A Scheme for Feature Construction and a Comparison of Empirical Methods*

Der-Shung Yang   Larry Rendell   Gunnar Blix
Beckman Institute and Computer Science Department
University of Illinois at Urban a-Champaign
405 N. Mathews Ave., Urbana, IL 61801.
yang@cs.uiuc.edu rendell@cs.uiuc.edu blix@cs.uiuc.edu

Abstract

A class of concept learning algorithms *CL* augments standard similarity-based techniques by performing feature construction based on the SBL output. Pagallo and Hausslcr's FRINGE, Pagallo's extension Symmetric FRINGE (Sym-Fringe) and a refinement we call DCFringe are all instances of this class using decision trees as their underlying representation. These methods use patterns at the fringe of the tree to guide their construction, but DCFringe uses limited construction of conjunction and disjunction. Experiments with small DNF and CNF concepts show that DCFringe outperforms both the purely conjunctive FRINGE and the less restrictive SymFringe, in terms of accuracy, conciseness, and efficiency. Further, the gain of these methods is linked to the size of the training set. We discuss the apparent limitation of current methods to concepts exhibiting a low degree of feature interaction, and suggest ways to alleviate it. This leads to a feature construction approach based on a wider variety of patterns restricted by statistical measures and optional knowledge.

1 Introduction

Supervised *concept learning* is the problem of finding a description of an unknown class of objects for which we are given a set of training examples. The most common approaches to this problem can be collectively referred to as *similarity-based learning* (SBL), including agglomerative learning systems and splitting algorithms [Breiman *et al*., 1984; Quinlan, 1983].

The effectiveness of most SBL algorithms is influenced by *concept difficulty,* variously measured as feature interaction [Devijver and Kittler, 1982; Rendell and Seshu, 1990], concept dispersion [Rendell and Cho, 1990], cross entropy [Ragavan and Rendell, 1990], and term and literal complexity [Ehrenfeucht *et al*., 1988]. These studies show that SBL concept difficulty affects the accuracy and conciseness of machine learning systems.

*The authors gratefully acknowledge the support of NSF grant IRI 8822031. Blix is supported by the Royal Norwegian Research Council for Science and Humanities and University of Illinois CS Fellowships.

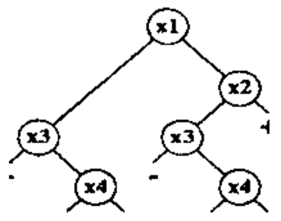

Figure 1: Decision tree representation for $x_1x_2 \vee x_3x_4$.

Concepts used to test similarity-based learning systems are often not difficult, because their corresponding feature set is selected with the aid of human experience. However, for many interesting and important concepts the appropriate level of abstraction is not directly available. A shift of inductive bias through feature construction may help construct an instance space at a proper level of abstraction for conventional SBL system to work [Utgoff, 1986; Matheus, 1989]. This paper discusses a general approach to feature construction and compares specific approaches based on the FRINGE algorithm.

FRINGE [Pagallo and Haussler, 1990; Pagallo, 1989] is a feature construction algorithm for decision-tree learning designed to combat the replication problem. The *replication problem* is the duplication of a sequence of tests in different branches of a decision tree (Fig. 1), leading to an inconcise representation that also tends to have low predictive accuracy. Although replications are inherent to the decision tree representation rather than the concept itself, the replication problem is a manifestation of underlying concept characteristics that is accentuated for difficult concepts [Yang *et al*., 1991].

Section 2 presents a generalized algorithm schema based on FRINGE that utilizes the output of similarity-based learning for feature construction. We present some specific approaches captured by this algorithm schema. Section 3 summarizes and interprets the results of experiments with three feature construction algorithms. Finally, section 4 discusses inherent limitations and research directions.

```
Concept learning algorithm CL
Let Prm be the set of primitive user-given attributes.
Input the Data expressed using Prm, each
        with its class-member ship value.
Initialize active feature set Act <— Prm.
Repeat until some stopping criterion is met:
    1. convert Data from ground Prm to active form Act;
    2. perform SBL(Act, Data, Tree)
        (build a decision tree using splitting algorithm);
    3. if required, call FC to construct new feature(s) New
        to add to Act;
    4. if Act is large, prune to eliininate less useful features.
```

Figure 2: Concept learning with feature construction.

2 Feature Construction Based on SBL Output

FRINGE [Pagallo, 1989] and CITRE [Matheus, 1989] are two members of a class of concept learning algorithms *CL* that incorporate feature construction. Given a set of primitive attributes and corresponding data annotated with class-membership values, *CL* learns a concept by iterating two basic steps: (1) attempt to learn the concept through some standard induction method, and (2) use the output to construct new features and repeat the process. This algorithm schema also summarizes other approaches to feature construction (e.g., [Devijver and Kittler, 1982; Schlimmer, 1987; Seshu *et* a/., 1989]).

Figure 2 shows more detail. Although step 2 could use any attribute-based induction algorithm, we confine this discussion to SBL methods that output decision trees (e.g., [Breiman *et* a/., 1984; Quinlan, 1983]). To detect whether feature construction *FC* is appropriate, step 3 might base its decision on accuracy or the size of *Tree* [Matheus, 1989; Pagallo, 1990; Rendell and Seshu, 1990], but in our simplified approach, the loop's stopping criterion is failure to produce new features. Step 4 is important for difficult problems [Devyver and Kittler, 1982; Pagallo, 1990; Seshu *et al,* 1989], but is not part of the algorithms we test.

Henceforth, the acronym *FC* refers to feature construction based on decision *Tree* output of SBL. Since feature construction is complex, appropriate bias is essential. The following sections present increasingly involved attempts to utilize the structure of *Tree* to produce *useful* new features.

2.1 First Attempts at Effective Constraint

One scheme for construction is to form exactly one feature for each positive branch in *Tree.* Each branch produces a conjunction, e.g., Figure 1 would give three conjunctions, one of which is $x_1 \neg x_2 x_3 x_4$. This sort of construction is not very useful.

A more interesting technique is to retain the branch oriented approach but to restrict the conjunction to a binary operation. This restriction effectively allows multiple conjunction, but only after further iteration of the SBL procedure. On a variety of problem domains, Matheus [1989] tried three choices of operands for binary

```
FRINGE(Tree)
    New = NIL
    for every Leaf at depth $\geq 2$ in Tree
        if Leaf is a positive leaf then
            Feature = Conjoin(Leaf)
        New = New + Feature
    return(New)
```

Figure 3: FRINGE feature construction.

```
Conjoin(Leaf)
    if Leaf is left child of Parent then
        if Parent is left child of Grandparent then
            return($\neg$Parent & $\neg$GrandParent)
        else
            return($\neg$Parent & GrandParent)
    else
        if Parent is left child of Grandparent then
            return(Parent & $\neg$GrandParent)
        else
            return(Parent & GrandParent)
Disjoin(Leaf)
    if Leaf is left child of Parent then
        if Parent is left child of Grandparent then
            return($\neg$Parent $\vee$ GrandParent)
        else
            return($\neg$Parent $\vee$ $\neg$GrandParent)
    else
        if Parent is left child of Grandparent then
            return(Parent $\vee$ GrandParent)
        else
            return(Parent $\vee$ $\neg$GrandParent)
```

Figure 4: Binary operations used by *FC.*

conjunction: the two nodes at the *root* of a branch, the two at the *fringe* (the leaf and its predecessor [Pagallo, 1989]), and any two *adjacent* nodes. The best accuracy was usually attained using the Fringe method, although in some cases Adjacent had slightly better accuracy (but poorer efficiency).

2.2 FRINGE

Pagallo [1989; 1990] and Pagallo and Haussler [1990] reported results using a different set of schemes, but again the best choice was FRINGE (Fig. 3). (Pagallo applies this name to her entire algorithm analogous to *CL;* we use FRINGE to denote the feature construction scheme only.) Construction proceeds by conjoining the parent and grandparent nodes of all positive fringes in the tree; the position of the leaf node relative to the parent and grandparent determines whether the parent and grandparent occur negated in the conjunction (Fig. 4).

FRINGE alleviates the replication problem. In her analysis, Pagallo [1990] noted that even a simple boolean concept such as $C = x_1 x_2 \vee x_3 x_4$ causes replication. For example if we traverse the decision tree for *C* in Figure 1, we find $H = x_1 x_2 \vee \neg x_1 x_3 x_4 \vee x_1 \neg x_2 x_3 x_4$. Replication causes inconciseness and inaccuracy because *H* is divided into more disjunctive components than necessary, thus dispersing the training examples. In contrast, FRINGE

```
SymFringe( Tree)
    New = NIL
    for every Leaf at depth $\geq$ 2 in Tree
        Feature = Conjoin (Leaf)
        New = New + Feature
    return( New)
```

Figure 5: SymFringe (Symmetric FRINGE).

```
DCFringe(Tree)
    New = NIL
    for every Leaf at depth $\geq$ 2 in Tree
        if Leaf is a positive leaf then
            if (sibling of Leaf is a negative leaf) and
                (parent's sibling of Leaf is a positive leaf) then
                Feature = Disjoin(Leaf)
            else
                Feature — Conjoin(Leaf)
        New = New + Feature
    return(New;)
```

Figure 6: DCFringe, improved feature construction.

prevents dispersion by constructing conjunctive features such as x_3x_4, which tend to be selected early in another round of SBL.

Pagallo [1990] demonstrated the value of FRINGE for several DNF expressions, most of which used 16 to 80 attributes with 6 to 16 terms of length 3 to 7, and two of which had more terms. But she also noted limited success on parity concepts, and anticipated problems with CNF-type concepts. *CNF-type* refers to concepts whose CNF representations are more compact than the corresponding DNF-type concept expression.

2.3 Pagallo's Improvements to FRINGE

Pagallo [1989] proposed that the CNF problem can be attacked with a dual heuristic for negative leaves, using disjunction instead of conjunction. Later, Pagallo [1990] implemented an algorithm Symmetric FRINGE that combines the constructions of FRINGE and Dual FRINGE.

Figure 5 shows the symmetric version, whose name we abbreviate to SymFringe. Technically, SymFringe differs from Symmetric FRINGE in that SymFringe performs conjunction for both positive and negative leaves. However, the two versions produce the same features, modulo negation. Conjunction of positive leaves in SymFringe is identical to the FRINGE component of Symmetric FRINGE; conjunction of negative leaves in SymFringe is equivalent to negation of disjunctions in the Dual FRINGE component. Pagallo [1990] showed that Dual FRINGE and Symmetric FRINGE give better accuracy on two difficult CNF-type concepts that have more than a thousand terms when expressed in DNF form.

2.4 An Improved Construction Method

Shown in Figure 6, DCFringe is like SymFringe except that DCFringe guides construction using more detailed properties of the tree output by SBL. Whereas SymFringe chooses disjunction or conjunction according to

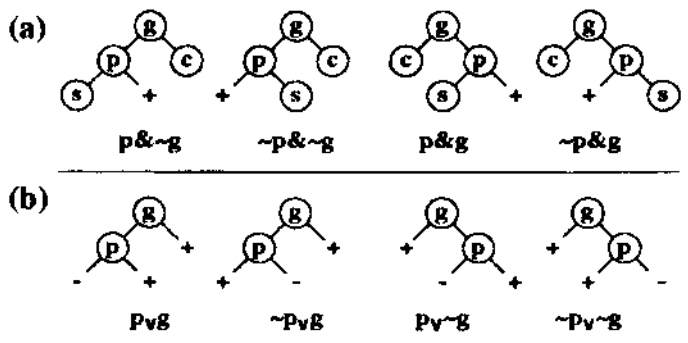

Figure 7: Patterns near the fringe of a decision tree. Right (Left) branch implies feature tests true (false).

the sign of a single leaf node, DCFringe makes its choice based on additional information. To understand the difference between SymFringe and DCFringe, consider first a simpler relationship between DCFringe and the original FRINGE.

Figure 7 depicts all possible patterns that can occur near the fringe of a binary decision tree. The crucial observation is that patterns reveal useful ways to combine features. Although both FRINGE and DCFringe use this observation, DCFringe considers more of the context in which the pattern occurs before deciding which feature to construct. FRINGE performs the construction shown in Figure 7a regardless of the node types of the sibling and the parent's sibling. In contrast, DCFringe constructs a disjunctive feature when the sibling is a leaf and the parent's sibling is a positive leaf, i.e., for patterns typical of trees representing CNF-type concepts as depicted in Figure 7b. Otherwise DCFringe performs the same conjunction operation as FRINGE. The signs within the new conjunction or disjunction depend on whether the current node lies in the left (false) branch or the right (true) branch of its parent node, and also the relative position of the parent node to the grandparent node (Fig. 4).

Compared with DCFringe, SymFringe is less selective because it forms all conjunctions and disjunctions, regardless of the tree structure. SymFringe produces more features, which can proliferate in multiple iterations.

2.5 Summary and Discussion of Methods

All our *FC* algorithms perform the binary operations of conjunction and/or disjunction, applying them to nodes in the decision tree output by SBL. The more interesting implementations are summarized in Table 1. Matheus' [1989] results demonstrate another interesting point. In addition to Root and Adjacent, he also tested heuristics such as Root-Fringe, which is the combination of Root and Fringe. The features generated by these variations have the following relationships: Fringe, Root $\subseteq$ Root-Fringe $\subseteq$ Adjacent. Intuitively, Adjacent should be most accurate since it generates a superset of the other heuristics' features. However, Matheus' results are not consistent with this intuition. For most of the cases, Adjacent is inferior to Fringe. Unnecessary features confuse SBL evaluation and cause overfitting, which sug-

Table 1: Four binary feature construction schemes.

Name	Construction Method	Application Condition(s)
Adjacent	Conjunction anywhere in branch	Nodes adjacent on branch
FRINGE	Conjunction at leaves	Positive leaves only
SymFringe	Conjunction	Positive and negative leaves
DCFringe	Conjunction or disjunction	Choice depends on pattern

gests that since DCFringe $\subseteq$ SymFringe, DCFringe may work better than SymFringe. In Section 3 we present and analyze experiments on this and other relationships among FRINGE, SymFringe and DCFringe.

3 Experimental Results and Analysis

We compare the behavior of three feature construction algorithms, by varying size of training sample and type of concept. We also discuss the strengths and weaknesses of the approaches.

3.1 Experimental Design

The three systems, FRINGE, SymFringe, and DCFringe, were run on 160 randomly generated concepts over 10 attributes. More precisely, 10 concepts each were generated from 16 pre-determined classes, each class a four-dimensional boolean choice. The boolean dimensions were DNF versus CNF, monotone versus non-monotone, m versus non-M, and 4/2 versus $2/4$.[1] An expression is monotone if all its literals are positive. A concept is M-DNF (μ-CNF) if each attribute occurs in at most one term (clause) of its DNF (CNF) expression [Ehrenfeucht *et ai,* 1988].

For each of the 160 target concepts, *N* training examples and 200 testing examples were generated such that no two examples were the same. The values of *N* were determined empirically to be within the range where DCFringe is most effective [Yang, 1991]; this gave values of N = 30, 60, and 90.[2] The underlying SBL system was PLS1, which performs much like ID3 [Rendell and Cho, 1990]. Although their splitting criteria differ, this factor has been shown to have a minor effect on the behavior of a decision tree learner [Breiman *et* a/., 1984; Mingers, 1989; Rendell and Cho, 1990].

In the following three sections we present and discuss results from our experiments with the three feature construction schemes. First we compare their overall performance; the criteria are predictive accuracy, learning efficiency, and tree conciseness. Then we investigate the effect of training set size on accuracy improvement. Finally, we show how learning behavior depends on concept //-ness, which leads to a discussion of the limitations of the *FC* algorithms studied.

[1] We use the notation *k/l* to signify a DNF expression having exactly *k* terms of exactly *l* literals, or a CNF expression having *k* clauses of *l* literals. Note that $2/4$ μ-CNF is equivalent to non-μ $16/2$-DNF, a $4/2$ μ-CNF equivalent to non-μ $16/4$-DNF.

[2] In continuing experiments, the size of the training sample is being varied over a wider range.

Table 2: Behavior of three construction algorithms.

	FRINGE	SymFringe	DCFringe
Accuracy (%)	93.7 ± 1.0	95.0 ± 0.9	96.5 ± 0.7
Acc. Improv. (%)	2.5 ± 1.0	4.2 ± 0.9	5.7 ± 0.7
# Leaves	8.7 ± 0.6	4.6 ± 0.4	4.8 ± 0.5
# Features	13.4 ± 1.6	30.1 ± 2.9	17.7 ± 2.0
# Iterations	2.7 ± 0.3	4.1 ± 0.4	4.3 ± 0.5

3.2 General Utility of the Different Algorithms

Table 2 provides a comparison of the overall performance of each algorithm. Values are averaged over all 160 runs for 90 training data. Each entry represents a 95% confidence interval for the corresponding algorithm. The first row shows final *predictive accuracy* (separate test sample) after convergence (several rounds of SBL and *FC).* The second row gives the difference between this value and the basic SBL algorithm. The third row indicates the *tree conciseness* as measured by the number of leaves in the final tree. The fourth and fifth rows indicate *learning efficiency* as the number of new features and the number of iterations before convergence.

FRINGE generates conjunctive features only, and is incapable of learning CNF-type concepts [Yang, 1991]. Since half of the 160 target concepts are CNF-type, FRINGE cannot perform well on average. SymFringe and DCFringe generate appropriate disjunctive features in addition to conjunctive features. They both perform better than FRINGE. DCFringe is even more accurate than SymFringe (the *t* value for improvement in accuracy is 1.7, which implies a confidence level of 90%).

In terms of efficiency, DCFringe is better than SymFringe. DCFringe generates significantly fewer features than SymFringe and uses almost the same number of iterations. FRINGE sacrifices accuracy and conciseness to be faster than DCFringe.

In terms of conciseness, SymFringe and DCFringe are significantly better than FRINGE. DCFringe and SymFringe have the smallest final trees. A reduction in tree size not only facilitates human comprehensibility, but also provides better statistical support for splitting decisions since the sample size within a node increases.

3.3 Accuracy Improvement vs. Sample Size

Table 3 shows how training set size influences accuracy for each algorithm. Here we consider the 80 M concepts only. The results for FRINGE are retained mainly as a baseline for comparison. The *t* values listed in the last column compare DCFringe with SymFringe. The entries under the algorithm names show the 95% confidence interval for *predictive accuracy improvement* beyond the basic SBL algorithm.

This table indicates that DCFringe significantly outperforms SymFringe in terms of accuracy. DCFringe

Table 3: Variation of accuracy improvement with sample

# Data	FRINGE	SymFringe	DCFringe	t
30	-0.5 ± 0.9[3]	1.0 ± 1.1	1.6 ± 1.1	0.79
60	2.3 ± 1.5	3.7 ± 1.6	5.9 ± 1.7	1.84
90	3.9 ± 1.6	5.5 ± 1.4	7.4 ± 1.4	1.97

has consistently better accuracy across training set size. (The *t* value of 1.8 in the second row corresponds to a significance level of about 97%, and the consistent trend substantiates the claim further.)

The extra features generated by SymFringe confuse the feature selection mechanism in the underlying SBL system. This effect and possible overfit cause SymFringe to perform less well than might be expected.

3,4 Accuracy vs. Concept Difficulty

Figure 8 shows the accuracy improvement obtained by DCFringe and SymFringe, as a function of concept /i-ness. DCFringe attains significantly better accuracy im-

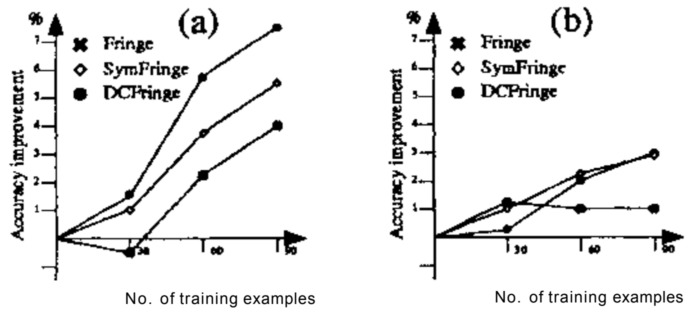

Figure 8: Accuracy improvement with SymFringe and DCFringe for μ concepts (a) and non-μ concepts (b)

provement than SymFringe for all training sets on /x concepts (Fig. 8a). But the tendencies are less clear for *non-fi* concepts (Fig. 8b). The superiority of DCFringe over SymFringe has vanished, and the relationship even seems to reverse, although not significantly.

Our conjecture to explain this phenomenon is that hard concepts have degrees of concept difficulty. From the perspective of the decision tree output of SBL, the replication problem is just one manifestation of difficulty. When concepts are μ DNF (μ CNF) the difficulty is restricted in two ways. First, the number of literals in a μ concept is limited to the dimensionality of the instance space, whereas concepts become more difficult for SBL roughly as the number of literals (attribute occurrences) increases [Ehrenfeucht *et a/.*, 1988; Rendell and Seshu, 1990]. Second, our preliminary experiments indicate that μ concepts often have replications, whereas non-μ concepts exhibit less replication (although they have obscure patterns of attribute duplication). On non-/i concepts, DCFringe is not more

[3]Accuracies relative to SBL are shown as percentage point differences.

```
Extended feature construction procedure FC/2.
  Using the SBL Tree, and possibly knowledge, create
    a set of patterns;
  Grouping these patterns by class-membership values
    and other constraints, form candidate pattern classes.
  Convert stronger pattern classes into new features
    to add to New.
  Return New to CL.
```

Figure 9: Refined construction from a decision tree.

accurate than SymFringe because the features generated by DCFringe help replications only.

Moreover, all three *FC* methods we evaluated improve accuracy only marginally with non-μ concepts. This elucidates the limitation of the basic FRINGE approach. The *FC* methods we tested are unlikely to generate useful features if the decision tree does not exhibit replication.

4 Discussion and Future Work

Our experimental analysis has raised several issues. One is the general utility of FRINGE-like algorithms. If they primarily help μ concepts, these algorithms have limited practical value. Of all possible boolean concepts, the proportion of non-μ concepts is large, and increases with the number of attributes. In hard practical problems, non-μ concepts (or their non-boolean analogues) often occur since low-level primitive attributes tend to participate repeatedly in many high-level features. To understand more precisely when FRINGE-like algorithms work, we need to address the problem of characterizing concepts. Although concept characterization is itself problematic [Ragavan and Rendell, 1990], the basic intuition is that concepts become more difficult as the numbers of literals and terms increase [Ehrenfeucht *et al.,* 1988], i.e., as feature interaction worsens [Rendell and Seshu, 1990].

Given a suitable measure of concept difficulty for SBL, and after using that measure to ascertain the limits of current feature construction, we anticipate several possible improvements to *FC.* For non-M concepts, the *Tree* regularities may be more complex than the replications enjoyed by DCFringe, requiring more subtle feature construction. Figure 9 outlines *FC/2,* an extended version of *FC,* for boolean and non-boolean problems for which we may have some knowledge or hunches.

Analogous to the conjunctive construction of *FC,* Step 1 of *FC/2* forms conjunctions as *patterns.* One way to obtain patterns is to conjoin attributes that occur frequently in multiple branches of the tree: a candidate pattern is favored if its attributes appear in many branches. A second way to decide good patterns may be combined with the first: Given a pattern proposed by the branch popularity method, one or more conjuncts may be dropped from or added to that initial pattern for various reasons. One reason for dropping a conjunct is to respond to overfit. Another reason for dropping or adding specific conjuncts is to match available knowledge or patterns from other branches, to make a more coherent set of patterns.

For hard problems with much feature interaction, correct patterns are difficult to determine because greedy SBL is limited even when aided by FRINGE-like algorithms [Pagallo, 1990]. We need to study the tradeoffs between relaxation of SBL greediness [Breiman *et al.,* 1984, chapter 5] and overall learning behavior.

Step 2 of *FC/2* forms *pattern classes,* which are potential disjunctions. A pattern class is a set of similar patterns. Two patterns are similar if their class-membership values are alike, and if they share some syntactic or semantic commonality to improve their *coherence* (i.e., unifying principle [Smith and Medin, 198l]). In real-world problems, pieces of knowledge are often available that can help form pattern classes [Matheus, 1990; Rendell and Seshu, 1990].

Especially when little knowledge is available for hard problems, *FC/2* will encounter extreme problems involving large numbers of patterns and pattern classes. Since large numbers of features aggravate problems of overfit and evaluation, we need sensitive measures of feature utility [Ragavan and Rendell, 1990]. Step 3 of *FC/2* should create new features only after they attain credibility in terms of support from data, pattern coherence, and general and specific knowledge. We also need effective means to combine multiple measures of support [Gunsch, 1991].

Acknowledgments

Discussions with Tom Dietterich, Chris Matheus, Eduardo Perez, Harish Ragavan, Raj Seshu, and Larry Watanabe contributed to this paper.

References

[Breiman *et al,* 1984] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[Devijver and Kittler, 1982] P. A. Devijver and J. Kittier. *Pattern Recognition: A Statistical Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[Ehrenfeucht *et aL,* 1988] Andrzej Ehrenfeucht, David Haussler, Michael Kearns, and Leslie Valiant. A general lower bound on the number of examples needed for learning. In *Proceedings of the conference on Computational Learning Theory,* pages 139-154, 1988.

[Gunsch, 1991] Gregg H. Gunsch. *Opportunistic Constructive Induction: Using Fragments of Domain Knowledge to Guide Construction.* PhD thesis, University of Illinois at Urbana-Champaign, 1991. Forthcoming.

[Matheus, 1989] Christopher J. Matheus. *Feature Construction: An Analytical Framework and an Application to Decision Trees.* PhD thesis, University of Illinois at Urbana-Champaign, December 1989.

[Matheus, 1990] Christopher J. Matheus. Adding domain knowledge to SBL through feature construction. In *Proceedings of the Eighth National Conference on Artificial Intelligence,* pages 803-808, 1990.

[Mingers, 1989] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning,* 3:319-342, 1989.

[Pagallo and Haussler, 1990] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning,* 5:71-99, 1990.

[Pagallo, 1989] Giulia Pagallo. Learning DNF by decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* 1989.

[Pagallo, 1990] Giulia Pagallo. *Adaptive Decision Tree Algorithms for Learning from Examples.* PhD thesis, University of California at Santa Cruz, June 1990.

[Quinlan, 1983] J. Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach,* chapter 15, pages 463-482. Tioga Publishing Co., Palo Alto, CA, 1983.

[Ragavan and Rendell, 1990] Harish Ragavan and Larry A. Rendell. Estimating the utility of feature construction in empirical learning. Unpublished manuscript, 1990.

[Rendell and Cho, 1990] Larry A. Rendell and Howard Cho. Empirical learning as a function of concept character. *Machine Learning,* 5(3):267-298, 1990.

[Rendell and Seshu, 1990] Larry A. Rendell and Raj Seshu. Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence,* 6:247-270, 1990.

[Schlimmer, 1987] Jeffrey C. Schlimmer. Learning and representation change. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* pages 511-515, 1987.

[Seshu *et al.,* 1989] Raj Seshu, Larry Rendell, and Dave Tcheng. Managing constructive induction using optimization and test incorporation. In *Proceedings of the Fifth International Conference on Artificial Intelligence Applications,* pages 191-197, Miami, FL, 1989.

[Smith and Medin, 198l] E. E. Smith and Doug L. Medin. *Categories and Concepts.* Harvard University Press, 1981.

[Utgoff, 1986] Paul E. Utgoff. Shift of bias for inductive concept learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II,* chapter 5, pages 107-148. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[Yang *et ah,* 199l] Der-Shung Yang, Gunnar Blix, and Larry A. Rendell. The replication problem: A constructive induction approach. In *Proceedings of the Fifth European Working Session on Learning,* Porto, Portugal, 1991.

[Yang, 1991] Der-Shung Yang. Feature discovery in decision tree representation. Master's thesis, University of Illinois at Urbana-Champaign, 1991. In preparation.