetes with Δ = 0.88, and (seven attribute) majority voting with Δ = 0.93. Other domains having high Δ include protein structure and bankruptcy [Ragavan et al., 1993, this volume; also see Rendell, 1985]. For boolean concepts, the highest blurring is for parity. In

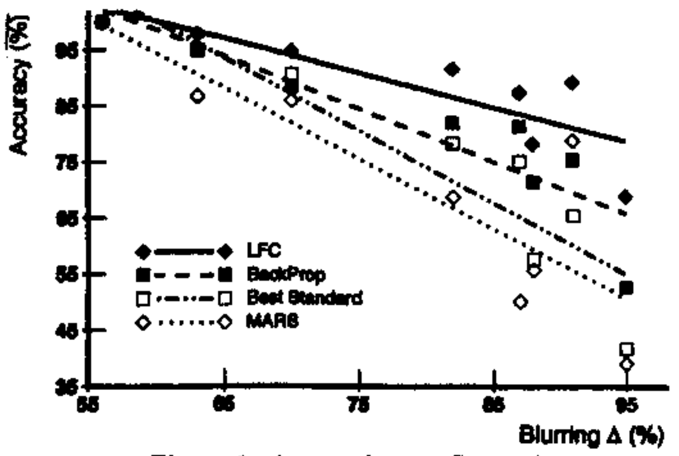

Figure 1: Accuracies vs. Scattering

the most extreme feature interaction, no single attribute can individually provide information about the concept, but collectively the attributes provide full information. While parity and other boolean concepts are artificial, they can mimic hard concepts in real-world domains. In particular, degrees of interaction can be simulated using different orders of parity. Parity attributes provide a necessary and sufficient (basis) set for boolean concepts [Seshu, 1989].

As defined, blurring is non-linear; higher values of Δ compress intuitive differences. For example Δ = 0.88 for Pima diabetes indicates much greater difficulty than Δ = 0.56 for Iris. Beyond some threshold (around 0.9), increasing the blurring slightly requires many more data and a more complex algorithm to learn, as seen below.

Although we use Δ to measure concept scattering and attribute interaction, the data can be blurred for two other reasons. One is fundamental representation inadequacy, as opposed to representation unfavorable to the algorithm. Fundamental inadequacy can cause one point in the instance space to represent different objects and class values; destroying such a distinction increases entropy. The other cause of blurring (entropy) is noise. Attribute and class noise cause false positives and negatives, which increase scattering.

The blurring Δ captures all three factors that exacerbate concept learning, but does not discriminate them. If causes are unknown, the value of Δ becomes an upper limit for estimating concept scattering. For real-world domains, the exact sources of entropy are often uncertain, although other measures may conceivably disambiguate. In our controlled experiments using synthetic data, the only cause of blurring is scattering.

We must consider concept scattering and attribute interaction when developing learning algorithms for primitive representations. If we do not design experiments to assess induction systems on hard concepts, our design efforts will be handicapped. Experiments would not diagnose system deficiencies [Bradshaw, 1993]. Blurring, or variation, or some characterization of real-world difficulty can facilitate and systematize the assessment of algorithms.

4.2 Algorithm Performance

This summary of algorithm accuracy as a function of blurring omits much of the experimental detail found in [Ragavan *k* Rendell, 1993]. We ran several algorithms, including *IDS, C4.5* [Quinlan, 1983], *Fringe, GREEDYS* [Pagallo, 1990], *DCFringe* [Yang et al., 1991], *BackProp* [Rumelhart et al., 1986], *MARS,* [Friedman, 1991], and *LFC* [Ragavan *k* Rendell; Ragavan et al., 1993], on four synthetic and four real-world concepts. Figure 1 shows predictive accuracies obtained using ten-fold crossvalidation; differences are mostly significant at the 0.001 level. The curve labeled "Best Standard" is a composite formed by choosing, row by row, the best accuracy of all greedy algorithms that output logic or tree hypotheses (*ID3, GREEDYS, DCFringe,* etc.). This curve is a mixture of systems, *IDS* or *DCFringe* often giving the best results.

The graph shows that the accuracy of all algorithms degrades with increasing Δ. The nonlinear anomalies are unsurprising; one reason is that training sample sizes differ [Ragavan *k* Rendell, 1993]. Otherwise, differences other than blurring have a relatively minor effect [Rendell *k* Cho, 1990]. Δ explains most of the accuracy differences: hard concepts are learned poorly by standard algorithms [see also Rendell *k* Seshu, 1990].

But the tested algorithms differ markedly. This is contrary to the results of Weiss and Kapouleas [1989] and Mooney et al. [1989], because their studies did not account for concept difficulty. When we consider concept scattering and attribute interaction as represented by the blurring, we see that *BackProp* scales up better. *LFC* [Ragavan *k* Rendell, 1993; Ragavan et al., 1993] scales up best. For hard concepts, we find large accuracy differences such as 66% versus 90% and 42% versus 70%.

Concept scattering and attribute interaction are the essence of difficulty in poorly-understood domains. The ability of algorithms to manage scattered concepts may be their least understood but most important functional difference. In the analysis below, we correspond the behavior of current algorithms with their design elements and with requirements for hard concepts.

5 Algorithm Functionality

5.1 Basic SBL Techniques

Similarity-based methods are sometimes dichotomized into iterative splitting algorithms [divide-and-conquer, e.g., *CART,* Breiman et al., 1984; *PLS1,* Rendell, 1986; *IDS,* Quinlan, 1983] and set covering algorithms [separate-and-conquer, e.g., *AQ,* Michalski, 1983; *CN2,* Clark *k* Niblett, 1989; *GREEDYS,* Pagallo, 1990]. The former are often associated with decision tree representations, while the latter frequently use decision lists. Basic iterative splitters use one attribute at a time to build a decision tree. Although a greedy, hill-climbing strategy is efficient, it produces inaccurate trees when the concept is hard—when class membership is scattered and attributes interact [Rendell *k* Cho, 1990].

In hard concepts a number of attributes together determine the class of an example; a single attribute provides little or no information about the class. Consequently, depending upon statistical coincidences in the data, a decision tree learner can easily make a poor choice of attributes. Simple boolean parity illustrates the effects of attribute interaction: If the concept is $XOR(x_1, X_2)$ the probability of an example from a uniform distribution being positive is 1/2 both before *and* after consideration of the value of x_1 or x_2 alone. Each attribute by itself is uninformative, but greedy splitting still chooses one of them for creating a decision node, making an "uninformed" split. This produces a tree with three decision nodes, as the two conjunction relations X_1X_2 and x_1X_2 need to be captured to discriminate all the examples correctly. Because every split on average halves the number of data, the likelihood of building the correct tree diminishes, unless the training sample is very large. This problem worsens if the parity (or any attribute interaction) is higher-order.

Moreover, a greedy algorithm is likely to select irrelevant attributes: their discriminatory power is as good as, or better than, relevant but interacting attributes (statistical anomalies produce a stronger effect than the zero information in a relationship such as parity). The result of greedy splitting is a verbose tree. With limited training data, verbosity perpetuates poor node selection, now farther down the tree because of reduced statistical support there. One instance of this data fragmenting is Pagallo's [1990] fringe replication problem.

Functionally, basic decision tree algorithms perform well on concepts having little concept scattering or attribute interaction, but otherwise these algorithms are highly inaccurate (as illustrated in Fig. 1). One way to attempt improvement of SBL algorithms is to search their design space.

5.2 Some Extensions and Other Designs

Some algorithms extend greedy splitters. *FRINGE* [Pagallo, 1990], *DCFringe* [Yang et al., 1991] and *CITRE* [Matheus & Rendell, 1989], use the decision tree produced by a greedy splitter to construct new features for improving tree quality. Repeated local patterns in the decision tree are coalesced to give new features. The new features are added to the original attributes for competitive splitting, to build a new tree. The process continues iteratively until no new features are found, giving a tree in which repeated patterns are eliminated. *FRINGE* conjoins two adjacent nodes near the positive leaves of the tree. *DCFringe* constructs both conjunctions and disjunctions based on the fringe structure. *CITRE* conjoins adjacent fringe, root, or intermediate nodes. But such greedy feature construction algorithms improve relatively few concepts [Yang et al., 1991].

FRINGE-like algorithms are limited by the quality of the original decision tree from which they construct features. If the basis for the construction of the original tree is a greedy splitter, accuracies remain low (Fig. 1). With adequate data in a parity problem, *FRINGE* would eventually construct the right features, but this construction is coincidental. Over many runs, is just as likely to construct irrelevant features because greedy splitters cannot discriminate properly. With limited data, even this inefficient technique will break down.

Other approaches than *FRINGE* transform decision trees for incremental induction. *IDL* [Van de Velde, 1990] switches nodes in the tree based on frequency of attribute occurrence. Indurkhya and Weiss [1990] discuss a similar attribute swapping technique. But these approaches are also limited by the quality of the algorithm that produces the original decision tree.

Other design varieties are also possible. Instead of extending greedy SBL, one could use a different representation. A decision list is a set of rules involving conjunctions of attributes [Rivest, 1987]. Each conjunction is a specific rule for attribute interaction. However, algorithms such as *GREEDYS* [Pagallo, 1990] are not designed to find the precise interaction except by assessing the marginal effect of one attribute at a time. As currently designed, these algorithms also suffer greedy limitations. Consequently, they perform poorly for harder problems, as shown in Figure 1.

Although search of design space is necessary, fast advancement involves a good decomposition of functionality [Bradshaw, 1993]. Decision lists and greedy algorithms *are* functional: both are designed for economy. But this is just one of many functions. To guide design space search, we need to focus on the critical dimensions of algorithm functionality.

In terms of Section 3, greedy algorithms are good at the region description function RD as long as the blurring is low and attribute interaction is weak. But for hard concepts, finding the regions of unform class membership is not so simple, because peaks and valleys are blurred in greedy projections. To help find regions, the learning approach needs to *discover the specifics of attribute interaction by countering blurring.*

5.3 Finding Interactions using Lookahead

Backpropagation and statistical techniques designed to handle attribute interaction have had good results across a variety of domains [Rumelhart et al., 1986; Friedman, 1991]. Another way to manage interaction is to look ahead in a decision tree, effectively using all multidimensional projections of the training sample on the attributes. An SBL splitting algorithm could look ahead to observe the effects of current attribute selection further down in the hypothesis construction. The best sequence is found after evaluating all expansions of a decision tree. This avoids problems of attribute interaction by evaluating combinations of features. Norton [1989] found that his exhaustive lookahead algorithm *IDX* gave fairly good improvement.

Exhaustive lookahead, however, is expensive. We found that the lookahead program within Buntine's and Caruana's [1993] *IND* takes hours to run for moderately difficult financial concepts [Ragavan et al., 1993, this volume]. Even with the fastest available architectures, dimensionality growth in the training data can swamp the system, because the search space grows doubly exponentially as the lookahead increases.

Greedy tree building is too limited and exhaustive lookahead is too expensive. So a cure for attribute interaction may lie in dynamic lookahead. For fixed dimensionality instance space, naive lookahead would evaluate the same number of features at every node, independent of concept complexity. Instead, Ragavan's lookahead feature construction *LFC* uses a combination of several techniques to guide lookahead search. The algorithm applies both analytic and heuristic methods to decide dynamically which paths to attempt and how far to pursue the search. Results showed reasonable speeds.

However, even lookahead is not enough to maximize performance. We found that accuracy improves further when feature construction caches the information [Ragavan & Rendell, 1993]. (Figure 1 summarizes some accuracy results.) Feature construction also benefits comprehensibility and abstraction.

In terms of Section 3, SBL with lookahead performs the region description function RD well, even when concepts are scattered and attributes interact. To implement the coalescing abstraction function CA, an algorithm can create compact hypotheses by finding suitable new features.

5.4 Compressing Hypotheses using New Features

Even with lookahead to specify attribute relationships, an independent problem remains. The *replication problem* [Pagallo, 1990] is just one type of verbosity. In hard concepts, local (fringe) replications are relatively few [Yang et al., 1991], yet "complex global replication" seems prevalent. *Global replication* is the disguised occurrence of similar attribute combinations in different parts of the tree. Many hard concepts require repeated decision patterns which are distributed throughout the tree and not readily recognizable, rather than being confined to the fringes and clearly demarcated.

For example, given all the lookahead it needs, a greedy learner for two-parity *(XOR)* builds a tree having three layers and seven nodes. This verbosity has splits on one attribute occurring on two subtrees of the root. With limited training data, or with fixed training sample size but higher-degree attribute interaction, the data are exponentially decimated as the tree is grown.

To alleviate this rapid data fragmenting, separate-and-conquer methods have been developed that increase the available data for further hypothesis growth. Decision list algorithms such as *CN2* [Clark & Niblett, 1989] and *GREEDYS* [Pagallo, 1990] construct a new feature at every node, and retain the examples not covered by the feature for further list construction. As features are formed by conjoining data attribute literals to give specialized terms, this technique increases the available data for making further statistical inferences to construct subsequent features.

But this approach also creates some problems. One is that specializing on each disjunct of the concept at every step can cause the hypothesis to grow rapidly for concepts having numerous disjuncts scattered over the instance space. This was the case with a bankruptcy concept, where *GREEDYS* gave an accuracy of 58%, compared with *LFC's* 90% [Ragavan et al., 1993].

LFC handles global replication differently. Complex concepts typically contain implicit patterns caused by prominent relationships among the original data attributes. The algorithm caches search information by constructing useful new features (e.g., sequences of decisions down a path of a tree is a conjunction of attributes). In the *XOR* example, *LFC* constructs new features such as f_1 = x_1 A x_2 and f_2 = x_1 A$x2$, to build a concise and accurate tree, f_1 and f_2 represent abstractions "closer" to the parity concept than x_1 or x_2- New features describe relationships intermediate between the original attributes and the concept. *LFC* tackles the feature interaction problem directly, constructing features while building the decision tree. The new features compress the hypothesis.

Compared with *CN2, GREEDYS,* and *GS* [Murphy *k* Pazzani, 1991], *LFC* differs primarily because it tackles both the attribute interaction problem (using directed lookahead) and the global replication problem (through feature construction). Norton's [1989] *IDX* also reacts to the required degree of conjunction, though it uses exhaustive lookahead and does not cache features. The "lookback" techniques of Indurkhya and Weiss [1990] and Van de Velde [1990] do not construct new features and so are limited to the expressive power of the original attributes at each node.

LFC's combination of dynamic lookahead and feature construction consistently and considerably outperformed all other algorithms we ran, especially on hard concepts [Ragavan & Rendell, 1993] because the design addressed both attribute interaction and representation verbosity. These design requirements resulted from an analysis of the nature of hard concepts. When concepts are scattered and their attributes interact, the region description function RD leads to lookahead (or its equivalent). When concepts are scattered and their attributes interact, the coalescing abstraction function CA leads to construction of new features (found during lookahead) that compress the representation by reducing global replication through economic abstraction.

6 Conclusion

These results show the importance of careful analysis. Currently many learning systems perform very poorly, displaying accuracies (Fig. 1) tens of percentage points worse than *LFC* [Ragavan and Rendell, 1993; Ragavan et al., 1993, this volume]. This new feature construction system was designed for hard real-world problems characterized by attribute interaction and concept scattering. To continue to improve algorithm design, we need to pay close attention to the nature of real-world problems. Concept difficulty measures such as variation and entropy can help pinpoint algorithm behavior and aid system development.

References

[Bradshaw, 1993] G.L. Bradshaw. Heuristics and strategies of invention: Lessons from the invention of the airplane. UIUC Report, submitted to Cognitive Science.

[Breiman et al., 1984] L. Breiman, J.H. Friedman, RA. Olshen, and C.J. Stone. *Classification and Regression Trees;* Wadsworth, Belmont, CA, 1984.

[Buntine St Caruana, 1993] W.L. Buntine and R. Caruana. Introduction to IND Version 2.1 and Recursive Partitioning. NASA Ames Research Center Report FIA-93-03, 1993.

(Clark St Niblett, 1989] P. Clark and T. Niblett. The CN2 induction algorithm. Machine Learning 3, 1989, 261-284.

[Devijver & Kittler, 1982] PA. Devijver and J. Kittler. Pattern Recognition; A Statistical Approach. Prentice Hall, Englewood Cliffs, NJ, 1982.

[Deyroye St Gyorfi, 1985] L. Devroye and L. Gyorfi. Non-pmrmmetric Density Estimation: The L_1 View. Wiley, New York, NY, 1985.

[Drastal et al., 1989] G. Drastal, G. Csako, and S. Raats. Induction in an abstraction space: A form of constructive induction. In Proc. Eleventh Intl. Joint Conf. on AI, 1989, 708-712.

(Ehrenfeucht et al., 1988] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. In Proc. Computational Learning Theory, 1988, 139-154.

[Friedman, 1991] J.H. Friedman. Multi-variate adaptive regression splines. Annals of Statistics, 19, 1991, 1-141.

[Holte, 1993] R.C. Holte. Very simple classification rules perform well on most datasets. Machine Learning (in press).

[Indurkhya St Weiss, 1990] N. Indurkhya and S.M. Weiss. Iterative Rule Induction Procedures. Laboratory for Computer Science Research Report LCSR-TR-145, Rutgers University, New Brunswick, NJ, 1990.

[Matheus St Rendell, 1989] C.J. Matheus and LA. Rendell. Constructive induction on decision trees. In Proc. Eleventh Intl. Joint Conf. on AI, 1989, 645-650.

[Michalski, 1983] R.S. Michalski. A theory and methodology of inductive learning. In R.S. Michalski et al. (Ed.), Machine Learning: An Artificial Intelligence Approach. Tioga, 1983.

[Mooney et al., 1989] R. Mooney, J. Shavlik, G. Towell, and A. Gove. An experimental comparison of symbolic and connectionist learning algorithms. In Proc. Eleventh Intl. Joint Conf. on AI, 1989, 775-780.

[Murphy St Pazzzani, 1991] P. Murphy and M. Passani. Constructive induction of M-of-N concepts. In Proc. Eighth Intl. Machine Learning Workshop, 1991, 183-187.

(Norton, 1989] S. Norton. Generating better decision trees. In Proc. Eleventh Intl. Joint Conf. on AI, 1989, 800-805.

[Pagallo, 1990] G. Pagallo. Learning DNF by decision trees. Ph.D. Dissertation, University of California at Santa Crus, 1990.

[Quinlan, 1983] J.R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R.S. Michalski et al. (Ed.), Machine Learning: An Artificial Intelligence Approach. Tioga, 1983.

(Ragavan & Rendell, 1993] H. Ragavan and L.A. Rendell. Lookahead feature construction for learning hard concepts. In Proc. Tenth Intl. Machine Learning Conf., 1993.

(Ragavan et al., 1993] H. Ragavan, L.A. Rendell, M. Shaw, and A. Tessmer. Complex concept acquisition through directed search and feature caching. In Proc. Thirteenth Intl. Joint Conf. on AI, 1993 (this volume).

[Rendell, 1985] LA. Rendell. Substantial constructive induction using layered information compression: Tractable feature formation in search. In Proc. Ninth Intl. Joint Conf. on AI, 1985, 650-658.

(Rendell, 1986] L.A. Rendell. A general framework for induction and a study of selective induction. Machine Learning 1, 1986, 177-226.

(Rendell St Cho, 1990] L.A. Rendell and H.H. Cho. Empirical concept learning as a function of concept character, Machine Learning 5, 1990, 267-298.

[Rendell St Seshu, 1990] L.A. Rendell and S.M. Seshu. Learning hard concepts through constructive induction: Framework and Rationale. Computational Intelligence $, 1990, 247-270.

[Rivest, 1987] R. Rivest. Learning decision lists. Machine Learning 2, 1987, 229-246.

[Rumelhart et al., 1986] D.E. Rumelhart, G.E. Hinton, and R. Williams. Learning internal representations by error propagation. Parallel Distributed Processing: Explorations in the Micro structures of Cognition 1, MIT Press, 1986, 318-362.

[Samuel, 1959] A.L. Samuel. Some studies in machine learning using the game of checkers. In IBM Journal of Research and Development. 3, 1959. Reprinted in E.A. Feigenbaum (Ed.), Computers and Thought. McGraw-Hill, 1963.

[Saxena, 1991] S. Saxena. On the effect of instance representation on generalisation. In Proc. Eighth Intl. Machine Learning Workshop, 1991, 198-202.

[Seshu, 1989] R.M. Seshu. Solving the parity problem. In Proc. of the European Working Session on Learning, Morgan Kaufman, 1989, 283-271.

[Seshu et al., 1989] R.M. Seshu, L.A. Rendell, and D.K. Tcheng. Managing constructive induction using subcomponent assessment and multiple-objective optimization. In Proc. Fifth Intl. Conf. on AI Applications, 1989, 191-197.

[Towell et al., 1990] G. Towell, J. Shavlik, and M. Noordwier. Refinement of approximate domain theories by knowledge-based neural networks. In Proc. Eighth Ntl. Conf. on AI, 1990, 861-866.

[Van de Velde, 1990] W. Van de Velde. Incremental induction of topologically minimal trees. In Proc. Seventh Intl. Machine Learning Conf., 1990, 66-74.

[Watanabe, 1985] S. Watanabe. Pattern Recognition: Human and Mechanical. Wiley, New York, NY, 1985.

[Weiss St Kapouleas, 1989] S.M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In Proc. Eleventh Intl. Joint Conf. on AI, 1989, 781-787:

[Yang et al., 1991] D-S. Yang., L.A. Rendell, and G. Blix. A scheme for feature construction and a comparison of empirical methods. In Proc. Twelfth Intl. Joint Conf. on AI, 1991, 899-704.