Complementary Discrimination Learning with Decision Lists

Wei-Min Shen
Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759
wshen@mcc.com

Abstract

This paper describes the integration of a learning mechanism called complementary discrimination learning with a knowledge representation schema called decision lists. There are two main results of such an integration. One is an efficient representation for complementary concepts that is crucial for complementary discrimination style learning. The other is the first behaviorally incremental algorithm, called CDL2, for learning decision lists. Theoretical analysis and experiments in several domains have shown that CDL2 is more efficient than many existing symbolic or neural network learning algorithms, and can learn multiple concepts from noisy and inconsistent data.

Introduction

Complementary discrimination learning (CDL) (Shen 1990) is a general learning mechanism inspired by Piaget's child development theories. The key idea is to learn the boundary between a hypothesis concept and its complement incrementally based on the feedback from predictions. The framework is general enough to be applied to many learning tasks (Shen 1989, Shen 1992), including concept learning, adaptive control, learning finite state machines, and discovering new theoretical terms. However, earlier implementations of CDL use standard CNF and DNF Boolean formulas to represent separately the hypothesis concept and its complement. Since hypotheses are revised frequently in the learning process, to keep them complementary in these canonical forms is computationally expensive.

A decision list is a representation schema proposed by Rivest (1987) for Boolean formulas. He proves that k-DL (decision lists with functions that have at most k terms) are more expressive than k-CNF, k-DNF, and Decision Trees. He also gives a polynomial-time algorithm for learning decision lists from instances. However, one of the open problems listed in his paper is that there is no known incremental algorithm for learning decision lists.

Complementary discrimination learning and decision lists are well suited for each other. On the one hand, decision lists eliminate the burden of maintaining separate representations for complements. (Since decision lists are closed under complementation, they can be used to represent a concept and its complement in a single format.) On the other hand, complementary discrimination provides a way to learn decision lists incrementally.

The integration of complementary discrimination learning and decision lists gives some surprising results. The most salient one is efficiency. In the experiments we conducted, CDL2 is much more efficient than most other behaviorally incremental algorithms[1], such as ID5r, and its performance is even competitive with the nonincremental learning algorithm ID3. Compared to nonsymbolic algorithms such as backpropagation neural networks, CDL2 is much faster to train and has comparable results in recognizing hand-written numerals that are noisy and inconsistent. CDL2 also extends Rivest's definition of decision lists from binary concepts to multiple concepts. Such decision lists provide a compact representation for concepts. When applied to Quinlan's (1983) chess task, CDL2 has learned a decision list of 36 decisions.

The rest of the paper is organized as follows. Section 2 gives a general description of complementary discrimination and points out the need for decision lists. Section 3 describes in detail the new learning algorithm CDL2 and how decision lists are used in complementary discrimination style learning. Section 4 reports experimental results of CDL2 in several concept learning tasks and compares the results with other learning algorithms. Section 5 analyzes the complexity of CDL2.

Complementary Discrimination and Decision Lists

Complementary Discrimination Learning (CDL) is a mechanism for learning concepts from instances. In some sense, instances can be viewed as points defined

[1] A learning algorithm is behaviorally incremental if it is incremental in behavior (i.e., it can process examples one at a time), but not incremental in storage and processing costs (e.g., it may require to remember some or all previous examples.)

in a feature space and concepts as regions in such a space. The main idea of CDL is to move a hypothesized boundary between a hypothesis concept H and its complement $\overline{H}$ in the feature space until it converges to the real boundary of the real concept and its complement. The boundary between H and $\overline{H}$ is adjusted based on feedback from predictions. When a new instance arrives, CDL predicts that the instance belongs to one of the hypothesized complements, either H or $\overline{H}$, depending on which side of the boundary the instance lies on. If the prediction is correct, CDL remembers the new instance as an *example* of the hypothesis. If the prediction is wrong, then the boundary is "shifted" towards the hypothesis in the prediction. This is done by shrinking one hypothesis and expanding the other. How much the hypothesis is shrunk depends on the *difference* between the new instance and all the previous examples of that hypothesis. Technically, the shrinking is done by conjoining the hypothesis with the difference so found. After the hypothesis is shrunk, the other hypothesis is expanded to be the complement of the shrunken hypothesis. The new instance is remembered as an example of the expanded (complement) hypothesis.

Efficiency of this CDL scheme depends on its implementation. If both H and $\overline{H}$ are separately represented, then one must make sure that H and $\overline{H}$ are always complementary each time the boundary is moved. One trick is to keep them in some canonical forms, say one in CNF and the other in DNF (Shen 1990). However, since the differences between a new instance and previous examples are always in DNF, it is computational expensive to keep the concepts in these canonical forms. Moreover, this implementation is awkward for multiple concepts. Since there is only one boundary to keep, the names of multiple concepts must be represented as artificial "features" of the instances so that CDL can learn them as if they were binary concepts.

Thus, in order to use the idea of CDL efficiently, a better representation for complementary concepts must be found. Such a representation should make maintenance of complementarity between H and $\overline{H}$ computationally inexpensive. Decision lists seem to be an ideal choice.

Following Rivest, a *decision list* is a list L of pairs

$$(f_1, v_1), \ldots, (f_r, v_r),$$

where each test function f_j is a conjunction of literals, each v_j is a value in {0,1}, and the last function f_r is the constant function *true*. For convenience, a pair (f_j, v_j) is called the j-th *decision*, f_j the j-th *decision test*, and v_j the j-th *decision value*. In this paper, we use an extended definition of decision lists in which v_j is a value from a finite set of concept names rather than from the binary set {0,1}.

A decision list L defines a set of concepts as follows: for any instance x, the decision on x is defined to be $D_j = (f_j, v_j)$, where j is the least index such that $f_j(x) = 1$. The value v_j is the concept of x.

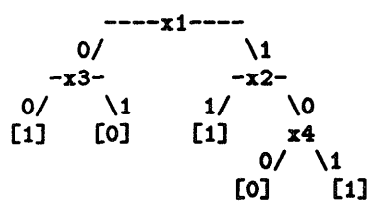

Figure 1: A decision tree

To see how decision lists represent concepts, consider the decision tree in Figure 1. The concept defined by this tree is equivalent to a Boolean formula in DNF:

$$\overline{x}_1\overline{x}_3 \vee x_1x_2 \vee x_1\overline{x}_2x_4$$

This can be easily translated into a decision list:

$$(\overline{x}_1\overline{x}_3, 1)(x_1x_2, 1)(x_1\overline{x}_2x_4, 1)(\mathit{true}, 0) \qquad (1)$$

Since a decision list can have values other than 1s, the same concept can also be represented as the following different decision lists:

$$(\overline{x}_1\overline{x}_3, 1)(\overline{x}_1x_3, 0)(\overline{x}_2\overline{x}_4, 0)(\mathit{true}, 1) \qquad (2)$$

$$(\overline{x}_1x_3, 0)(x_2, 1)(x_1\overline{x}_4, 0)(\mathit{true}, 1) \qquad (3)$$

Decision lists make the movement of concept boundaries easier. We let $|D|$ denote the domain of a decision $D = (f, v)$, which is a set of instances $\{i | f(i) = 1\}$. An interesting property of decision lists is that decisions with smaller indices "block" decisions with larger indices if their domains overlap. To discriminate a decision, one can simply "shrink" or "break" the domain of the decision so that instances that do not belong to this decision can "fall through" to later decisions that have the correct values.

Consider, for example, a decision list ($(\overline{x}_3$,1)(true, 0)). If evidence has shown that ($\overline{x}_3$,1)'s domain is too large, i.e., some part of that domain, say those that have $\overline{x}_1$, should have concept value 0 instead of 1, then one can simply append $\overline{x}_1$ to $\overline{x}_3$ to "shrink" the decision ($\overline{x}_3$,1) and get a new decision list: (($\overline{x}_1\overline{x}_3$,1)(true,0)). This is actually the key concept of the new complementary discrimination learning algorithm CDL2.

The CDL2 Algorithm

As we indicated in the last section, complementary discrimination is a mechanism that learns the boundary between a concept and its complement. The necessary subtasks of general CDL are: to detect which hypothesis is overly general; to find out what part of the hypothesis is overly general; and to actually move the boundary towards that hypothesis.

In the case of learning decision lists, the first task is accomplished as follows: when a new instance x (with a concept value v_x) arrives, the learner uses the current decision list to make a prediction as to what concept the new instance belongs to. The prediction is the value v_j,

```
Let x be the new instance and v_x be its concept value.
Loop: Let D_j = (f_j, v_j) be the decision on x;
  If the decision is correct, i.e., v_x = v_j,
  then store x as an example of D_j and return;
  else
    Let (σ_1, ···, σ_d) = DIFFERENCES(examplesOf(D_j), x);
    Replace (f_j, v_j) by (f_j ∧ σ_1, v_j), ···, (f_j ∧ σ_d, v_j);
    Distribute the examples of D_j to the new decisions;
    If D_j was the last decision,
    then append (true, v_x) at the end of the list, and return.
```

Figure 2: The basic CDL2 learning algorithm

where j is the least index such that $f_j(x) = 1$. If v_j is not equal to v_x, then the decision (f_j, v_j) is too general.

Since each decision in the decision list is associated with a list of previous examples that belong to the decision, to find out what part of f_j is overly general entails finding the difference between the new instance and all of the previous examples that belong to f_j. The difference will be a list of terms that are true for previous examples but false for the new instance. For example, if the new instance is 10001 (in the form of $x_1x_2x_3x_4x_5$) and the previous examples are 00011, 00010, 00001, 00000, 11000, 11001, 11010, and 11011, then the difference will be $(\overline{x}_1, x_2)$, where $\overline{x}_1$ distinguishes (00011, 00010, 00001, 00000) from 10001 and x_2 distinguishes (11000, 11001, 11010, 11011) from 10001. For a detailed description of how differences are found, see (Shen 1990).

Let the differences so found be a list $\{\sigma_1, \cdots, \sigma_d\}$. To shrink or break the decision (f_j, v_j), we replace it with a list of new decisions $(f_j \wedge \sigma_1, v_j), \cdots, (f_j \wedge \sigma_d, v_j)$. Clearly, none of the new decisions will capture the new instance again. The previous examples of the old decision are then distributed to these new decisions in the following manner: an example e is distributed to $f_j \wedge \sigma_i$ where i is the least index such that $[f_j \wedge \sigma_i](e) = 1$.

After the incorrect decision is replaced, the new instance continues to look for a decision in the remainder of the old decision list. Suppose the new prediction is from the decision D_k, where $k \geq (j + d)$. If $v_k = v_x$, then the instance is remembered as an example of D_k. Otherwise, D_k is shrunken or broken just as D_j was. This process continues until the instance finds a decision with the correct value. If the instance reaches the end of the list, i.e., $D_k = (true, v_k)$, then either $v_k = v_x$ and x can be added to D_k's example list, or D_k is shrunk and a new "default" decision $(true, v_x)$ is appended at the end of the list with x as its only example. The pseudocode of the CDL2 algorithm is listed in Figure 2.

It is interesting to notice that Rivest's nonincremental learning algorithm constructs decision lists from the beginning to the end, while CDL2 constructs decision lists from the end to the beginning (roughly). This is because CDL style learning is based on discrimination, and decisions with greater indices are more general than those with smaller indices. The use of discrimination also sets apart CDL2 from "exemplar-based learning" mechanisms, e.g., (Salzberg 1991). The main idea of those algorithms is to "grow" a region around some seed instances, while the idea of CDL2 is to "repartition" regions based on surprising instances. Furthermore, the decision tests learned by CDL2 are very similar to the features used at the nodes of decision trees (Pagallo and Haussler 1990).

To illustrate the algorithm, let us go through an example to see how CDL2 learns the concept defined Figure 1: $\overline{x}_1\overline{x}_3 \vee x_1x_2 \vee x_1\overline{x}_2x_4$. We assume that the training instances are from 00000 to 11111 and will be given in that order. Each training instance has a concept value. For example, (11000 +) means the instance 11000 is in the concept, while (00111 −) means 00111 is not in the concept.

When the first training instance (00000 +) arrives, CDL2 initiates a decision list with a single "default" decision:

$$((true, +)),$$

The instance is stored as an example of this sole decision. Since the next three instances 00001, 00010, and 00011 are all being predicted correctly, they also become examples of the default decision. The fifth instance is (00100 −), and CDL2's prediction "+" is wrong. The difference between (00000 00001 00010 00011) and 00100 is found to be $(\overline{x}_3)$, the decision is shrunk to be $(\overline{x}_3, +)$, and a new default $(true, -)$ (with the instance 00100 as its example) is appended at the end. The new decision list is now:

$$((\overline{x}_3, +)(true, -)).$$

With this decision list, the succeeding instances are predicted correctly until 10000. CDL2's decision is $(\overline{x}_3, +)$ but the instance belongs to "−." Comparing the examples of the decision with the new instance, CDL2 finds the difference to be $\overline{x}_1$. The decision is then replaced by $(\overline{x}_1\overline{x}_3, +)$:

$$((\overline{x}_1\overline{x}_3, +)(true, -)).$$

The troublesome instance then finds $(true, -)$ to be correct and becomes an example of the default decision. For succeeding instances, CDL2's prediction is correct on 10001, but wrong on 10010. The decision is $(true, -)$ but 10010 is a +. The differences found are $(x_3\overline{x}_1, \overline{x}_4)$, so the decision $(true, -)$ is replaced by $((x_3\overline{x}_1, -)(\overline{x}_4, -)(true, +))$, yielding:

$$((\overline{x}_1\overline{x}_3, +)(x_3\overline{x}_1, -)(\overline{x}_4, -)(true, +)).$$

With this decision list, CDL2 predicts correctly the next five instances 10011, 10100, 10101, 10110, and 10111 but fails on 11000. The decision is $(\overline{x}_4, -)$ but the instance is a +. The difference between $\overline{x}_4$'s examples (10101 10100 10001 10000) and the instance 11000 is found to be $(\overline{x}_2)$. The decision $(\overline{x}_4, -)$ is then shrunk into $(\overline{x}_4\overline{x}_2, -)$ and the new decision list is now:

$$((\overline{x}_1\overline{x}_3, +)(x_3\overline{x}_1, -)(\overline{x}_4\overline{x}_2, -)(true, +))$$

This decision list is equivalent to the target concept, and it correctly predicts 11001 through 11111.

The basic CDL2 algorithm can be improved in several ways. The first one is to construct decision lists that are shorter in length. CDL2 does not guarantee learning the shortest decision list, and the length of the final decision list depends on the order of the training instances. If the instances that are more representative (i.e., that represent the critical differences between complementary concepts) appear earlier, then the length of the decision list will be shorter. Although learning the shortest decision list is a NP-complete problem (Rivest 1987), there are some strategies to aid in maintaining the list as short as possible.

Every time a decision is replaced by a list of new decisions, we can check to see if any of these new decisions can be *merged* with any of the decisions with greater indices. A merger of two decisions (f_i, v) and (f_j, v) is defined to be (f_m, v), where f_m is the intersection of the literals of f_i and f_j. Two decisions $D_i = (f_i, v_i)$ and $D_j = (f_j, v_j)$ $(i < j)$ can be merged if the following conditions are met: (1) The two decisions have the same value, i.e., $v_i = v_j$; (2) None of the examples of D_i is captured by any decisions between i and j that have different decision values; (3) The merged decision (f_m, v_j) does not block any examples of any decisions after j that have different values.

To illustrate the idea, consider the following example. Suppose the current decision list is $((\overline{x}_3, +)(x_1, -)(true, +))$, and examples of these three decisions are (010, 000), (111, 101), and (011), respectively. Suppose the current instance is (100 −), for which CDL2 has made the wrong decision $(\overline{x}_3,+)$. Since the difference between (010, 000) (the examples of D_1) and 100 is $(\overline{x}_1)$, the decision $(\overline{x}_3,+)$ should be replaced by $(\overline{x}_3\overline{x}_1,+)$, which would result in a new decision list $((\overline{x}_3\overline{x}_1,+)(x_1,-)(true,+))$. However, the new decision $(\overline{x}_3\overline{x}_1,+)$ can be merged with (true,+) because it has the same decision value, and none of its examples can be captured by $(x_1,-)$, and the merged decision, which is (true,+), does not block any other decisions following it. Thus, the decision list is shortened to: $((x_1,-)(true,+))$.

The second improvement over the basic CDL2 algorithm is to deal with inconsistent data. When data are inconsistent, there will be no difference between some instances that have different concepts values. In other words, the same instance may belong to different concepts simultaneously. To handle such data, we relax the criterion for a correct decision to be:

> A decision $D_j = (f_j, v_j)$ is *correct* on an instance x that has concept value v_x if either $v_j = v_x$, or x is already an example of D_j.

For example, suppose a decision $(x_1x_2,+)$ currently has the example ((11 +)) and a new instance (11 −) arrives, then the decision $(x_1x_2,+)$ will be considered to be correct because 11 is already in its example set. With this new criterion, a decision may have duplicate examples. Examples that belong to the same decision may have the same instance with different concept values, and the value of the decision may be inconsistent with some of its examples. To deal with this problem, we have to adopt a policy that the value of a decision is always the same as the concept value that is supported by the most examples. If another example (11 −) is by the decision above, then the value of the decision will be changed from "+" to "−" because "−" will be supported by two examples vs. only one example for "+".

Table 1: Comparison on classifying chess end games

Program	Passes	CPU sec.	Acc.(%)	#decs.
ID3	N.A.	15.7	100	78
ID5$\hat{r}$	1	408.6	88.92	65
	8	1420.5	100	78
ID5r	1	2277.7	100	85
CDL2	1	11.4	100	36

Experiments

To date, the improved CDL2 has been tested in four domains: 6-bit Multiplexor (Barto 1985, Utgoff 1989), Quinlan's chess task (Quinlan 1983), the noisy LED display data (Breiman *et al.* 1984), and a hand-written character recognition task. The first two domains are noise free, and the last two domains are noisy and contain inconsistent instances. In this section, I report CDL2's performance only on the two more difficult tasks: the chess task and the recognition of handwritten numerals. Comparison with other learning algorithms will also be given.

In Quinlan's chess task, the instances are configurations of end games in chess, each of which is represented as a binary feature vector of length 39. Each instance is labeled either "win" or "lose," and the task is to learn the concept of "win" (or "lose") from a given set of 551 end games. I have compared CDL2 with two leading algorithms ID3 and ID5r on this domain. The results are listed in Table 1.

The first column in the table is the name of the algorithm. ID3 (Quinlan 1986) is a nonincremental algorithm that learns decision trees. ID5$\hat{r}$ and ID5r (Utgoff 1989) are incremental algorithms that learn decision trees. The difference between ID5$\hat{r}$ and ID5r is that when the decision tree is revised, ID5r ensures that all the previous instances are still correctly classified, while ID5$\hat{r}$ does not.

The second column is the number of passes that each algorithm made on the data set. This number is not meaningful for the nonincremental ID3 because it requires to see all the instances at once. The rest of the columns are the running times, the percentage of instances that are correctly classified after learning, and the smallest size for the learned concept, respectively. All algorithms are implemented in Allegro Common Lisp on Sun4, and the running time is an average for runs in which the instances are presented in different orders. As we can see, with the same accuracy, CDL2

Table 2: Comparison on the numeral recognition task

Prog. Names	Epc.	CPU (min)	Acc.% (train)	Acc.% (test)	(n)odes (w)ghts
Linked	30	300	96.6	87.8	385n
Local	45	450	96.9	87.8	2860w
NNet	50	500	96.9	88.2	
ID3	N.A.	4.4	99.9	74.2	440dec
ID5$\hat{r}$	—	—	—	—	—
CDL2	1	13.1	99.9	77.9	360dec

is about 200 times faster than ID5r and is even faster than the nonincremental algorithm ID3. The size of concept learned by CDL2 is also very competitive with others. (The numbers in the last column are just rough indications. A size comparison between decision trees and decision lists involves a very lengthy discussion.)

The second set of experiments performed is in the domain of recognizing hand-written numerals. The instances are 3301 numerals (from 0 to 9) written by humans, each of which is represented as an 8x8 binary bit map. The data are noisy and have inconsistent instances (i.e., two bit maps that look the same may represent different numerals). The task is to learn to recognize these numerals by training on some numerals (65% of the data), then testing the recognition on numerals that have not been seen before (35% of the data). Note that a random guess in this domain has only 10% chance of being correct.

CDL2 is compared to some existing backpropagation neural networks and ID3. (This task is too complex for ID5$\hat{r}$ family algorithms.) The results are summarized in Table 2. Compared to ID3, CDL2 is slightly slower but has higher accuracy on the test data. Compared to the NNet, CDL2 learns faster and has higher accuracy on the training data, but it does not perform as well on the test data. One reason behind this is that the concepts in this domain are a set of arbitrary shapes (which can be handled by the neural net), while CDL2's current generalization is best suited for hyperrectangles in a high-dimension space. Another reason is that CDL2 is overfitting. Yet another reason is that CDL2, unlike these networks, has no knowledge about this domain. If such knowledge were embedded in the algorithm, just as in the choice of architecture for the networks, CDL2 might improve its performance.

Complexity Analysis

In this section, we analyze the complexity of the basic CDL2 algorithm. Let d be the number of attributes of instances, and let n be the number of training instances. We assume all the attributes are binary and analyze the complexity under the following three cases.

In the first case, which is extreme, we assume that every training instance will cause a prediction failure for CDL2, and the length of the decision list is always the longest possible (the number of decisions in the list is equal to the number of instances seen so far). When the i-th instance arrives, the length of the decision list is $i-1$ and each decision has only one example. Therefore, finding a decision for the instance takes at most $d(i-1)$ bit comparisons, finding the difference takes at most d comparisons, and replacing the decision takes 1 deletion and 1 insertion. Since we assume each decision can only capture one instance, the decision must be at the end of the list and the new instance will find a correct decision after one loop. Thus the total number of comparisons to process the i-th instance is $O(d \cdot i)$. There are n training instances, so the total number of comparisons to process all the training instances is:

$$\sum_{i=1}^{n} O(d \cdot i) = O(d \cdot n^2).$$

In the second case, which is also extreme, we assume that there is only one decision and it has all the previous $(i-1)$ examples. In this case, finding a decision takes d comparison, finding the differences takes $d(i-1)$ comparisons, replacing the decision takes 1 deletion and at most $(i-1)$ insertions, distributing examples takes at most $(i-1)^2 d$ comparisons. This "explosion" of decisions can only happen once and after that the analysis is the same as in the first case. To be conservative, we assume that the explosion happens at the n-th instance, so the total complexity is:

$$d \cdot n^2 + \sum_{i=1}^{n} O(d \cdot i) = O(d \cdot n^2).$$

In the third case, we assume the number of decisions is $\sqrt{i}$ and each of them has $\sqrt{i}$ examples. Then finding a decision takes $d\sqrt{i}$ comparisons, finding the difference takes $d\sqrt{i}$ comparisons, replacing decisions takes 1 deletion and at most $\sqrt{i}$ insertions, and distributing examples takes at most $d\sqrt{i}\sqrt{i}$ comparisons. We assume conservatively that in the worst case all the $\sqrt{i}$ decisions are broken (i.e., CDL2 loops $\sqrt{i}$ times for the i-th instance), and the total time for processing all n instances is:

$$\sum_{i=1}^{n} O(d \cdot i^{\frac{3}{2}}).$$

The above analysis is for the basic CDL2 algorithm. In the improved version, new decisions may be merged with the existing ones. Suppose there are $\sqrt{i}$ decisions in the list, a merge may take $d\sqrt{i}\sqrt{i}$ comparisons (condition 1 takes at most $\sqrt{i}$, conditions 2 and 3 together take at most $d\sqrt{i}\sqrt{i}$). There are at most $\sqrt{i}$ new decisions, so a revision of decision may require $d \cdot i\sqrt{i}$ comparisons. We assume conservatively that all the $\sqrt{i}$ decisions are broken, and the total time becomes:

$$\sum_{i=1}^{n} O(d \cdot i^2) = O(d \cdot n(n+1)(2n+1)) = O(d \cdot n^3).$$

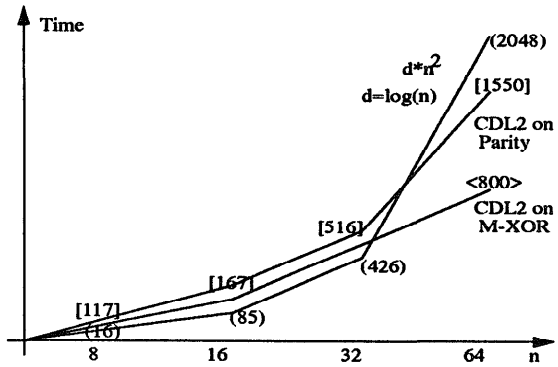

Figure 3: Comparison of CDL2 experiments with $d \cdot n^2$

Intuitively, it seems possible to tighten the above analysis considerably because CDL2's performance in practice is much better. To gain more information about the real complexity, we compared CDL2's actual running time with the complexity $d \cdot n^2$. The result is in Figure 3. In these experiments, we ran the improved CDL2, whose complexity is $O(d \cdot n^3)$ in theory, on two tasks: learning the d-bit parity concept and learning the d-bit multipleXOR. The first task is known to be the worst case for learning decision trees (Utgoff 1989) and the length of its decision list is n (corresponding to the first case in our analysis above). The second task has concepts that have exceptional instances and the length of its decision list is roughly $\sqrt{n}$ (corresponding to our analysis in the third case). In all these experiments, CDL2 is presented with all 2^d instances.

As we can see in Figure 3, CDL2's complexity is less than $d \cdot n^2$ in both tasks. These results suggest that our complexity analysis may indeed be too conservative. Nevertheless, the current complexity of CDL2 is at least comparable with that of ID3 and ID5r. According to (Utgoff 1989), ID3 takes $O(n \cdot d^2)$ additions and $O(2^d)$ multiplications, and ID5r takes $O(n \cdot d \cdot 2^d)$ additions and $O(2^d)$ multiplications. CDL2 uses only comparison operations and its complexity has no exponential components.

Conclusions and Acknowledgment

In this paper, I have described a successful integration of complementary discrimination and decision lists. Compared to earlier complementary discrimination learning algorithms, CDL2 has provided a solution to the problem of representing complementary concepts (or the boundaries of concepts) efficiently. It also provides a natural way to represent and to learn multiple concepts simultaneously. Compared to Rivest's initial results on decision lists, this work extends the definition of decision lists to represent multiple concepts and, more importantly, has provided a behaviorally incremental algorithm for learning decision lists. Such an algorithm can learn from data that is potentially noisy and inconsistent. Moreover, experiments have shown that the efficiency of this incremental algorithm is comparable to some widely used nonincremental concept learning algorithms.

There are several future directions to improve CDL2. One is to use statistics to record historical information so that examples need not be stored. This would make CDL2 incremental not only in behavior but also in storage and processing costs. Another is to increase the expressiveness of decision tests so that decision lists can represent concepts with arbitrary shapes and concepts with predicates.

I thank Paul Utgoff, Jim Talley, Ray Mooney and Jude Shavlik for providing tools and data for the experiments, Michael Huhns, Jim Barnett, Jim Talley, Mark Derthick and three anonymous reviewers for their comments on the earlier draft. Special thanks to Phil Cannata who provides both moral and financial support for this work.

References

(Barto, 1985) A.G. Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

(Breiman *et al.*, 1984) L. Breiman, L.H. Fredman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA, 1984.

(Pagallo and Haussler, 1990) Guilia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 1990.

(Quinlan, 1983) R.J. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning.* Morgan Kaufmann, 1983.

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

(Rivest, 1987) L. Ronald Rivest. Learning decision lists. *Machine Learning*, 2, 1987.

(Salzberg, 1991) S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6(3), 1991.

(Shen, 1989) W.M. Shen. *Learning from the Environment Based on Actions and Percepts.* PhD thesis, Carnegie Mellon University, 1989.

(Shen, 1990) W.M. Shen. Complementary discrimination learning: A duality between generalization and discrimination. In *Proceedings of Eighth AAAI*, Boston, 1990.

(Shen, 1992) W.M. Shen. *Autonomous Learning from the Environment.* W.H. Freeman, Computer Science Press. Forthcoming. 1992.

(Utgoff, 1989) P.E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.