Integrating Feature Construction with Multiple Classifiers in Decision Tree Induction

Ricardo Vilalta
vilalta@cs.uiuc.edu

Larry Rendell
rendell@cs.uiuc.edu

Beckman Institute and Dept. of Computer Science
University of Illinois at Urbana-Champaign
405 N. Mathews Avenue, Urbana, IL 61801

Abstract

Feature construction can be attained by conducting a search for the best logical feature combination at every node of a decision tree. The exponential size of this search space, however, causes an instability: adding or removing few examples on each node subset tends to produce different best-feature combinations. We propose an approach to stabilize the feature construction mechanism by applying techniques that combine multiple classifiers (*bagging* and *boosting*) locally, at every node of a decision tree. As a result, each splitting function becomes a combination of multiple constructed features. We explore various alternatives to carry out this integrating approach, and perform an empirical evaluation of each method over a list of artificial and real-world domains. Finally, we discuss the utility of this approach when compared to a global application of multiple-classifier techniques, over whole decision trees.

1 INTRODUCTION

One technique to improve the quality of the hypothesis output by a supervised learning algorithm automates the construction of new features (i.e., attributes, variables) during learning, in order to attain progressively new example representations that more closely resemble the (hidden) structure of the target concept (Matheus, 1989; Rendell & Seshu, 1990). Feature construction has been successfully applied in domains such as financial applications and protein folding prediction (Ioerger, Rendell, & Subramaniam, 1995). A second learning technique, proved effective in augmenting the generalization performance of current inductive systems, combines several classifiers to predict the class of unseen examples (e.g., *bagging* (Breiman, 1996a), *boosting* (Freund & Schapire, 1996)). Combining multiple classifiers has consistently outperformed a single classifier alone, often with significant reduction of misclassification rates (Wolpert, 1992; Breiman, 1996a, 1996b; Quinlan, 1996; Buntine, 1991; Kwok & Carter, 1990; Freund & Schapire, 1996).

In this paper we integrate feature construction with multiple classifiers in the context of decision tree induction. Since universal learning is impossible (Schaffer, 1994), studying how different techniques can be integrated is important to identify effective guidelines in the design of learning systems that can perform successfully over the set of structured real-world domains.

Voting over multiple classifiers to predict the class of unseen examples complicates the interpretability of the output hypothesis (now represented as a combination of several hypotheses), thus suppressing an important advantage of symbolic learning models over other inductive models (e.g., over a connectionist model). We evaluate several approaches that integrate feature construction with multiple classifiers while retaining the interpretability of the output hypothesis. Hence, our goal is not only to improve the predictive capabilities of current learning systems, but to preserve our ability to extract knowledge or discover patterns relevant to the domain under study.

The organization of this paper follows. Section 2 provides background information on feature construction and on techniques to combine multiple classifiers. Section 3 defines our approach to integrate both techniques in the context of decision tree induction, and discusses why such integration is expected to yield increased accuracy. Section 4 reviews related work.

Section 5 presents and analyzes experimental results. Lastly, Section 6 gives a summary and conclusions.

2 PRELIMINARIES

Our study addresses domains where each example X is described by n boolean features $(x_1, x_2, \cdots, x_n)$, and where an underlying target concept $C : \{0,1\}^n \mapsto \{-,+\}$ classifies the space of all 2^n examples, also referred to as the *instance space*, into 2 classes. A learning system (i.e., inducer) A attempts to discover C by analyzing the information given by a training set $L : \{(X_i, c_i)\}_{i=1}^m$, where c_i is the class assigned by C to X_i, i.e., $C(X_i) = c_i$. We assume L is obtained by uniformly sampling examples that come from an underlying distribution D. The result of the analysis is a hypothesis/classifier H approximating C. $H_L(X)$ represents the class assigned by H to the unseen example X, where H is the hypothesis output by learning system A when trained on set L. Our main interest is in the ability of H to correctly *predict* the class of examples outside L. We look for hypotheses not only consistent with L, but that *generalize* beyond that set.

2.1 FEATURE CONSTRUCTION IN DECISION TREE INDUCTION

We review the internal mechanism of decision tree inducers and explain how feature construction can be incorporated into this learning model. Proceeding top-down, the root of a decision tree is formed by selecting a function $f : \{0,1\}^n \mapsto \{0,1\}$ that splits the training set L into mutually exclusive subsets S_0 and S_1, such that $S_0 = \{X \in L \mid f(X) = 0\}$, $S_1 = \{X \in L \mid f(X) = 1\}$, $L = S_0 \cup S_1$, and $S_0 \cap S_1 = \emptyset$. The same methodology is recursively applied to S_0 and S_1 to construct the left and right subtrees respectively. A node is terminal (i.e., a leaf) if the set of examples S covered by that node are all of the same class, or if $|S| < \beta$, where β is a predefined parameter. An example X is classified, starting from the root of the tree, by following the branch that matches the output of every splitting function (i.e., by following the left branch if $f(X) = 0$, or the right branch if $f(X) = 1$). At the end of the path, the class attached to that leaf is assigned to X.

The splitting function f is commonly a single feature, selected via some impurity measure, e.g. entropy, gini, Laplace, χ^2, etc., which yields axis-parallel partitions over the instance space. Our approach to feature construction views f as a logical combination of original features; this view implies a change in the input-language representation that allows more refined partitions over the instance space. In particular, our decision tree inducer, *DALI* (Dynamic Adaptive Lookahead Induction), conducts, at every tree node, a beam search over the space of all boolean-feature conjuncts or *monomials* (i.e. conjunction of boolean features or their complements). Figure 1 outlines *DALI*'s search mechanism at every tree node. To avoid exploring all possible states, *DALI* constrains the size of the search space through three main operations:

```
Algorithm 1: Search Mechanism in DALI
Input: A list of literals (i.e., boolean features and
their complements) $L_{\text{bool}} \leftarrow \{x_1, \bar{x}_1, x_2, \bar{x}_2, \cdots x_n, \bar{x}_n\}$
Output: Best combination $F_{\text{best}}$
DALI-SEARCH($L_{\text{bool}}$)
(1)     $L_{\text{beam}} \leftarrow$ best literals in $L_{\text{bool}}$
(2)     while (true )
(3)         $L_{\text{new}} \leftarrow$ Systematically form the conjunction
(4)         of every $F_i \in L_{\text{beam}}$ with every $F_j \in L_{\text{bool}}$
(5)         Apply pruning into $L_{\text{new}}$
(6)         if $L_{\text{new}} = \emptyset$
(7)             return global best combination $F_{\text{best}}$
(8)         $L_{\text{beam}} \leftarrow$ best combinations in $L_{\text{new}}$
(9)     end while
```

Figure 1: *DALI*'s Search Mechanism. At every tree node, the best logical combination (i.e., best monomial), is used as the next splitting function.

1. A systematic search to avoid redundant combinations (Rymon, 1993) (Lines 3-4, Fig. 1). Each combination (i.e., monomial) F_i conjoins several boolean features (or their complements), e.g., $F_i = x_1\bar{x}_3x_5$. Because conjunction is commutative, the search space is defined by avoiding any other combination F_j that is identical to F_i except for the order in which features appear, e.g., $F_j = \bar{x}_3x_5x_1$.

2. A pruning technique (Webb, 1995; Quinlan, 1995) (Line 5, Fig. 1). Define F_{best} as the best explored combination according to some impurity measure $\mathcal{H}$ (e.g., entropy), such that, for all explored F_i, $\mathcal{H}(F_{\text{best}}) < \mathcal{H}(F_i)$. As long as $\mathcal{H}$ is monotonic, a combination F_i can be eliminated if the best value F_i can ever attain along its search path – according to $\mathcal{H}$ – is worse than F_{best}.

3. A dynamic beam-width. At each level of search, only the best combinations are selected to expand the search (Lines 1 and 8, Fig 1). Specifically, we retain those combinations having a value of $\mathcal{H}$

within $\delta = \log_2(1 + \frac{1}{|S|})$ from the best combination at that level; $|S|$ is the number of examples covered by the current tree node. The value δ is simply a measure of the uncertainty that arises when the degree of impurity H is estimated on a sample of size $|S|$. Large training samples guarantee a confident estimate (i.e., reduce the uncertainty factor). This scenario is expected to occur in nodes near the root of the tree. Conversely, small training samples give large uncertainty factors, to compensate for the lack of statistical support near terminal nodes.

The search terminates when no more combinations can be generated, at which point the best explored combination, according to entropy, is used as the next splitting function. Instead of a single feature, a splitting function now conjoins several features (e.g., $x_1\bar{x}_2x_3$).

With few training examples or noisy datasets, looking for refined partitions over the instance space may result in *data overfitting* (i.e., in poor generalizations). In practice, however, enabling an inducer to construct non-orthogonal partitions often results in increased accuracy (Brodley & Utgoff, 1995). In addition, constructing new features at every tree node combats the fragmentation problem (Pagallo & Haussler, 1990) (but in a limited way (Vilalta, Blix, & Rendell, 1997)).

2.2 COMBINING MULTIPLE CLASSIFIERS

Multiple classifiers can be combined to improve classification. A learning system A is said to be *unstable* if small changes in the original training set L (i.e., adding or removing few examples in L) produce different hypotheses. Formally, let $L_S = \{L_1, L_2, \cdots, L_k\}$ be a set of training sets, each obtained by exerting some small changes in L. Applying A over each element in L_S produces hypotheses $H_{L_1}, H_{L_2}, \cdots, H_{L_k}$. In a typical scenario, A is applied to the original set L to output a single hypothesis H; unseen examples are classified according to the predictions made by H. But if A is unstable, H may be a poor representative of the set of hypotheses obtained from variants of L. Thereby, instead of being subject to the instability of A, one could generate a set of different training sets L_S from L, apply A to each element in L_S, and then classify a new example X by voting over the prediction of all output hypotheses (i.e., by voting over $H_{L_1}(X), H_{L_2}(X), \cdots, H_{L_k}(X)$). The classification of X is now based on a combination of different hypotheses or classifiers.

Using multiple classifiers for class prediction has several advantages: 1) the algorithm being applied is stabilized, so that less variability in performance occurs on different runs, and 2) the concept- language representation is expanded (to a combination of several classifiers), possibly rendering a better approximation to the target concept. A major disadvantage, however, is the difficulty introduced in the interpretability of the final hypothesis.

Figure 2 illustrates the mechanism for a general multiple-classifier learning algorithm. We focus on two recent approaches for combining classifiers. The first approach, named *bagging* (Breiman, 1996a), generates, k times, a variant or replicate of the original training set L by randomly sampling with replacement examples from L. Each variant approximates the underlying distribution D from which L originated, but may omit or duplicate some of the examples in L (i.e., each D_i gives equal weight to every example in L). Applying system A over each variant of L produces k hypotheses. An unseen example is assigned the majority class of the predictions made by these (equally weighted, i.e., $a_i = 1$ in Figure 2) hypotheses.

The second approach, named *boosting* (Freund & Schapire, 1996), differs from *bagging* in that each example is assigned a weight; a variant of L is generated by sampling with replacement under this weighted distribution (i.e., examples with higher weight increase their probability of being selected). This distribution is modified for every new variant of L, by giving more attention to those examples incorrectly classified by the most recent hypothesis (see reference for details). Thus, in contrast to *bagging*, *boosting* takes into consideration the accuracy of each hypothesis (over the original training set L) to progressively improve the classification of those examples in L for which the last hypothesis failed. As in *bagging*, an unseen example is classified on a majority-vote basis, but the vote of each hypothesis H_i is weighted (factor a_i, Figure 2) according to the accuracy of H_i on L.

3 INTEGRATING TECHNIQUES

Our approach to integrate feature construction with multiple classifiers incorporates both techniques locally, into a decision tree inducer. Current approaches to combining classifiers require a learning system A as input to generate several hypotheses (Figure 2); iterating over several decision trees to find new feature combinations is one form of feature construction (Pagallo & Haussler, 1990). In contrast, we view both techniques as local components of A (A seen as a decision

```
Algorithm 2: Combining Multiple Classifiers
Input: Learning system $A$, Number of iterations $k$,
Training set $L$
Output: Compound classifier $H_C$
COMBINE-CLASSIFIERS($A$,$T$,$L$)
(1)   foreach  $i = 1 \cdots k$
(2)       Set current distribution to $D_i$
(3)       Generate variant $L_i$ from $L$ according to $D_i$
(4)       Apply $A$ into $L_i$ to generate hypothesis $H_{L_i}$
(5)       Let $a_i$ measure the performance of $H_{L_i}$
(6)   end for
(7)   Let $H_C = \{a_i H_{L_i}\}_{i=1}^{k}$
(8)   return $H_C$
(9)   An unseen example is classified on a
(10)  majority-vote basis over $H_C$
```

Figure 2: A General Description of a Multiple-Classifier Algorithm

tree inducer), which permits analysis of their combined effect while maintaining a single working hypothesis.

Specifically, every node of a decision tree covers a subset S of examples of the original training set L, i.e., $S \subseteq L$. Slight changes in S may produce different splitting functions, according to the instability of the mechanism F employed to select the best candidate function. Therefore, techniques for combining multiple classifiers can be utilized locally, at each node of a decision tree, by generating variants of S, $S_s = \{S_1, S_2, \cdots, S_k\}$, and then applying F to each element of S_s to output different splitting functions $f_{S_1}, f_{S_2}, \cdots, f_{S_k}$. Example $X \in S$ is sent to the left or right branch of the current tree node by voting over $f_{S_1}(X), f_{S_2}(X), \cdots, f_{S_k}(X)$.

A direct correlation exists between the instability of F and the size of the space of all possible splitting functions. A large space increases F's instability because small changes in S may cause different splitting functions look superior. We use *DALI*'s search (Section 2.1) as the mechanism to generate different splitting functions from variants of S. *DALI*'s search mechanism explores the space of feature conjuncts, with size exponential in the number of original features. We expect such a space to produce instability on F when applied to variants of S, a problem that can be rectified by applying multiple-classifier techniques.

To reduce the instability of *DALI*'s search mechanism, we experiment with both *bagging* and *boosting* (Section 2.2). We apply each method at each tree node, by taking the set of examples in current node subset S as the training set L, and *DALI*'s search mechanism as the learning algorithm A (Figure 2). Each method generates k new features from variants of S, so that each splitting function f can now be seen as a linear feature combination $f(X) = a_1 f_1(X) + a_2 f_2(X) + \cdots + a_k f_k(X)$. In *bagging*, $a_i = 1$ (i.e., each a_i is constant); in *boosting* each a_i reflects the accuracy of f_i over subset S. In the latter case, each feature f_i is seen as a hypothesis over the examples in S. The majority class of the examples in the coverage of f_i, $\{X \in S \mid f_i(X) = 1\}$, is taken as the class predicted by f_i in that set. The accuracy of f_i is the proportion of examples in S being correctly classified by f_i. In either *bagging* or *boosting*, the branch to which an example $X \in S$ is directed depends on whether $f(X) > (0.5 \cdot \sum_i a_i)$ is true or false. Both methods output a single decision tree structure.

Since *bagging* ignores the performance of each generated hypothesis, we consider an additional possibility in which only the best feature from the k produced by *bagging* at each tree node is selected. We select the feature with lowest entropy by using the original example subset S as a testing set (Breiman, 1996a). The single output tree is identical to *DALI*'s trees, but each node comprises the best feature over all variants of subset S. Section 5 empirically evaluates all these different methods.

One drawback of integrating feature construction with multiple-classifier techniques is time complexity. *DALI*'s search mechanism is exponential in the number n of original features. This is further aggravated when the search is executed k times at every tree node, once for every variant of current subset S. The total time is $O(c^n kml)$, where c is a constant, m is the number of training examples, and l is the number of nodes in the decision tree. Nevertheless, the search for the best splitting function on every variant of S is amenable to parallelization. Section 5.2 reports running times on the systems tested.

4 RELATED WORK

DALI, as described in Section 2.1, is a successor of the *LFC* system (Ragavan & Rendell, 1993). Both *DALI* and *LFC* are decision tree inducers that conduct a beam search over the space of all boolean-feature conjuncts at every tree node. But while in *LFC* the search space is limited by user-defined width and depth, *DALI* obviates these parameters by extending the search depth until no more combinations can be generated, and selects the beam width dynamically. *DALI* exhibits a faster response time than *LFC*, with similar performance in terms of predictive accuracy.

Voting over multiple features is equivalent to evaluating a Boolean threshold function, and is akin to the constructive induction approach of M-of-N concepts (Murphy & Pazzani, 1991). Our approach, however, combines features in order to reduce the instability that stems from variations on the training set when the space of candidate splitting-functions is large.

Breiman (1996) measures the utility of *bagging* over the *CART* system (Breiman, Friedman, Olshen, & Stone, 1984) reporting significant gains in predictive accuracy. His study reveals that *bagging* can improve performance only if the algorithm being applied is unstable (Section 2.2). For stable algorithms *bagging* can even degrade performance.

Freund & Schapire (1996) experimentally assess their *boosting* algorithm, *AdaBoost*, and compare it to *bagging*. They conclude that when applied over *weak* classifiers, *boosting* shows uniformly superior to *bagging*, but the two approaches perform similarly when the classifier employed is *strong*. Quinlan (1996) compares *bagging* and *boosting* using his *C4.5* system, and finds *boosting* significantly better than *bagging*. Both studies show the advantage of using multiple classifiers over a single classifier alone.

Our approach to applying multiple-classifier techniques locally differs from previous methods, in which globally invoking a learning system to produce different hypotheses ignores the inherent deficiencies of the learning system itself. But by viewing these techniques as local learning components, we are in better position to understand and control their effect during the generalization phase.

5 EXPERIMENTS

We refer to combining *bagging* and *boosting* with *DALI*'s search mechanism at every decision tree node as *DALI-Node-Bag* and *DALI-Node-Boost* respectively (Section 3). For comparison purposes, the same strategies are applied (as typical) in a global fashion, over the decision trees output by *DALI*, here referred to as *DALI-Tree-Bag* and *DALI-Tree-Boost* respectively. Both *DALI-Tree-Bag* and *DALI-Tree-Boost* output multiple decision trees, thus complicating the interpretability of the final hypothesis. Selecting only the lowest-entropy feature at every tree node in *DALI-Node-Bag* is named *DALI-Node-Best-Bag*, and selecting only the most accurate tree from *DALI-Tree-Bag* (using the original training set as testing set) is named *DALI-Tree-Best-Bag*. We also report on *DALI* alone, to isolate the feature-construction component, and on *C4.5*-trees (Quinlan, 1994), as a simple decision tree inducer.

5.1 METHODOLOGY

Our experiments use both artificial concepts and real-world domains. The artificial concepts are defined on nine and twelve features (see Appendix A), and include DNF formulae, CNF formulae, Multiplexor (MUX), Majority (MAJ), and Parity (PAR). We report on 10% training-set size for 9-feature concepts, and 5% training-set size for 12-feature concepts. Each reported value is the average over 50 runs; predictive accuracy is computed as the percentage of correct classifications for all examples outside the training set.

Experiments on real-world domains estimate predictive accuracy by using stratified 10-fold cross-validation (Kohavi, 1995), averaged over five repetitions. Since *DALI* is limited to boolean domains, we performed an initial discretization step on all numeric (following Catlett (1991)) and nominal features (constructing a boolean feature for each nominal value).

All systems were set to default parameters. For *C4.5*-trees we chose the best between the pruned and unpruned tree. We set to $k = 50$ the number of iterations for *bagging* and *boosting*. On each concept, we mark with an asterisk the system with best accuracy performance; two asterisks mean the system outperforms all others significantly at the $p = 0.05$ level (using a two-sided t-test). Runs were performed on a SPARC-station 10/31. All databases are accessible through the UCI database repository[1], except for the star-cluster domain (Table 22.1 in Andrews and Herzberg (1985)).

5.2 RESULTS ON ARTIFICIAL DOMAINS

Table 1 depicts results on all artificial domains in terms of predictive accuracy. Both 9 and 12-feature concepts are grouped separately and arranged on increasing difficulty (Rendell & Seshu, 1990). When *bagging* and *boosting* are applied locally (i.e., applied at each node of *DALI*'s tree), *DALI-Node-Bag* seems to perform worse; forming a linear combination of features at each node without recurring to any pruning technique may cause data overfitting. Both *DALI-Node-Boost* and *DALI-Node-Best-Bag* behave similarly, with no apparent advantage over *DALI* alone. On moderate parity (PAR9a and PAR12a), however, *DALI-Node-Best-Bag*

[1]http://www.ics.uci.edu/ mlearn/MLRepository.html

Table 1: Tests on Predictive Accuracy for Artificial Concepts.

Concept	C4.5	DALI	DALI-node (local)			DALI-tree (global)		
			boost	bag	best-bag	boost	bag	best-bag
DNF9*a*	92.01	99.65	99.50	98.61	99.65	99.65	99.30	99.86*
CNF9*a*	97.98	99.17	99.43	98.58	99.17	99.34	98.83	99.44*
MAJ9*a*	98.77	97.84	97.78	95.00	97.98	99.03	98.49	99.43*
MAJ9*b*	76.96	80.94	86.05*	81.70	80.93	85.56	84.87	80.23
CNF9*b*	84.40	90.97	91.66	85.60	92.36	93.09	90.22	94.91**
MUX9	73.35	86.16	86.23	74.77	84.76	87.71	84.04	88.09*
DNF9*b*	72.13	86.41	85.34	76.84	85.36	85.80	83.07	90.36**
PAR9*a*	59.40	79.70	74.88	63.40	91.50**	83.08	74.98	88.09
PAR9*b*	47.17*	45.50	45.77	46.37	46.19	44.31	44.43	46.31
MAJ12*a*	97.63	100.0*	99.98	99.65	100.0*	100.0*	100.0*	100.0*
DNF12*a*	95.99	99.18	97.32	94.80	99.48	99.34	99.58	100.0*
CNF12*a*	94.90	100.0*	100.0*	100.0*	100.0*	100.0*	100.0*	100.0*
MAJ12*b*	80.79	83.33	91.80*	79.49	82.76	91.14	88.87	82.22
CNF12*b*	80.83	96.92	95.89	89.18	96.54	99.73	98.46	100.0*
DNF12*b*	82.79	95.87	92.25	84.89	97.00	95.70	97.13	97.83*
MUX12	72.42	89.63	82.01	67.28	92.91*	90.72	91.17	90.04
PAR12*a*	57.68	79.09	58.59	52.48	95.66**	70.77	69.03	80.57
PAR12*b*	48.48*	47.70	48.32	48.76	47.81	46.72	47.00	48.20

shows significant improvement over all other competitors. This suggests that the difficulty of certain domains (e.g., parity) given a similarity-based bias may be alleviated by selecting best features from variations of sets of training examples. Also, we observed the number of different feature combinations comprised at every tree node is often less than $k = 50$. This facilitates interpretation of the splitting function, and occurs because the same combinations repeat on different subset variations.

A global application of *bagging* and *boosting* (i.e., applied over *DALI*'s trees) reveals that *DALI-Tree-Boost* is more accurate than *DALI-Tree-Bag* on most domains, which agrees with results on previous comparisons (Quinlan, 1996; Freund & Schapire, 1996). All global applications of *bagging* and *boosting* degrade CPU-time to around three orders of magnitude when compared to *DALI*; on both artificial and real-world domains, a single run could consume several hours. Local applications of *bagging* and *boosting* degrade CPU-time to one order of magnitude (except for *DALI-Node-Bag* at approximately two orders of magnitude).

In general, the improved performance displayed by *DALI-Node-Best-Bag* and *DALI-Tree-Best-Bag* under artificial conditions (e.g., no noise, sufficient examples) is noteworthy, but does not hold on real-world applications (Section 5.3). One reason may be the advantage obtained when more refined partitions are produced over the instance space, through the explicit combination of multiple features or classifiers.

5.3 RESULTS ON REAL-WORLD DOMAINS

When features describing examples bear a low level representation to the target concept, so that complex combinations need to be discovered before a proper hypothesis is output, the domain is said to have high *feature interaction.* We attempt a characterization of the real-world domains used for our study in terms of feature interaction, in order to explain the results depicted in Table 2.

Specifically, we observe a global application of *bagging* and *boosting* over *DALI* performs particularly well on domains exhibiting high interaction. Examples of these domains are chess-end game (KrvsKp) and Tic-Tac-Toe, where features describing board configurations are the only available information to determine win or loss; using sequences of nucleotides to identify promoter instances; or inferring conditions of the ionosphere with complex numbers obtained from electromagnetic signals. We claim an extension of the concept-language representation in decision tree induction —through both the combination of multiple trees and the construction of new features— enables the discovery of the intricate structure of these concepts.

Our approach to applying *bagging* and *boosting* locally exhibits a general advantage in domains with medium interaction. These domains comprise features that stand closer to the target concept being approximated, such that less difficulty is expected in elucidating concept interactions, as when distinguishing

Table 2: Tests on Predictive Accuracy for Real-World Domains.

Concept	C4.5	DALI	DALI-node (local)			DALI-tree (global)		
			boost	bag	best-bag	boost	bag	best-bag
High Interaction								
TicTacToe	85.80	98.40	96.11	93.71	97.77	99.73^*	98.74	99.28
KrvsKp	96.20	94.87	96.07	95.60	94.67	96.87	97.00^*	94.87
Promoters	81.86	83.40	88.20	87.67	84.67	89.00	92.20^{**}	83.20
HeartC	76.16	77.44	78.83	77.66	75.10	79.31	81.17^{**}	75.17
Ionosphere	91.22	91.31	92.29	53.14	91.52	63.97	92.57^*	90.86
Medium Interaction								
Cancer	95.20	95.68	96.74^*	96.23	95.74	65.21	96.47	95.26
Lympho[2]	83.46	87.86	84.00	88.29	87.71	87.71	88.57^*	88.00
Lympho[3]	77.54	84.57	85.43	87.57^*	85.29	87.43	87.00	84.14
NThyroid [Hyper]	96.78	96.19	97.52^{**}	96.38	96.48	84.76	96.10	96.00
NThyroid [Hypo]	96.40	96.29	95.71	96.67^*	96.00	86.86	96.38	96.38
StarCluster	98.40	98.38	98.19	98.48^*	98.29	68.00	98.48^*	98.19
Low Interaction								
Mushroom	100.0^*	99.80	99.92	99.84	99.80	99.88	99.84	99.82
CreditApp	86.36^{**}	80.71	84.31	84.71	80.25	84.17	85.51	80.98
Hepatitis	82.32^*	80.18	81.09	82.00	80.00	80.18	82.00	78.55
LiverDis	67.14^{**}	65.18	65.26	65.06	64.23	57.68	66.00	63.94

benign from malign cancer based on characteristics of the diseased cell (Cancer, Lymphography); finding thyroid disease according to medical-record contents; or estimating star-cluster membership on spatial coordinates, relative motion, and luminosity. In this approach, a change of representation is carried out to improve on each splitting function, but the combination of whole classifiers is omitted. When compared to *DALI* alone, however, both *DALI-Node-Bag* and *DALI-Node-Boost* show improved performance on most domains.

Finally, Table 2 shows a disadvantage of any form of change of representation when the domain under study is simple (i.e., has low feature interaction), e.g., predicting if a mushroom is edible (or poisonous) based on direct characteristics of the plant (color/shape); or diagnosing hepatitis or liver disorder based on blood-test results. The features describing examples in these domains are known to be highly correlated with the concept being learned; searching for complex interactions results in data overfitting.

Our results confirm the importance of an automatic adaptation of learning bias to respond for a need of change of representation during learning (Romaniuk, 1994; Matheus, 1989). Simple domains are better off without any constructive mechanism, whereas a local or global integration of feature construction with multiple classifiers has a greater utility on more complex domains (in the former case while retaining the interpretability of the final hypothesis).

6 SUMMARY AND CONCLUSIONS

Finding logical combinations of features at every node of a decision tree is one approach to bridge the gap between the input-language representation and the (possibly) complex structure of the target concept. We apply multiple-classifiers techniques (*bagging* and *boosting*) at every node of a decision tree to tackle the instability effect expected over large search spaces, where small variations on each subset node results in different best-feature combinations.

Our empirical results show different effects when multiple-classifier techniques are integrated with feature construction on either a local or global fashion. These differences are explained according to the degree of feature interaction characterizing the domain under study, and stem from the degree of sophistication of each approach. A local application of *bagging* and *boosting* over *DALI* refines the feature construction mechanism at each node exclusively (i.e., exerts a change of representation locally), and shows improved performance (for *DALI-Node-Bag* and *DALI-Node-*

Boost) over *DALI* alone on most real-world domains. A global application augments the concept-language representation to a combination of multiple classifiers (i.e., exerts a change of representation locally and globally), and has the greatest advantage over domains characterized by high-feature interaction. This suggests that increasing the degree of change of representation during learning must be guided according to the characteristics displayed by the domain under study. Without such guiding mechanism we cannot hope to attain a learner exhibiting uniformly good performance over the set of structured real-world domains. The amount of feature interaction can decide the level of refinement of the partitions drawn over the instance space. Ideally, increasing the degree of change of representation would result from the interaction of local learning components within a single learning system, and thus would not impede us from extracting useful pieces of knowledge from the final output hypothesis, as is achieved within our approach.

Acknowledgments

We thank Prof. Leo Breiman for his many fruitful suggestions. We are grateful to the anonymous reviewers and all people who have contributed into the UCI Machine Learning repository. Our work was supported in part by National Science Foundation (IRI 92–04473) and Consejo Nacional de Ciencia y Tecnología (México).

Appendix A. Definitions for Artificial Concepts

Let: $X = (x_1, x_2, \ldots, x_n)$,
$\text{address}(x_1, x_2, \ldots, x_m) = 2^0 x_1 + 2^1 x_2 + \ldots + 2^{m-1} x_m$

Definitions:

DNF9*a* :	$x_2x_3 + \bar{x}_2x_3x_7 + x_2\bar{x}_3x_8 + x_2\bar{x}_7\bar{x}_8 + \bar{x}_3x_7x_8$
DNF9*b* :	$x_1x_2x_3 + \bar{x}_1\bar{x}_2 + x_7x_8x_9 + \bar{x}_7\bar{x}_9$
DNF12*a* :	$x_1x_2x_9x_{12} + x_1x_2\bar{x}_9x_{11} + \bar{x}_1x_2x_5x_9 + \bar{x}_1\bar{x}_2x_8\bar{x}_9$
DNF12*b* :	$x_1x_2\bar{x}_7\bar{x}_8 + x_1x_2x_{11}x_{12} + x_1\bar{x}_2\bar{x}_{11}x_{12} + \bar{x}_1x_2x_{11}\bar{x}_{12} + \bar{x}_1\bar{x}_2\bar{x}_{11}\bar{x}_{12} + x_7x_8x_{11}x_{12}$
CNF9*a* :	$(x_4 + x_6)(\bar{x}_4 + \bar{x}_5 + x_7)$
CNF9*b* :	$(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_5)(x_1 + x_2 + \bar{x}_3)$
CNF12*a* :	$(x_8 + x_9)(x_7 + x_6 + x_5)(\bar{x}_8 + \bar{x}_6 + x_2)$
CNF12*b* :	$(x_1 + x_2 + x_6)(\bar{x}_1 + \bar{x}_2 + x_7)(x_9 + x_{10} + x_{12})(\bar{x}_6 + \bar{x}_7 + x_8)(x_9 + \bar{x}_{10} + \bar{x}_12)(\bar{x}_8 + \bar{x}_9 + \bar{x}_7)$
MAJ9*a* :	$\{X \mid (\sum_{i=1}^{9} x_i) \geq 3\}$
MAJ9*b* :	$\{X \mid (\sum_{i=1}^{9} x_i) \geq 6\}$
MAJ12*a* :	$\{X \mid (\sum_{i=1}^{12} x_i) \geq 4\}$
MAJ12*b* :	$\{X \mid (\sum_{i=1}^{12} x_i) \geq 8\}$
PAR9*a* :	$\{X \mid (\sum_{i=1}^{3} x_i \bmod 2) > 0\}$
PAR9*b* :	$\{X \mid (\sum_{i=1}^{6} x_i \bmod 2) > 0\}$
PAR12*a* :	$\{X \mid (\sum_{i=1}^{4} x_i \bmod 2) > 0\}$
PAR12*b* :	$\{X \mid (\sum_{i=1}^{8} x_i \bmod 2) > 0\}$
MUX9 :	$\{X \mid x_{(\text{address}(x_1,x_2)+3)} = 1\}$
MUX12 :	$\{X \mid x_{(\text{address}(x_1,x_2,x_3)+4)} = 1\}$

References

Andrews, D., & Herzberg, A. (1985). *Data: A Collection of Problems from Many Fields for the student and Research Worker.* Springer-Verlag.

Breiman, L. (1996a). Bagging predictors. *Machine Learning, 24*, 123–140.

Breiman, L. (1996b). Stacked regressions. *Machine Learning, 24*, 49–64.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth, Belmont, CA.

Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning, 19:1*, 45–78.

Buntine, W. (1991). Learning classification trees. In *Artificial Intelligence Frontiers in Statistics*, pp. 182–201. D.J. Hand, Chapman and Hall, London.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *Proceedings of the Fifth European Working Session on Learning*, pp. 164–178. Springer-Verlag.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156. San Francisco: Morgan Kaufmann.

Ioerger, T. R., Rendell, L. A., & Subramaniam, S. (1995). Searching for representations to improve protein sequence fold-class prediction. *Machine Learning, 21*, 151–175.

Kohavi, R. (1995). A study of cross validation and boostrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1137–1143. Morgan Kaufmann.

Kwok, S. W., & Carter, C. (1990). Multiple decision trees. In *Uncertainty in Artificial Intelligence*, Vol. 4, pp. 327–335. Elsevier Science Publishers.

Matheus, C. J. (1989). *Feature Construction: An Analytical Framework and an Application to Decision Trees.* Ph.D. thesis, University of Illinois at Urbana-Champaign.

Murphy, O. J., & Pazzani, M. (1991). Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 183–187. San Francisco: Morgan Kaufmann.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5*, 71–99.

Quinlan, J. R. (1994). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

Quinlan, R. (1995). Oversearching and layered search in empirical learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1019–1024. Morgan Kaufmann.

Quinlan, R. (1996). Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730. MIT Press.

Ragavan, H., & Rendell, L. A. (1993). Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 252–259. San Francisco: Morgan Kaufmann.

Rendell, L. A., & Seshu, R. (1990). Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence*, *6*, 247–270.

Romaniuk, S. (1994). Learning to learn: Automatic adaptation of learning bias. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 871–876. MIT Press.

Rymon, R. (1993). An SE-tree based characterization of the induction problem. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 268–275. San Francisco: Morgan Kaufmann.

Schaffer, C. (1994). A conservation law for generalization performance. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 259–265. San Francisco: Morgan Kaufmann.

Vilalta, R., Blix, G., & Rendell, L. A. (1997). Global data analysis and the fragmentation problem in decision tree induction. In *9th European Conference on Machine Learning.* Springer-Verlag.

Webb, G. I. (1995). Opus: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, *3*, 431–435.

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.