University
of Southern
California

Efficient Incremental Induction of Decision Lists

--Can Incremental Learning Outperform Non-Incremental Learning?

Wei-Min Shen

USC/Information Sciences Institute

January 1996

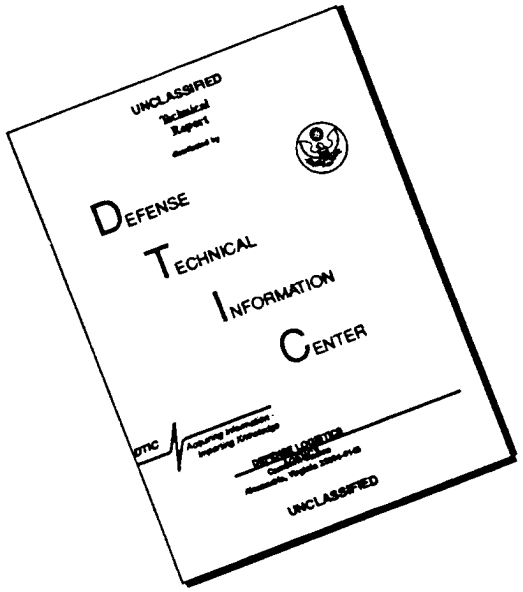
ISI/RR-96-433

INFORMATION
SCIENCES
INSTITUTE

310/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1996	Research Report

4. TITLE AND SUBTITLE

Efficient Incremental Induction of Decision Lists
--Can Incremental Learning Outperform Non-Incremental Learning?

5. FUNDING NUMBERS

Rome Labs/ARPA:
F30602-94-C-0210
and
MDA972-94-2-0010
CA: C94-0031
Motorola/ECC: None

6. AUTHOR(S)

Wei-Min Shen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

ISI/RR-96-433

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

ARPA	Motorola, Inc.	Rome Labs	Eastman Chemical Co.
3701 N. Fairfax Dr.	5918 W.Courtyard Dr.	Griffiss AFB	Lincoln Street
Arlington, VA 22203	Austin,TX 78730	NY 13441	Kingsport, TN 37662

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Although incremental learning has many advantages in theory, it is not in practice as widely used as non-incremental learning for real-world applications. One major reason for this situation is the lack of incremental algorithms that can perform as fast as non-incremental algorithms in general. In this paper, we present an effective yet very efficient incremental algorithm CDL4 for learning decision lists whose complexity is $O(dn^2)$, where d is the number of attributes and n the number of training instances. On the experiments we have conducted, CDL4's performance is as fast and accurate as the best non-incremental learning algorithms for batch tasks, and is much faster than the best-known incremental and non-incremental learning algorithms for serial tasks. We also show that efficient incremental algorithms can provide new research opportunities for learners to actively select training instances for better accuracy and higher speed, and that incremental learning may eventually outperform non-incremental learning in many aspects.

14. SUBJECT TERMS

algorithms, decision lists, incremental learning, inductions

15. NUMBER OF PAGES

18

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Efficient Incremental Induction of Decision Lists

— Can Incremental Learning Outperform Non-Incremental Learning?

Wei-Min Shen
Information Sciences Institute and
Computer Science Department
University of Southern California
Marina del Rey, CA 90292
shen@isi.edu

Abstract

Although incremental learning has many advantages in theory, it is not in practice as widely used as non-incremental learning for real-world applications. One major reason for this situation is the lack of incremental algorithms that can perform as fast as non-incremental algorithms in general. In this paper, we present an effective yet very efficient incremental algorithm CDL4 for learning decision lists whose complexity is $O(dn^2)$, where d is the number of attributes and n the number of training instances. On the experiments we have conducted, CDL4's performance is as fast and accurate as the best non-incremental learning algorithms for batch tasks, and is much faster than the best-known incremental and non-incremental learning algorithms for serial tasks. We also show that efficient incremental algorithms can provide new research opportunities for learners to actively select training instances for better accuracy and higher speed, and that incremental learning may eventually outperform non-incremental learning in many aspects.

1 Introduction

Incremental learning is a style of learning where the learner updates its model of the environment (e.g., a hypothesis for an unknown concept) whenever a new significant experience becomes available. Considerable research has been devoted to this style of learning, including Samuel's checkers player [1959], Winston's algorithm [1975], Mitchell's Candidate Elimination Algorithm [1982], Fisher's COBWEB [1987], Laird et al's SOAR [1986], Sutton's temporal-difference learning [1988], Utgoff's ID5R [1989] and ITI [1993], Shen's CDL1 [1990], CDL2 [1992], and D^* [1993], and Rivest and Schapire's algorithm for learning finite state machines without resetting [1993].

Compared to non-incremental or batch learning, incremental learning has the advantages of being more widely applicable and reactive. For example, it can be applied to situations where input data come only in sequence and a timely updating model is crucial for actions. However, no existing incremental algorithms can perform as fast as non-incremental algorithms in general, and this limits the practice of incremental learning in real-world applications.

The slowness of incremental algorithms stems from the nature of incremental learning. Different from batch learning, incremental learning deals with theory revision at each time step instead of theory creation from scratch. An incremental algorithm, without knowing any training information that might become available in the future, must diagnose the deficiency of its current theory and make the best choice in revising the theory. Both these tasks, however, are not concerns for non-incremental learning.

To increase the efficiency of incremental algorithms, fast methods must be developed for both diagnosis and revision tasks. CDL4 achieves its speed by realizing that generalizing a theory is equivalent to discriminating its complement, thus using discrimination as a single mechanism for both theory generalization and specialization. When using CDL4 to learn a decision list, theory diagnosis becomes detecting incorrect decisions in the current list, and theory revision becomes discriminating an incorrect decision using the differences found between a misclassified training instance and the accumulated instances of that decision. Compared to the earlier algorithms in this family, such as CDL1 and CDL2, CDL4 uses a very efficient strategy to find the differences between instances and can deal with both discrete and continuous attributes in a uniform fashion.[1]

CDL4 has been applied to several (very) large learning tasks. Experimental results have shown that, with roughly the same accuracy, CDL4 is much faster than other incremental algorithms (e.g., ITI) for serial tasks, and most surprisingly, is as fast as the best-known non-incremental algorithms (e.g., C4.5) for batch tasks. Furthermore, in some learning tasks, CDL4 achieves much higher accuracy than other algorithms when the training instances are presented in certain orders. This phenomenon suggests that by arming themselves with a strategy to choose training instances actively, incremental learning algorithms could eventually outperform non-incremental algorithms in both speed and accuracy.

In the rest of this paper, Section 2 presents some key arguments for why incremental learning is necessary. Section 3 gives a brief review on decision lists. Section 4 and 5 describes the CDL4 algorithm in detail. Section 6 reports the experimental results of CDL4

[1]CDL3 [Shen, 1994] is an algorithm that learns decision lists in the format of First-Order Logic.

in three learning tasks and provide a comparison with other algorithms. Section 7 analyzes the complexity of CDL4. Section 8 examines the effects of training order on CDL4's accuracy performance and calls for new research in actively selecting training instances for incremental learning algorithms. The paper is concluded with a list of future research directions.

2 The Significance of Incremental Learning

What is incremental learning and why is it useful? In this section, we argue that in the context of agent learning from their environment [Shen, 1994], incremental learning is the only way to provide a continuously updated model of the environment for agents to intelligently choose their actions (e.g., actively select training instances for concept learning). In many situations, such an ability is crucial for the survival of the agents.

The task of learning a target concept C from a sequence of training instances can be summarized as follows: the learner is to construct a concept hypothesis H_j to approximate C based on the training instances, $x_1, ..., x_j$, that have been seen so far. Such a task can be accomplished by either batch (non-incremental) learning or incremental learning.

In batch learning, a hypothesis H_j at time j is created from the scratch based on all the instances from x_1 through x_j, without any intermediate hypothesis, as in the following function $\mathcal{F}_1$:

$$H_j = \mathcal{F}_1[x_1, ..., x_j].$$

In incremental learning, on the other hand, H_j is constructed recursively based on the intermediate hypothesis H_{j-1} and the current instance x_j, as follows:

$$H_i = \mathcal{F}_2[H_{i-1}, x_i], \text{where } 1 \leq i \leq j, \text{ and } H_0 \text{ is given.}$$

Between these two styles of learning, empirical evidence has shown that $\mathcal{F}_1$ can be computed much faster than $\mathcal{F}_2$ and that raises the question whether the latter is needed at all. The three early arguments in favor of $\mathcal{F}_2$ are: the relief of keeping past instances or equivalent information [Michalski, 1985; Schlimmer and Fisher, 1986], the availability of a continual hypothesis for reacting to new stimuli [Schlimmer and Fisher, 1986], and the suitability for serial tasks in which instances come only in sequence [Utgoff, 1989]. Unfortunately, the strength of the first argument is weakening because the low cost and high speed of modern memory. The strength of the last two arguments is based on the assumption that there is an efficient mechanism for updating the hypothesis incrementally, for otherwise one can always use a $\mathcal{F}_1$ algorithm to build a new H_j from scratch for every new instance x_j, or accumulate enough instances from the stream and update the hypothesis periodically. Since the most efficient and accurate concept learning algorithms today are of $\mathcal{F}_1$ type, doubts remain whether $\mathcal{F}_2$ type of learning is indeed necessary.

Notice that all the earlier arguments for $\mathcal{F}_2$ style of learning assume that the learners are passive. They are *given*, either in batch or in sequence, the training instances to learn. In many applications, however, the learner can or must actively select the next training instance. For example, in language learning, a child or a system can ask specific questions. In learning a finite state machine, the learner must act upon the machine. In learning grammars, the learner must query the teacher for new strings and their memberships (e.g., [Angluin, 1987]).

In all these cases, the next training instance will not come unless the learner performs an action.

Active learning has two important implications. First, choosing the next action imposes a firm demand on a model that is continuously updated based on the learner's experience in the environment. It will be too costly to delay updating model and wait for more instances to accumulate. Second, choosing the "right" next instance may affect the performance of learning dramatically. For example, in function approximation (a much more formal type of "concept learning"), the samples used in interpolation must be selected according to Čebyšev's formula [Nikolskii, 1963] or else the approximation may not converge.

Having said that the most important aspect of incremental learning is to have a continuously updated model for active learning, we may allow ourselves to use some memory in the process of learning. In other words, we may consider the following $\mathcal{F}_3$ style

$$H_i = \mathcal{F}_3[H_{i-1}, x_1, ..., x_i], \text{where } 1 \leq i \leq j, \text{ and } H_0 \text{ is given.}$$

also as incremental learning even though it is allowed to access subsets of previous instances. Such a style of learning seems justifiable because intelligent creatures do have memories, and remembering past experience can be very useful in dealing with noise. We may call this style of learning *behavior-incremental* learning for it does incrementally update the hypothesis H at each step.

Thus, the real question is not whether incremental learning is needed or not, but whether we can develop efficient and accurate incremental learning algorithms to make the task of continuously updating a model practical. The CDL4 algorithm is one of such attempts.

3 Decision Lists

Before we present the CDL4 algorithm, let us first review briefly the notion of decision lists. Proposed by Rivest [1987] for Boolean formulas, 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, we shall refer a pair (f_j, v_j) as 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}, and f_j is a conjunction of predicates.

Given a decision list L, the decision on an instance x is defined to be $D_j = (f_j, v_j)$, where j is the least index such that $f_j(x) = 1$. In this case, we say that x is *covered* by D_j, that D_j is the *covering decision* of x, and that x is classified by L as an instance of v_j. For example, given the following decision list:

$$L_1 = (((a_3 < 1000.0) \wedge (a_4 \neq \text{'u'}), C_1), ((a_1 < 980.0), C_2), (true, C_1))$$

where a_i are attribute names and C_i are concept names, and an instance x = [(a_1=300.0) (a_2='g') (a_3=100.0) (a_4='u')], then this instance x is covered by the second decision $((a_1 < 980.0), C_2)$ and it is classified as an instance of C_2.

4 The CDL4 Algorithm

CDL4 is a $\mathcal{F}_3$ style algorithm whose goal is to construct incrementally a decision list L from a sequence of training instances under the supervision of a teacher. When a new instance x is misclassified by L (i.e., the classification made by L does not match the classification, say C_x, given by the teacher), CDL4 revises L to classify x correctly. To accomplish this task, the algorithm must

1. Determine whether there is a misclassification and which decision D_j in L is responsible;
2. Determine to what extend D_j should be discriminated;
3. Discriminate D_j and modify L.

For the first task, CDL4 searches through the decision list and returns the decision $D_j = (f_j, v_j)$, where j is the least index such that $f_j(x) = 1$. If v_j is not equal to x's classification provided by the teacher, then f_j is the decision test that needs to be discriminated. Otherwise, the instance x is recorded as an example of D_j.

If the new instance is misclassified by the current decision list, then CDL4 performs the second task by finding out the minimal differences between the new instance x and the set E of previous examples of D_j (recall that an example of a decision is an instance recorded under that decision). The key idea here is to first construct a *differentiator* predicate for each attribute (except those that have unknown values in the new instance), and then select these differentiators one at a time, according to their strengths to distinguish x from E, until all (or most of) the examples in E are differentiated from x. The final result is a disjunction of a set of differentiators, each of which is true for some examples in E but false for the new instance x.

Given a new instance x and a set of previous examples E, the differentiators are constructed as follows. If an attribute a_k is discrete, then the corresponding differentiator is constructed as a predicate

$$T_k(y) \equiv (a_k \neq v_k),$$

where y is a parameter to be bound to an instance (or an example), a_k is an attribute variable for y, and v_k is a constant equal to the value of a_k in the new instance x. For example, if the new instance x is [..., (a_4='g')], then the differentiator $T_4(y)$ is defined as ($a_4 \neq$'g'). This differentiator is true for those examples in E whose a_4 value is not equal to 'g', and is false for the instance x.

If an attribute a_k is continuous, then CDL4 scans the attribute values, represented as e_{a_k}, in the example set E, and constructs the corresponding attribute differentiator as

$$T_k(y) \equiv \begin{cases} (a_k < v_k) & \text{if more than half of the values } e_{a_k} \text{ in } E \text{ are less than } v_k \\ (a_k > v_k) & \text{otherwise} \end{cases}$$

where y and v_k are defined as before. Again, this differentiator is true for some of the examples in E and is false for the instance x. The motivation of this definition is to construct a differentiator that can distinguish as many examples in E as possible from the new instance

x. Since differentiators are guaranteed to be false on the new instance x, their strength are computed as the number of examples in E for which they return true.

To illustrate the entire step of finding differences between x and E, suppose a decision has three examples: [300.0,g,100.0,u], [300.0,g,1000.0,g], [1000.0,g,1200.0,g], and the new instance is [980.0,g,1000.0,u]. In this case, the differentiators are defined as $T_1(y) \equiv (a_1 < 980.0)$ with strength 2, $T_2(y) \equiv (a_2 \neq \text{'}g\text{'})$ with strength 0, $T_3(y) \equiv (a_3 > 1000.0)$ with strength 1, and $T_4(y) \equiv (a_4 \neq \text{'}u\text{'})$ with strength 2. The minimal differences are returned as $((a_1 < 980.0), (a_4 \neq \text{'}u\text{'}))$ because these two differentiators have higher strengths and they can distinguish all three examples from the new instance.

After the differences between a misclassified instance and the previous examples of the covering decision D_j are found, the third task is to discriminate the covering decision and modify the decision list. For this task, CDL4 replaces $D_j = (f_j, v_j)$ by a new decision $D'_j = (f_j \wedge \Sigma, v_j)$, where Σ is the differences found in the second step. Since Σ is a disjunction of differentiators σ_1, ..., σ_d, the new decision D'_j can be represented as a list of new decisions:

$$(f_j \wedge \sigma_1, v_j), \cdots, (f_j \wedge \sigma_d, v_j).$$

Note that none of the new decisions will capture the new instance. The previous examples of the old decision D_j 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, $1 \leq i \leq d$, such that $[f_j \wedge \sigma_i](e) = 1$.

After the incorrect decision is replaced, the new instance x continues to look for a decision in the remainder of the old decision list. Suppose the new covering decision is D_k, where $k \geq (j + d)$. If $v_k = C_x$ (i.e., the decision is correct), then the instance is recorded as an example of D_k. Otherwise, D_k is discriminated 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 $v_k = C_x$ and x can be added to D_k's example list, or D_k is discriminated and a new default decision $(true, C_x)$ is appended at the end of the list with x as its only example. The pseudocode of the CDL4 algorithm is listed in Figure 1.

To illustrate the algorithm, let us go through an example to see how CDL4 learns a decision list representing the following binary concept: $C = \overline{a}_1\overline{a}_3 \vee a_1a_2 \vee a_1\overline{a}_2a_4$. We assume there are five attributes a_1, a_2, a_3, a_4, and a_5, and use a_i, where $i = 1, ..., 5$, to represent $(a_i = 1)$, and $\overline{a}_i$ to represent $(a_i \neq 1)$ or $(a_i = 0)$. The training instances are given from $x_0 = [\overline{a}_1\overline{a}_2\overline{a}_3\overline{a}_4\overline{a}_5]$ to $x_{31} = [a_1a_2a_3a_4a_5]$ in that order.

When the first training instance $x_0 = [\overline{a}_1\overline{a}_2\overline{a}_3\overline{a}_4\overline{a}_5]$ $(\in C)$ arrives, CDL4 initiates a decision list with a single "default" decision:

$$((true, C)),$$

and the instance is recorded as an example of this sole decision. Since the next three instances $x_1 = [\overline{a}_1\overline{a}_2\overline{a}_3\overline{a}_4a_5]$, $x_2 = [\overline{a}_1\overline{a}_2\overline{a}_3a_4\overline{a}_5]$, and $x_3 = [\overline{a}_1\overline{a}_2\overline{a}_3a_4a_5]$ are all being predicted correctly, they also become examples of the default decision. The fifth instance is $x_4 = [\overline{a}_1\overline{a}_2a_3\overline{a}_4\overline{a}_5]$ $(\in \overline{C})$, and CDL4's prediction is wrong. The difference between (x_0, x_1, x_2, x_3) and x_4 is found to be $(\overline{a}_3)$, so the decision is shrunk to be $(\overline{a}_3, C)$, and a new default $(true, \overline{C})$ (with x_4 as its example) is appended at the end. The new decision list is now:

$$((\overline{a}_3, C)(true, \overline{C})).$$

Inputs: A decision list D for a set of unknown target concepts $C_1, ..., C_t$,
a new instance x of one of the targets C_x,
and a set E of previous examples of these targets;
Outputs: A refined decision list D;
Procedure CDL4(D, E, x):
If D is empty,
then $D = \{D_0\} = \{(true, C_x)\}$ and record x as an example of D_0,
else loop:
Let $D_j = (f_j, v_j)$ be the covering decision on x,
If the decision is correct, i.e., $v_j = C_x$,
then record x as an example of D_j and exit the loop,
else let $\Sigma = (\sigma_1, ..., \sigma_d)$ be the differences found between previous examples of D_j and x,
Replace (f_j, v_j) by $(f_j \wedge \sigma_1, v_j), \cdots, (f_j \wedge \sigma_d, v_j)$,
Distribute the examples of D_j to the new decisions,
If D_j was the last decision, then append $(true, C_x)$ at the end of D.

Figure 1: The CDL4 Incremental Learning Algorithm

With this decision list, the succeeding instances are predicted correctly until $x_{16} = [a_1\overline{a}_2\overline{a}_3\overline{a}_4\overline{a}_5]$. CDL4's decision is $(\overline{a}_3, C)$ but the instance belongs to $\overline{C}$. Comparing the examples of the decision with the new instance, CDL4 finds the difference to be $\overline{a}_1$. The decision is then replaced by $(\overline{a}_1\overline{a}_3, C)$:

$$((\overline{a}_1\overline{a}_3, C)(true, \overline{C})).$$

The troublesome instance then finds $(true, \overline{C})$ to be correct and becomes an example of the default decision. For succeeding instances, CDL4's prediction is correct on $x_{17} = [a_1\overline{a}_2\overline{a}_3\overline{a}_4a_5]$ but wrong on $x_{18} = [a_1\overline{a}_2\overline{a}_3a_4\overline{a}_5]$. The decision is $(true, \overline{C})$ but x_{18} is in C. The differences found are $(\overline{a}_4, \overline{a}_1)$, so the decision $(true, \overline{C})$ is replaced by $((\overline{a}_4, \overline{C}), (\overline{a}_1, \overline{C}), (true, C))$, yielding:

$$((\overline{a}_1\overline{a}_3, C)(\overline{a}_4, \overline{C})(\overline{a}_1, \overline{C})(true, C)).$$

With this decision list, CDL4 correctly classifies the next five instances, x_{19} through x_{23}, but fails on $x_{24} = [a_1a_2\overline{a}_3\overline{a}_4\overline{a}_5]$. The decision is $(\overline{a}_4, \overline{C})$ but the instance is in C. The difference between this decision's examples $(x_{16}, x_{17}, x_{20}, x_{21})$ and x_{24} is found to be $(\overline{a}_2)$. The decision $(\overline{a}_4, \overline{C})$ is then shrunk into $(\overline{a}_4\overline{a}_2, \overline{C})$ and the new decision list is now:

$$((\overline{a}_1\overline{a}_3, C)(\overline{a}_4\overline{a}_2, \overline{C})(\overline{a}_1, \overline{C})(true, C))$$

This decision list is equivalent to the target concept C, and it correctly classifies all the rest instances x_{25} through x_{31}.

5 Improvements of CDL4

The CDL4 algorithm can be improved in several ways. The first one is to construct decision lists that are shorter in length. CDL4 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 target 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 a conjunction of those predicates that appear in *both* f_i and f_j. (Conceptually, f_m covers at least the union 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 $((\overline{a}_3, C)(a_1, \overline{C})(true, C))$, and examples of these three decisions are $\{[\overline{a}_1\overline{a}_2\overline{a}_3], [\overline{a}_1 a_2\overline{a}_3]\}$, $\{[a_1a_2a_3], [a_1\overline{a}_2a_3]\}$, and $\{[\overline{a}_1a_2a_3]\}$, respectively. Suppose the current instance is $x = [a_1\overline{a}_2\overline{a}_3]$ $(\in \overline{C})$, for which CDL4 has made the wrong decision $(\overline{a}_3, C)$. Since the difference between $\{[\overline{a}_1\overline{a}_2\overline{a}_3], [\overline{a}_1a_2\overline{a}_3]\}$, (the examples of D_1) and x is $(\overline{a}_1)$, the decision $(\overline{a}_3, C)$ should be replaced by $(\overline{a}_3\overline{a}_1, C)$, which would result in a new decision list $((\overline{a}_3\overline{a}_1, C)(a_1, \overline{C})(true, C))$. However, the new decision $(\overline{a}_3\overline{a}_1, C)$ can be merged with $(true, C)$ because it has the same decision value, and none of its examples can be captured by $(a_1, \overline{C})$, and the merged decision, which is $(true, C)$, does not block any other decisions following it. Thus, the decision list is shortened to: $((a_1, \overline{C})(true, C))$.

The second improvement over the basic CDL4 algorithm is to deal with instances that belong to different concepts simultaneously. (This is sometimes called the noise in the training data.) To handle such instances, 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 C_x if either $v_j = C_x$, or x is already an example of D_j.

For example, suppose a decision (a_1a_2, C) currently has one example $\{[a_1a_2] \in C\}$ and a new instance $[a_1a_2]$ arrives and claims to be in $\overline{C}$, then the decision (a_1a_2, C) will be considered to be correct because $[a_1a_2]$ 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 adopt a policy that the value of a decision is always the same as the concept value that is supported by the most examples. For instance, if another example $[a_1a_2] \in \overline{C}$ is claimed by the above decision again, then the value of the decision will be changed from C to $\overline{C}$ because $\overline{C}$ will be supported by two examples vs. only one example for C. With this policy, we must also relax the criteria for finding differences. Instead of returning a set of differentiators that can distinguish *all* previous examples of a decision from the new instance, one may find it is enough to distinguish *most* previous examples (e.g., above a user-specified threshold).

6 Experiments with CDL4

In this section, CDL4 is compared with three existing algorithms: CN2 [Clark and Niblett, 1989], C4.5 [Quinlan, 1993], and ITI [Utgoff, 1993], on three (very) large learning tasks. These algorithms represent a wide range of non-incremental or incremental algorithms and different concept representations such as decision trees and classification rules. CN2 is chosen for its ability to learn ordered (and unordered) decision rules, which are very similar to decision lists learned by CDL4. C4.5 is chosen for its high performance for learning decision trees and its extension C4.5rules to learn unordered decision rules. Both CN2 and C4.5 are well-known non-incremental algorithms. For incremental algorithms, we have chosen ITI[2], which is a descendant of ID5R and the newest and perhaps the best algorithm that learns decision trees incrementally. It is also known that for serial tasks, ITI is faster than rerunning C4.5 on the new training set each time a new instance is accumulated. All these algorithms, including CDL4, are implemented in the C language and running on a SPARC-20 machine with 128M main memory. Furthermore, all source codes are obtained directly from their original authors and this author did not reimplement anything.

The experiments chosen here represent several different flavors of learning tasks. The first one is to learn a concept of "win" or "lose" from a set of 551 chess end games that represented by 39 binary attributes. Although this task does not have a large size and contains no noise, it is chosen for historical reasons. (The task is first proposed by Quinlan [1983] and subsequently used in many incremental learning papers.) The second task is to recognize hand-written numerals (from "0" to "9") from bit maps. The size of this task is relatively large (3301 bit maps each having 64 bits), and there is noise in the data (i.e., two bit maps that look the same are labeled as different numerals). Finally, the third task is to learn to recognize different font letters (from "A" to "Z") that are represented as 16 attributes. The size of this task is large, including 20,000 instances, and its attributes have continuous values. In fact, this is the largest learning task we found in the ML Repository maintained at the University of California, Irvine (ml-repository@ics.uci.edu).

The results of these three experiments are reported in Table 1, 2, and 3. The format of the tables are the same: the first column is the name of the algorithms, along with an indication whether the training instances are presented as a batch or a sequence (serial). The second column states for serial tasks if the order of the training instances are given as it is or chosen by hand. The third column gives the average accuracies (with standard deviations) of learned concepts on the unseen testing data. The fourth column is the average of CPU time taken in the cross-validation learning (testing time is not included). Finally, the last column is an indication of the size of the learned concept. If the learned concept is a decision list (or a set of rules in the case of CN2 and C4.5rules), then the value in this column is the average number of decisions or rules in the decision list or the rule set. If the concept is a decision tree (in the case of C4.5 with pruning and ITI), then the value is the average number of nodes in the decision tree. Note that the number of nodes in decision trees cannot be directly compared with the number of rules; they are included here only for the completeness.

[2]Both ITI and ID5R are $\mathcal{F}_3$ style algorithms.

Table 1: Comparison on classifying chess end games (551×39)

Program (Mode)	TrainingOrder	Accuracy	CPU sec.	ConceptSize
C4.5 (batch)	N/A	91.10% (3.67%)	0.30	46.6
C4.5rules (batch)	N/A	92.90% (3.40%)	3.50	21.4
CN2 (batch)	N/A	84.21% (6.85%)	7.70	17.0
ITI (batch)	N/A	91.96% (2.29%)	2.23	142.8
ITI (serial)	Insensitive	91.96% (2.29%)	95.88	142.8
CDL4 (serial)	Given	87.66% (2.82%)	0.31	76.4
CDL4 (serial)	Choose	96.19% (2.36%)	0.24	41.7

6.1 Classifying Chess End Games

In this experiment, each algorithm performs a 10 run cross-validation, using 90% of the randomly selected data for training and the remaining 10% for testing. The results in Table 1 indicate that the speed of CDL4 is almost the same as the best non-incremental algorithm C4.5 in batch mode (0.31 vs. 0.30), and is much faster than the other non-incremental or incremental algorithms. When the training orders are given as those selected by the cross-validation, CDL4's prediction accuracy is not the best but better than the non-incremental algorithm CN2. However, if a good training order is chosen for each training set, then CDL4 is faster (0.24) and more accurate (96.19%) than other algorithms. In this experiment, a good training order is determined as follows. We order the training instances by the values of each attribute (in both increasing and decreasing orders), and among these orders we select one that gives the best test accuracy. Note that there are $N!$ possible orders for a training set of N, and we have only considered $2 \times B$ of them in this experiment, where B is the number of attributes. In general, we have not yet found a systematic way to select a good training order (see more discussion in Section 8).

6.2 Recognizing Hand-Written Numerals

In this experiment, each algorithm again performs a 10 run cross-validation, using 90% of the randomly selected data for training and the remaining 10% for testing. As shown in Table 2, CDL4's speed (15.87 or 14.06) is again close to the best non-incremental algorithm (5.20 for C4.5) in batch mode and much faster than serial ITI (1578.49). When the training orders are given as those selected by the cross-validation, CDL4's prediction accuracy (75.07%) is lower than others in this domain. However, when good training orders are chosen, its accuracy is 77.91% and again higher than the others. Like the strategy we used for the chess domain, the good orders are selected among those orders based on the values of attributes (there are 64×2 such orders in this domain). In addition, we have also analyzed the error matrix (how much each class is misclassified by others) and considered some training orders based on class values that are "similar." For example, instances of "3" and "5" and instances of "8" and "9" are presented adjacently in such selected training orders.

Table 2: Comparison on the numeral recognition task (3331×64)

Program (Mode)	TrainingOrder	Accuracy	CPU sec.	ConceptSize
C4.5 (batch)	N/A	77.70% (2.88%)	5.20	548.2
C4.5rules (batch)	N/A	76.10% (3.01%)	342.27	124.0
CN2 (batch)	N/A	77.89% (1.61%)	161.40	142.0
ITI (batch)	N/A	77.31% (3.09%)	39.81	1182.20
ITI (serial)	Insensitive	77.31% (3.09%)	1578.49	1182.20
CDL4 (serial)	Given	75.16% (3.51%)	15.87	547.4
CDL4 (serial)	Choose	77.91% (1.73%)	14.06	438.4

Table 3: Comparison on the letter recognition task (20,000×16)

Program (Mode)	TrainingOrder	Accuracy	CPU sec.	ConceptSize
C4.5	N/A	87.20% (0.49%)	125.30	2199.0
C4.5rules	N/A	85.40% (0.49%)	26552.10	570.0
CN2	N/A	87.94% (0.89%)	8456.36	480.0
ITI (batch)	N/A	86.84% (0.48%)	129.60	4115.0
ITI (serial)	Insensitive	86.84% (0.48%)	31169.42	4115.0
CDL4 (serial)	Given	80.92% (0.25%)	123.02	2177.0
CDL4 (serial)	Choose	84.71% (0.86%)	120.20	2101.4

6.3 Recognizing Letters in Different Fonts

In this very large learning task (20,000 instances), each algorithm performs 5 run cross-validation, using 80% of the randomly selected data for training and the remaining 20% for testing. As shown in Table 3, CDL4's speed (123.02 and 120.20) is again very close to the best non-incremental algorithm C4.5 in the batch mode (125.30), and much faster than the other non-incremental algorithms in batch mode (26552.10 for C4.5rules and 8456.36 for CN2) and the other incremental algorithm in serial mode (ITI 31169.42). CDL4's prediction accuracy is both behind the others (80.92% for the given training orders and 84.71% for the selected orders using the same method as in the domain of hand-written numerals) but it is not too far away from the best of 87.94% of CN2. For reference, the best known result in this domain is 95.7%, obtained by an algorithm using a nearest-neighbor classifier [Fogarty, 1992]. Since that algorithm does not build descriptions of concepts, it is not included in this paper. Notice again that in selecting good training orders we only considered a very small subset of possible orders.

7 Complexity Analysis of CDL4

This section analyzes the complexity of CDL4. Let d be the number of attributes, m the number of previous examples, and c the length of the current decision list, we first consider how many comparison operations are needed for CDL4 to revise the current decision list for a new instance x. Here, we assume x will cause one and only one decision to be split.

Observe that the number of comparisons needed to find a covering decision D_j for x is at most cd because the size of each decision test is no greater than d. Assume the m examples are evenly distributed in c decisions, then the faulty decision D_j has m/c examples and finding differences between x and these examples takes at most $(dm/c)+(m/c)$ comparisons, where dm/c is needed for computing the strength of each differentiator, and m/c is needed for determining how many differentiators should be returned. Since there are at most d differentiators, rewriting D_j generates at most d new decisions, and distributing the examples of D_j to these new decision takes at most d^2m/c (each example takes at most d^2 comparisons to find a new host decision). Thus, if we assume $c > d$, then the total number of comparisons needed for revising an existing decision list for a new instance is:

$$cd + dm/c + m/c + d + d^2m/c = O(cdm/c) = O(dm).$$

Assume the whole training set has n instances, then the total number of comparisons to process all n instances incrementally is

$$\sum_{m=1}^{n} O(dm) = O(dn^2).$$

This complexity of CDL4 is at least comparable with most non-incremental algorithms. The complexity of CN2 is dn^2 [Clark and Niblett, 1989] but our experiments show CDL4 may be faster by a constant factor. According to [Utgoff, 1989], ID3 (an earlier version of C4.5) takes $O(nd^2)$ additions and $O(2^d)$ multiplications (one for each E-score calculation), and ID5R (an earlier version of ITI) takes $O(ndb^d)$ additions and $O(nb^d)$ multiplications, where b is the maximum number of possible values for an attribute. CDL4 uses only comparison operations and its complexity has no exponential components.

8 The Role of Active Learning

As we can see from the description and experiments, CDL4's prediction accuracy on unseen instances are sensitive to the orders in which training instances are presented. This is because CDL4 revises the current decision list based on the differences between the current instance and previous examples of the similar concept. With different training orders, the differences found by CDL4 may be different and different decision list will be built. For example, if more representative instances (i.e., those instances represent the critical differences between target concepts) appear early in a training order, then CDL4 will build a shorter decision list.

This effect of training orders on the accuracy of incremental learning is actually a two-edged sword. On the one hand, one can exploit the training orders to build more accurate

concept hypotheses, as we have attempted to do in our experiments. On the other hand, there is a lack of methods to select good training orders and manual selection is trail-and-error and time consuming process. For example, Dieterich [1995] has concluded that this is a common weakness of incremental learning algorithms.

Nevertheless, how to select training instances (also known as *membership queries* in computational learning theory) for incremental learning is an active research topic (see for example [Cornuejols *et al.*, 1993]). Despite some known negative results (for example, membership queries will not help with predicting CNF or DNF formulas [Angluin and Kharitonov, 1991]), there are many positive results regarding active learning. For example, it is known that with membership queries (and equivalence queries) the class of DFA is learnable [Angluin, 1987], and the class of decision trees is also learnable [Bshouty, 1993]. Furthermore, there are some initial results on how to choose the training order for learning certain types of concepts [Goldman *et al.*, 1989], although their emphasis is on minimizing the number of prediction mistakes in learning (regardless how much time it takes before a mistake can happen).

With an efficient incremental learning algorithm for learning decision lists such as CDL4, we are hoping to find similar methods as Bshouty's Λ-Algorithm that can choose instances actively and make the learning of decision lists faster and more accurate. For example, in our experiments with different training orders, we found that analysis of error distributions is a good way to find a set of classes that are most "confusing" so that we can choose an order in which instances of these classes are presented adjacently. For applications where instances are free to select, such methods can be used to learn the target concept quickly. For applications where the set of training instances is fixed, such methods can be used to choose the best training order of these instances so that the learner can make the most progress towards the target concept. If such methods can indeed be found, then it is foreseeable that incremental learning could outperform non-incremental learning in many aspects, such as speed, accuracy, and noise tolerance.

9 Conclusions and Future Work

In this paper, we have presented a very efficient incremental algorithm CDL4 for learning decision lists. Experimental results have shown that for serial tasks, CDL4 is much faster than other incremental algorithms such as ITI and ID5R. This implies that CDL4 is also much faster, for serial tasks, than any non-incremental algorithms rerunning on the training set each time a new instance is accumulated.

Most surprisingly, CDL4 in its serial mode can also perform as fast and accurate as the best-known non-incremental algorithms such as C4.5 and CN2 for batch tasks. This result has put incremental learning, for the first time to our knowledge, on the same performance stage as non-incremental learning for both serial and batch tasks.

The research on CDL4 has also revealed several important future research directions. In order to further develop incremental learning algorithms, we must work on algorithms that can actively select training instances (or training orders when instances are limited). Positive results on learning decision trees by membership queries have already provided some evidence that this may be a feasible task.

To be truly incremental, we must also develop algorithms that use fixed amount of space. For example, CDL4 should not require remembering all previous examples, but only those important characteristics of each example set associated with each decision. Some initial work in this direction is already underway.

Finally, we shall point out that the sensitivity of incremental learning algorithms to training orders should not be viewed as a negative aspect of incremental learning. Instead, we shall use this as an unique opportunity to further develop incremental learning algorithms that can eventually outperform non-incremental learning in many aspects.

References

Angluin, D and Kharitonov, M. 1991. When won't membership queries help? In *Proc. of 23th Annual ACM Symposium on Theory of Computing.* 444–454.

Angluin, D. 1987. Learning regular sets from queries and counter-examples. *Information and Computation* 75(2):87–106.

Bshouty, N. H. 1993. Exact learning boolean functions via the monotone theory. In *IEEE Symposium on Foundations of Computer Science (FOCS).*

Clark, P. and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 3:261–284.

Cornuejols, A.; Fisher, D.; Goldman, S.; Saitta, L.; and Schlimmer, J. 1993. Training issues in incremental learning. Technical Report AAAI Spring Symposium Working Notes, Stanford University.

Dietterich, Thomas G. 1995. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning* 19:5–20.

Fisher, D.H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2:139–172.

Fogarty, T.C. 1992. First nearest neighbor classification on frey and slate's letter recognition problem. *Machine Learning* 9:387–388.

Goldman, S.; Rivest, R.; and Schapire, R. 1989. Learning binary relations and total orders. In *Proceedings of 30th Annual Symposium on Foundation of Computer Science.* 46–51.

Laird, J.E.; Rosenbloom, P.S.; and Newell, A. 1986. Chunking in SOAR: the anatomy of a general learning mechanism. *Machine Learning* 1:11–46.

Michalski, R.S. 1985. Knowledge repair mechanisms: Evolution versus revolution. In *Proceedings of Third International Machine Learning Workshop*, Rutgers University.

Mitchell, T.M. 1982. Generalization as search. *Artificial Intelligence* 18:203–226.

Nikolskii, S.M. 1963. *Approximations of Functions.* MIT Press. chapter XII.

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

Quinlan, R. J. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

Rivest, R.L. and Schapire, R.E. 1993. Inference of finite automata using homing sequences. *Information and Computation.*

Rivest, L. Ronald 1987. Learning decision lists. *Machine Learning* 2:229–246.

Samuel, A. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3:211–229.

Schlimmer, J. and Fisher, D. 1986. A case study of incremental concept learning. In *Proceedings of Fourth National Conference on Artificial Intelligence.* MIT Press.

Shen, W.M. 1990. Complementary discrimination learning: A duality between generalization and discrimination. In *Proceedings of Eighth National Conference on Artificial Intelligence.* MIT Press.

Shen, W.M. 1992. Complementary discrimination learning with decision lists. In *Proceedings of Tenth National Conference on Artificial Intelligence.* MIT Press.

Shen, W.M. 1993. Learning finite state automata using local distinguishing experiments. In *Proceedings of IJCAI-93*, Chambery, France.

Shen, W.M. 1994. *Autonomous Learning from the Environment.* W. H. Freeman, Computer Science Press.

Sutton, R. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–44.

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

Utgoff, P.E. 1993. An improved algorithm for incremental induction of decision trees. In *Proceedings of the 10th International Machine Learning Workshop*, Rutgers University, New Brunswick, NJ.

Winston, P.H. 1975. Learning structural descriptions from examples. In *The psychology of computer vision.* MacGraw-Hill.