Learning Decision Rules by Randomized Iterative Local Search

Michael Chisholm CHISHOLM@MITRE.ORG
The MITRE Corporation, 202 Burlington Rd. M/S K302, Bedford MA, 01730, USA
Prasad Tadepalli TADEPALL@CS.ORST.EDU
Department of Computer Science, Oregon State University, Corvallis, OR 97331, USA

Abstract

Learning easily understandable decision rules from examples is one of the classic problems in machine learning. Most learning systems for this problem employ some variation of a greedy separate-and-conquer algorithm, which makes the rules order-dependent, and hence difficult to understand. In this paper, we describe a system called LERILS that learns highly accurate and comprehensible rules from examples using a randomized iterative local search. We compare its performance to C4.5, RIPPER, CN2, G-NET, Smog, and BruteDL, and show that it compares favorably in accuracy and simplicity of hypotheses in a number of domains.

1. Introduction

Decision rules offer many advantages as a representation language for machine learning. These include human-comprehensibility, their relationship to logic, their suitability for inference and for building expert systems, and their generalizability to first-order representations (Cohen, 1995). As a result, there is a significant amount of effort devoted to learning easily understandable decision rules from examples (Weiss & Indurkhya, 1991; Cohen, 1995; Cohen, 1999). However, with the exception of a few, e.g. Weiss, Galen, & Tadepalli (1990), most of these systems use some variation of a greedy separate-and-conquer algorithm. Almost all of these systems also make accuracy on the test set their only measure of performance. While greedy algorithms are time-efficient, it is unclear if the accuracy and comprehensibility can be improved by more exhaustive search for the simplest and most accurate rules.

Unfortunately, an exhaustive search of the rule space for the best rules is impractical except for highly restricted spaces. Hence, we need some constraints and heuristics to guide the search. Systems that rely on semi-exhaustive search for learning, such as neural networks, use some sort of randomized local search and produce highly accurate results (Bishop, 1996). Neural networks, however, are far from comprehensible, and extracting comprehensible rules from them is challenging. One relevant question, then, is whether it is possible to learn highly accurate and comprehensible rules by adopting more exhaustive randomized local search algorithms similar to those used by neural networks.

Randomized local search has been found to be beneficial in other areas of AI. For example, GSAT and Walksat algorithms solve hard satisfiability problems of thousands of variables using randomized multiple-restart hill climbing with simple evaluation functions (Selman, Kautz, & Cohen, 1994). These algorithms have also been successfully applied to planning, scheduling, and constraint satisfaction by encoding them as satisfiability problems (Selman & Kautz, 1996). However, it is fair to say that randomized local search has not made a big impact so far on the design of algorithms in symbolic machine learning. One of our goals is to bridge this gap and explore the viability of these algorithms for machine learning.

Human comprehensibility has recently gained importance as a desirable property of learned rules (Quinlan, 1999). Humans can learn from the hypotheses output by a computer, provided the hypothesis is in a human-comprehensible form. Human experts can also participate in the learning process by critiquing or improving the learned hypothesis if it is understandable to them. Decision rules appear to be more human-comprehensible than decision trees, more so if they are independently accurate, i.e., each rule is accurate regardless of its position in a list of rules. Unfortunately, learning multiple classes by “separating and conquering” each class of examples violates this property, since the examples in each class are removed before learning rules for the next class. This is the reason we avoid it.

Our system, called LERILS, has two parts. The first part finds rules and adds them to a large rule pool. The second part combines the rules in the rule pool to form a final ruleset. Both parts use randomized local search with a version of a minimum description length heuristic to trade off rule/ruleset accuracy with simplicity (Rissanen, 1984). The algorithm can handle missing attribute values and real-valued attributes.

We used 11 datasets to compare the performance of our algorithm with RIPPER, C4.5 (Quinlan, 1993), and CN2 (Clark & Niblett, 1989). We compare our performance to Smog and G-NET on 7 datasets. Our algorithm was comparable RIPPER, significantly better than CN2 on 3 datasets, and significantly worse on 2. It was significantly better than C4.5 on 4 datasets and significantly worse on only one. LERILS's hypotheses are similar in size to RIPPER's, but without the rule order dependence. CN2 was found to have excessively large hypotheses. From this we conclude that LERILS's hypotheses are the most comprehensible of the 4 algorithms tested. LERILS also compares favorably with G-Net and Smog in all but two of 7 domains tested and with BruteDL in 3 of the 4 domains tested.

2. Related Work

There are several rule learning algorithms in the machine learning literature, most of which use some form of greedy heuristic and separate-and-conquer strategy. For example, CN2 conducts a general-to-specific search through a space of rules by adding conditions, e.g. Sky=*Sunny*, to rules and using a statistical significance test and an entropy measure to keep track of the best rule found so far (Clark & Niblett, 1989).

Swap-1 (Weiss & Indurkhya, 1991) builds its rules greedily by removing as well as adding literals during the rule induction phase. To avoid overfitting, it uses weakest link pruning (the link that would result in the fewest new errors when pruned) with cross-validated data. The high computational complexity of this "reduced error pruning" motivated the design of RIPPER, which prunes rules incrementally (Cohen, 1995). First, rules are induced greedily. Unlike Swap-1, literals may only be added to the end of a rule, not deleted or swapped out. A rule is greedily pruned immediately after it is created. All three systems, CN2, Swap-1, and RIPPER, use separate-and-conquer to learn multiple rules, which increases rule dependence and decreases comprehensibility.

SLIPPER improves upon the accuracy of RIPPER by using a form of boosting (Schapire & Singer, 1998), where the weak hypotheses are rules grown and pruned similarly to RIPPER. Instead of separating the training examples covered by a rule, the probabilities of their selection is changed so that the examples which are incorrectly classified by the current rule have a better chance of being selected in the next round. Each rule is assigned a confidence rating, and classification of a test example is done by a form of confidence-weighted majority. LRI (Weiss & Indurkhya, 2000) is similar to SLIPPER in that it also uses a form of boosting. However LRI's rules are DNF formulas, and the final classification is done by simple majority vote among the rules. Both these programs produce highly accurate rulesets. However, the majority function at the top level in both these programs would make the hypotheses more expressive and so less comprehensible than simple sets of rules.

Segal and Etzioni (1994) introduce a condition called "homogeneity" which implies order-independence. A homogeneous rule is one whose accuracy does not change for any specialization. They describe an algorithm called BruteDL that tries to achieve comprehensibility via rule accuracy and order-independence, by doing a depth-bounded exhaustive search of the space of "homogeneous" rules. The goals of LERILS and BruteDL are similar, although the algorithms are quite different, with interesting consequences. We will discuss these in Section 4.

3. Rule Learning Algorithm

This section describes our learning algorithm in detail and analyzes its asymptotic complexity. Our algorithm is motivated by the need to learn accurate and comprehensible rules. This implies that the rules should be simple and few in number and should be independently accurate. This rules out learning multiple classes by separate-and-conquer, and boosting that relies on majority voting of several low accuracy rules.

3.1 Overview

LERILS iteratively works on each class, and operates in two stages:

1. Create a large pool of rules.
2. Try to find an optimal subset of these rules.

In the first stage, positive examples are picked at random, and rules are created from them. These rules are then refined by doing a random walk, using a description length heuristic to guide the search. The goal is to get a variety of good rules to choose from during the second stage. In the second stage, rulesets are created from random subsets of the rules in the rule pool. These rulesets are then refined by doing a greedy walk using another description length heuristic. These two stages are repeated for each class, and then the final rulesets are greedily ordered to produce the final list of rulesets.

3.2 Rule Generation

Rule generation begins by picking a random positive seed example. A rule is essentially represented as an example with extra relevance bits to indicate which literals of the example are a part of the rule. Each literal in the seed example has a 50% chance of being relevant in the initial rule created from it. The seed example always remains covered to ensure a good variety of rules. As an illustration of this, consider the 4-attribute example

Table 1

Sky	Temp	Time	Wind	EnjoyTennis
Sunny	*Cool*	?	*NotWindy*	*Yes*

where EnjoyTennis is the class attribute and ? represents a missing attribute, and consider the bit pattern 1001. The rule represented by these would be EnjoyTennis=*Yes* :- Sky=*Sunny*, Wind=*NotWindy*. Since the seed example has its third attribute missing, the third bit in the relevance bitset must always be 0. During the random walk, with probability p_1, a random relevance bit is flipped. With probability $1-p_1$, that bit is flipped which minimizes the description length of the rule. At the end of the walk, the best rule found is added to the rule pool. In the experiments of this paper, we used $p_1 = .35$ and added 100 rules to the rule pool.

The number of steps taken during the random walk is dependent on the number of attributes, a, in each example. More attributes require a longer walk. We have been using $3a$ steps in these experiments.

The relative description length of a rule is computed as $(s+n)/p$, where s is the number of relevant literals (rule size), n is the number of false positives (negative examples covered), and p is the number of positive examples covered by the rule. Missing attributes in the seed example are kept permanently irrelevant in the rule. Note that unlike RIPPER and CN2, after a rule is created and added to the rule pool, the examples it covers are *not* removed from the training example set. So every rule is built using all the training examples. This is necessary because it is not decided which rules are actually used in the final hypothesis until the second stage. Continuous attributes require special handling, as described later.

3.3 Ruleset Generation

Duplicate rules are first removed from the rule pool. A ruleset is simply a collection of rules for the same class, and initially it contains a random subset of the rule pool. Every rule in the rule pool has a 50% chance of being included in the initial ruleset. So on average, the initial ruleset contains half the rules in the rule pool. The choice of which rule to add or remove from the ruleset is purely greedy: that rule is removed or added to the ruleset that minimizes the resulting description length of the ruleset. A pure greedy method was found to be faster than doing randomized walking for a fixed number of steps, and the quality of the rulesets generated didn't decrease significantly. The ruleset description length is computed as $e + l$, where e is the total number of exceptions, positive and negative, to the ruleset, and l is the total number of literals in all the rules in the ruleset. Adding a count of literals to the description length causes the learner to prefer fewer and more general (i.e. those with fewer literals) rules to more specific rules. The walk continues as long as the ruleset description length continues to decrease. There are a fixed number of restarts, where a new ruleset is randomly chosen from the rule pool, and the greedy walk is begun again. This compensates for the lack of randomness in the walks. We used 5 restarts for each ruleset. The best ruleset found during all steps of all greedy walks becomes the final ruleset for the class.

To resolve classification conflicts, the rulesets are ordered in a decision list structure. The class of the first rule to cover the example becomes the example's predicted class. Ordering of rulesets is done in a greedy way. The exception count of the decision list is simply the number of exceptions to the rulesets in the list. At any point, that ruleset is added to the decision list which minimizes the exception count of the resulting decision list. For efficiency's sake, an incremental algorithm is used that adds the ruleset which minimizes its contribution to the exception count. All covered examples and false negatives from the example list are removed after each ruleset is added, and succeeding rulesets are only checked on the remaining examples.

3.4 Continuous Attributes

There are no relevance bits maintained for continuous attributes. Instead, a pair of threshold boundary values is stored for each continuous attribute, which represents the allowable range of the attribute. Thresholds are computed prior to learning, using the information gain criterion used in the ID3 algorithm (Quinlan, 1979). In particular, all the training examples are sorted according to a continuous attribute. All split points that result in an information gain greater than some minimum threshold (*minGain*) are then collected. There is one set of thresholds for each continuous attribute, for each class. Each set has $-\infty$ as its first threshold, and $+\infty$ as its last. An interval such as $(-\infty,5)$ for an attribute x is shown as $x < 5$, $[6,10)$ is shown as $6 <= x < 10$, and $[14,\infty)$ is shown as $x >= 14$. A continuous attribute is irrelevant if its threshold boundaries are $\pm\infty$. When a new rule is created from a seed example and a continuous attribute has been decided to be relevant, the threshold boundary values are initialized to the narrowest interval containing the seed example's attribute value.

The values chosen for threshold values are those that begin a run of examples of the class that rules are being induced for, or a run of all other classes. For example, if we label examples of class 3, say, as '+' and all others as '-', then the sorted list of examples can be split into alternating segments of '+' and '-' examples. The attribute values considered as thresholds are those that separate the +ve examples from the –ve.

When doing a random adjustment to a continuous attribute, both threshold bounds are moved to random thresholds, making sure the seed example remains covered. A greedy adjustment to a continuous attribute is an adjustment to exactly one of the threshold boundaries that minimizes the description length of the resulting rule, while keeping the seed example covered. We chose to only consider adjustments to one threshold bound to speed the algorithm up.

3.5 Complexity Analysis

Here we derive the time complexity of the algorithm. A set of training data may be characterized in terms of three parameters: the number of attributes per example, the number of examples, and the number of classes. In order to derive a worst-case bound on running time, two assumptions will be made: (1) all attributes are continuous, and (2) only greedy adjustments are made. Table 2 contains all the variables used in the system. Of those, all except *minGain* and p_1 are used in the following derivation. M, k_1, r, *minGain*, and p_1 are tunable.

Finding the relative description length of a rule takes $O(ea)$ time since rule/example coverage tests take $O(a)$ time, and only one pass through the training data is needed. There are ta threshold adjustments to check when making a greedy step, hence each greedy step takes $O(tea^2) = O(e^2a^2)$ time, since $t \le e$. k_1a steps are taken to build each rule; therefore rule generation takes $O(Mk_1e^2a^3)$ time for each class.

Finding the description length of a ruleset takes $O(Mea)$ time. There are $\le M$ rule inclusions/exclusions to check, therefore one greedy step takes $O(M^2ea)$ time. Finally, there are $\le e + l = O(e + Ma)$ greedy steps taken, since the max description length is $e + l$ and it can only decrease during the course of the greedy walk. Hence, ruleset generation for one class for one restart takes $O(M^2e^2a + M^3ea^2)$ time. r restarts therefore take $O(rM^2e^2a + rM^3ea^2)$ time.

The final stage is class ordering. Recall that finding the description length of a ruleset takes $O(Mea)$ time. There are c candidates for the first ruleset in the decision list, then $c-1$, etc., for a total time complexity of $O(c^2Mea)$.

Overall, there are rule and ruleset generation phases for each class, plus a single phase of class ordering. This

Table 2

Param.	Description
a	number of attributes per example
e	number of training examples
c	number of classes
M	Number of rules added to the rule pool during rule generation
k_1	Multiplier of a that determines the length of the walk during rule generation
r	*Restarts* from the pseudocode: number of restarts during ruleset generation
t	the length of the longest threshold list
l	the total number of literals in all rules in the rule pool.
minGain	Minimum information gain required to keep a threshold boundary value.
p_1	Probability of making a random step during rule generation.

results in a final time complexity of $O(cMk_1e^2a^3 + crM^2e^2a + crM^3ea^2 + c^2Mea) = O(c^2rk_1M^3e^2a^3)$.

In our experiments, parameters were tuned by observing runs on a few of the datasets – initially on the synthetic datasets, and then when the algorithm worked on more than just binary attributes and classes, on a couple of the other datasets, such as iris and ecoli.

We used M=100 in these experiments, and if there are few duplicates, this results in a constant factor of 1,000,000 in the third term. The size of the ruleset decreases with each step, so making the greedy steps gets faster the farther along in the walk you are. Nevertheless, making those first steps can be very time consuming. In practice, the time for rule generation is often outweighed by the time for ruleset generation. In any case, ruleset ordering is very fast, and the actual time is usually negligible in comparison to the other two stages.

4. Results

We used 9 datasets from the UCI Machine Learning Databases Repository (Blake & Merz, 1998) and 2 synthetic datasets of our own creation (CNF and DNF) to compare our algorithm to RIPPER, C4.5 (Quinlan, 1993), and CN2. Seven other datasets from the archive are used to compare LERILS to G-NET and Smog.

4.1 C4.5, RIPPER, CN2

C4.5 is a popular machine learning algorithm that greedily builds a decision tree using information gain to select an attribute to divide the training set. It is simple, fast, and performs well on most domains. RIPPER and CN2 were described in section 2. RIPPER's default behavior is to first order the classes by frequency of occurrence in the training data. Assuming *n* classes, it finds rules to separate the first class from all others, then class 2 from class 3 through *n*, etc. No rules are induced for the last (most frequently occurring) class; it becomes the default. To resolve ordering conflicts RIPPER tries the rules in the order they were induced. When using the unordered rules option, rules are created to separate class *k* from all other classes, for all classes *k*. To resolve conflicts, the rule that was most accurate on the training data is preferred. If no rules cover the example in question, the most commonly occurring class in the training data is predicted.

CN2 has the same capability to create unordered and ordered rules (Clark & Boswell, 1991). When CN2 creates unordered rules, the algorithm changes a little bit. Instead of removing *all* covered examples after a new rule is found, only those examples of the new rule's class are removed. This promotes the accuracy of succeeding rules, as the negative examples still remain in the training data to help shape the new rules. The conflict resolution strategy for classification is changed as well. Instead of using the decision list structure, the preferred class for an unseen example is chosen probabilistically from the classes of all rules that cover the example.

The last column of table 4 contains LERILS's scores when making the last class in the ruleset list a default class. Note that both RIPPER and CN2, whether creating ordered or unordered rules, always have a default class. If they had no default class, their accuracies would drop. The appropriate comparisons to LERILS to make are with unordered variants of the algorithms. This is because the rules for the ordered variants are produced by removing all examples of previous classes before learning rules for the next class. Thus, the rules are not independently accurate. Scores for the ordered variants appear in the table as a reference, because the ordered variants are more commonly used. LERILS outperforms unordered RIPPER and CN2 on many of the datasets. Ordered systems are generally more accurate than unordered systems, but they lose comprehensibility.

LERILS significantly outperformed C4.5 on four datasets: CNF, DNF, xd6, and balance-scale. It was outperformed only on ecoli. LERILS outperformed unordered CN2 on CNF, DNF, and xd6, and was outperformed on lost2p-high and balance-scale. CN2's hypothesis for these datasets contained 6 and 8 times (respectively) the number of rules that LERILS did. Clearly, this extra accuracy came at a price. LERILS outperformed unordered RIPPER on only hepatitis and xd6, but was not significantly outperformed by RIPPER on any of the datasets.

Table 5 contains hypothesis sizes for a single run of the algorithms. These are indicative of comprehensibility, although comprehensibility involves more than the rule size and there is no widely accepted measure. For example, it appears that rules are more comprehensible than trees, and rules that are independent of class order are even better than order-dependent rules. For the rule learners there are two numbers per cell: the number of rules, followed by the total number of literals in all rules. For those algorithms that used a default class, the default rule counted as one rule with no literals. For C4.5, the values represent the total number of nodes in the decision tree.

LERILS always induced hypotheses that were smaller than unordered RIPPER's. The number of literals in LERILS's rules were always smaller than the number of nodes in C4.5's trees. In almost all cases, CN2's hypotheses are significantly larger than the rest.

The above results may look good, but our algorithm is much slower than CN2 and RIPPER. Each of the ten trials with the iris dataset took less than 2 minutes to complete. In contrast, each trial of the credit-screening dataset took 2.5 to 3.5 hours to complete. But our goal with this algorithm is to sacrifice speed for accuracy and comprehensibility. We took the view that it is acceptable to take longer to learn, as long as good, independent, comprehensible rules are produced.

4.2 G-Net and Smog

We compared the performance of LERILS with two other search-intensive learning algorithms, G-net and Smog, that also use a minimum description length heuristic to avoid overfitting. G-net (Anglano et. al., 1998) learns concepts in the form of propositional and first order Horn-clauses. It has a two-level architecture where the lower level uses a form of ("coevolutive") genetic search to find the best clauses, which are then combined into a disjunction at the higher level using a greedy search. Similar to LERILS, it uses evaluation functions that incorporate minimum description length principle at both the levels. While LERILS and G-net are similar in sharing the same hypothesis space and two-level search, they use very different search strategies in both the levels.

Smog (Oliveira & Sangiovanni-Vincentelli, 1996) represents its hypotheses in the form of reduced ordered decision graphs (RODGs), which have the advantage of being much more compact than decision trees and rules. It uses a highy successful heuristic technique from logic synthesis for reordering the variables in the graph, and applies a stochastic search algorithm with minimum description length heuristic to find a compact RODG. LERILS shares the ideas of stochastic search and MDL

Table 3. Descriptions of Datasets

Dataset	Attributes	Examples	Attribute Type	# of Classes	Description
CNF	26	1000	Binary	2	Synthetic 3-clause CNF concept
DNF	13	500	Binary	2	Synthetic 5-term DNF concept
Xd6	10	200	Binary	2	DNF concept, 10% noise
Lost2p-high	24	928	Binary	2	Chess domain
Balance-scale	4	625	Nominal	3	Do objects balance each other?
Balloons	4	20	Binary	2	Conditions of an experiment
Iris	4	150	Continuous	3	Identify species of Iris
Ecoli	7	336	Continuous	8	Localization sites of proteins
Hepatitis	19	155	13 bin, 6 cont	2	Predict if a patient dies or lives
Credit-	15	690	4bin,5nom,6cont	2	Credit approval
Glass	9	214	Continuous	7	Predict type of glass

Table 4. Accuracies of the Algorithms (* indicates a performance increase relative to LERILS significant at the 0.05 level. † indicates a performance decrease relative to LERILS significant at the 0.05 level.)

Dataset	RIPPER (unordered)	RIPPER (ordered)	C4.5	CN2 (unordered)	CN2 (ordered)	LERILS
CNF	99.6 ± 0.27	100 ± 0	97.3 ± 0.40†	98.9 ± 0.35†	100 ± 0	100 ± 0
DNF	99.8 ± 0.20	94.4 ± 0.93	96.8 ± 0.90†	97.8 ± 0.63†	98.2 ± 0.92	99.4 ± 0.43
xd6	79.5 ± 2.93†	84.5 ± 3.20	74.5 ± 3.37†	77.5 ± 2.71†	77.0 ± 1.86	87.5 ± 2.50
Lost2p-high	97.1 ± 0.64	96.7 ± 0.48	96.9 ± 0.65	98.7 ± 0.27*	98.5 ± 0.37	97.4 ± 0.56
Balance-scale	71.5 ± 1.49	72.1 ± 2.07	60.6 ± 1.97†	79.4 ± 1.53*	87.7 ± 1.12	73.1 ± 2.11
Balloons	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0
Iris	94.0 ± 2.32	93.3 ± 2.22	94.5 ± 1.72	95.3 ± 1.42	93.3 ± 1.72	96.7 ± 1.49
Ecoli	81.8 ± 1.50	81.2 ± 2.06	82.8 ± 1.26*	79.7 ± 2.22	80.6 ± 2.37	78.0 ± 1.77
Hepatitis	77.5 ± 3.16†	80.0 ± 3.85	81.2 ± 3.09	79.9 ± 4.41	81.8 ± 3.81	85.3 ± 3.15
Credit-screening	87.5 ± 1.50	86.5 ± 1.79	85.4 ± 1.58	82.6 ± 1.54	80.9 ± 0.80	84.5 ± 1.61
Glass	58.4 ± 2.11	69.0 ± 2.15	68.9 ± 3.33	64.5 ± 2.81	71.2 ± 3.08	60.8 ± 3.31

Table 5. Hypothesis Size (rules/literals)

Dataset	RIPPER (unordered)	RIPPER (ordered)	C4.5	CN2 (unordered)	CN2 (ordered)	LERILS
CNF	23/71	4/11	45	33/120	18/44	4/11
DNF	16/52	11/38	41	22/81	12/24	6/14
xd6	7/14	6/17	17	39/174	32/87	4/9
lost2p-high	13/35	4/19	23	25/72	14/32	4/20
Balance-scale	27/53	15/31	26	148/474	56/102	18/36
Balloons	4/4	2/2	5	4/4	3/2	2/2
Iris	6/7	3/3	7	10/16	8/11	3/3
Ecoli	15/36	9/16	33	24/61	19/48	7/11
Hepatitis	6/8	4/6	13	18/47	12/36	4/7
Credit-screening	8/22	6/16	29	31/99	37/104	6/12
Glass	11/19	8/14	39	26/70	18/43	5/9

Table 6. Comparison to G-NET and Smog

Dataset	G-NET	Smog	C4.5	LERILS
Monk1	100 ± 0	100 ± 0	100 ± 0	100 ± 0
Monk2	97.2 ± 3.8*	100 ± 0*	67.2 ± 10.66	77.4 ± 0.97
Monk3	100 ± 0*	100 ± 0*	100 ± 0*	98.9 ± 0.40
Tictactoe	99.0 ± 0.62	97.2 ± 1.97	92.9 ± 1.82†	100 ± 0
Credit	84.2 ± 4.40	80.4 ± 5.08	86.0 ± 3.28	84.5 ± 1.61
Breast	94.7 ± 2.89	93.3 ± 2.44	94.1 ± 3.32	95.9 ± 0.65
Vote	94.9 ± 3.20	94.7 ± 2.64	95.4 ± 3.05	94.5 ± 0.86

Table 7. Comparison to BruteDL

Dataset	BruteDL	LERILS
Glass	62.0 ± 5.3	60.8 ± 3.3
Hepatitis	80.6 ± 7.9	85.3 ± 3.2
Iris	93.1 ± 4.5	96.7 ± 1.5
Vote	93.0 ± 3.2	94.5 ± 0.86

heuristic with Smog, although they use very different hypothesis spaces.

Table 6 includes comparisons to Smog and G-NET on 7 more datasets. The results for G-NET, Smog, and C4.5 are taken from (Anglano et al., 1998). Both G-Net and Smog beat LERILS on monk2 and monk3. Monk2's concept requires 15 rules of 2 literals each (exactly 2 of 6 attributes are true) to express exactly using our disjunctive hypothesis space; LERILS pruned away a few rules, resulting in decreased performance. Elsewhere, scores were mostly similar.

4.3 BruteDL

As Table 7 shows, the accuracies of the rules of BruteDL are similar to those of LERILS in domains where both of them are tested. It is not clear, however, how large the learned rules are, or how many of them there were in the learned hypotheses. Segal and Etzioni (1994) prove that a non-homogeneous ruleset can always be converted to a homogenous one. Unfortunately, this might require an exponential growth in the number of rules, as the rules themselves become progressively more specialized. Also, homogeneity is not the same as accuracy. If the accuracy of a rule would decrease for any specialization, then the rule is not homogeneous, and is therefore unusable by BruteDL, even if it is highly accurate and simple. There may be a set of equivalent homogeneous rules, but depth-limiting the search may prevent them from being found. With no depth limits, there may be too many rules of low coverage, and choosing some or all of them could overfit. In any case, the set of equivalent homogeneous rules is more complex than the original rule.

Thus, BruteDL sacrifices accuracy and simplicity for homogeneity. LERILS, on the other hand, does not care for homogenity, and trades off accuracy and rule size. It induces all its rules without example removal, which ensures that rules are independently accurate. Rule overlap is discouraged by insisting that each rule covers a new seed example. The disadvantages of LERILS are its many tunable parameters and its expensive search. BruteDL is a cleaner and faster algorithm.

5. Conclusions and Further Work

We have demonstrated an algorithm that induces human-comprehensible rules with accuracies comparable to the best rule inducers. It can handle datasets with any number of classes, and both missing and continuous-valued attributes. The last five datasets in table 3 have one or more continuous attributes. Unordered RIPPER outperformed LERILS on three of them, so there still seems to be room for improvement in the handling of continuous attributes. Any improvements made to the speed of the algorithm would also greatly increase its attractiveness. For example, these include making the ruleset generation process more incremental, and preprocessing training data by taking a subset of features and/or training data. Nevertheless, our accuracies seem to be comparable if not significantly better than the others. We have made a greater commitment to comprehensibility and rule order-independence than the others, and that's where our algorithm shines.

Due to the modularity of the algorithm, it is possible to add a "filtering module" in between the rule generator and the ruleset generator, which could look at the rules generated by the rule generator and throw out some rules before passing them on to the ruleset generator. This filtering could be done in different ways. Humans may have an idea of what rules make sense, and what rules don't. The rules in the rule pool could be shown to a human expert who selects rules to throw away. In this way, a human could become involved in the rule learning. Or, a knowledge based solution might be used to replace the human. This would allow prior domain-specific knowledge to assist in rule induction, which has historically been difficult to do.

BruteDL tries to minimize rule overlap by finding homogenous rules. However it can be argued that rule overlap is a good thing because it makes the ruleset more robust against noise and missing attributes. On the other hand, the time complexity of LERILS may be improved by using homogeneity as a heuristic in addition to description length. It appears that accuracy, comprehensibility, and robustness are three desirable properties of a rule set, and one might have to trade off one for the other. We took comprehensibility to mean simplicity and order-independence, but this is admittedly inadequate. More research is needed to formalize the notions of comprehensibility and robustness, and design learning algorithms that try to achieve them.

Acknowledgements

We would like to thank Ross Quinlan, William Cohen, Peter Clark, and Tim Niblett for making their rule induction systems available for use in our comparisons. We thank the anonymous reviewers of this paper for many excellent suggestions and pointers to previous work. This work was partially supported by NSF under grant number IIS-0098050.

References

Anglano, C. Giordana, A., Lo Bello, G. & Saitta, L. (1998) "An Experimental Evaluation of Coevaluative Concept Learning" in *Proceedings of the Fifteenth International Conference on Machine Learning*, 19-27, Morgan Kaufmann.

Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford, England: Oxford University Press.

Blake, C. L. & Merz, C. J. (1998). *UCI Repository of Machine Learning Databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Computer and Information Science.

Clark, P., & Niblett, R. (1989). "The CN2 Induction Algorithm." *Machine Learning*, 3, 261-284.

Clark, P., & Boswell, R. (1991). "Rule Induction with CN2: Some Recent Improvements." In *Proceedings of the Fifth European Conference on Machine Learning*, editor Yves Kodratoff, pp. 151-163. Springer-Verlag, Berlin.

Cohen, W. W. (1995). "Fast Effective Rule Induction." In *Proceedings of the 12th International Conference on MachineLearning,* pp. 115-123. Morgan Kaufmann.

Cohen, W. W., & Singer, Y. (1999). "A Simple, Fast, and Effective Rule Learner." *AAAI-99*, Orlando, Florida.

Mitchell, T. (1997). *Machine Learning*, McGraw Hill.

Quinlan, J. R. (1979). "Discovering Rules From Large Collections of Examples: A Case Study." In Michie, D., editor, *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California.

Quinlan, J. R. (1999). "Some Elements of Machine Learning." In *Proceedings of the 16th International Conference on Machine Learning*, Extended Abstract pp. 523-525

Rissanen, J. (1984). "Universal Coding, Information, Prediction, and Estimation." *IEEE Transactions on Information Theory*, IT-30(4):629-636.

Schapire, R. E., & Singer, Y. (1998). "Improved Boosting Algorithms using Confidence-rated predictions." In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 80-91.

Segal, R., & Etzioni, O. (1994). "Learning Decision Lists using Homogeneous Rules." *AAAI-94*.

Selman, B., Levesque, H. J., & Mitchell, D. G. (1992). "A New Method for Solving Hard Satisfiability Problems." *Proc. AAAI-92*, San Jose, CA, 440-446.

Selman, B. & Kautz, H., & Cohen, B. (1994). "Noise Strategies for Local Search." In *Proc AAAI-94*, Seattle, WA, 337-343.

Selman, B. & Kautz, H. (1996). "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search." *AAAI-96*, Portland, OR.

Oliveira, A. L. & Sangiovanni-Vincentelli, A. (1996) "Using minimum description length principle to infer reduced ordered decision graphs" in *Machine Learning*, 23--50, 25(1).

Weiss, S., Galen, R., & Tadepalli, P. (1990). "Maximizing the Predictive Value of Production Rules." *Artificial Intelligence.* 45, pp. 47-71.

Weiss, S. & Indurkhya, N. (1991). "Reduced Complexity Rule Induction." *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, California, pp. 678-684.

Weiss, S. & Indurkhya, N. (2000). "Lightweight Rule Induction" in *Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, pp 1135--1142