, , 1–42 ()

A Survey of Methods for Scaling Up Inductive Algorithms

FOSTER PROVOST provost@acm.org
Bell Atlantic Science and Technology, 500 Westchester Avenue, White Plains, New York 10604

VENKATESWARLU KOLLURI venkat@sis.pitt.edu
Department of Information Science, University of Pittsburgh, Pittsburgh, PA 15260, and Lycos, Inc., 5001 Centre Avenue, Pittsburgh, PA 15213

Editor: Usama Fayyad

Abstract. One of the defining challenges for the KDD research community is to enable inductive learning algorithms to mine very large databases. This paper summarizes, categorizes, and compares existing work on scaling up inductive algorithms. We concentrate on algorithms that build decision trees and rule sets, in order to provide focus and specific details; the issues and techniques generalize to other types of data mining. We begin with a discussion of important issues related to scaling up. We highlight similarities among scaling techniques by categorizing them into three main approaches. For each approach, we then describe, compare, and contrast the different constituent techniques, drawing on specific examples from published papers. Finally, we use the preceding analysis to suggest how to proceed when dealing with a large problem, and where to focus future research.

Keywords: scaling up, inductive learning, decision trees, rule learning

1. Introduction

The knowledge discovery and data mining (KDD) community has challenged itself to develop inductive learning algorithms that scale up to large data sets (Fayyad, Haussler, and Stolorz 1996) (Fayyad, Piatetsky-Shapiro, and Smyth 1996a) (Piatetsky-Shapiro, Brachman, Khabaza, Kloesgen, and Simoudis 1996). This paper[1] summarizes, categorizes, and compares various existing methods. We restrict the survey's scope to scalable algorithms, and do not consider issues of efficient file system design, storage design, network interface design, or problem formulation, except as they relate to the design of inductive algorithms. Although we believe the categorization and lessons apply more generally, our analysis focuses primarily on algorithms that build feature-vector-based classifiers (rather than those that include structural or relational terms) in the form of decision trees or rule sets.

We first address the meaning of "scaling up" and highlight important issues. We then show similarities between existing methods by grouping them into three high-level categories. Within each category, we discuss the techniques themselves in some detail, showing the similarities and differences between techniques of each type. Finally, we conclude with suggestions for research and practice that emerge from the survey's analysis.

2. Why scale up?

Organizations are amassing very large repositories of customer, operations, scientific, and other sorts of data. Fayyad et al. (1996b) cite several representative examples of databases containing many gigabytes (even terabytes) of data. KDD practitioners would like to be able to apply inductive learning algorithms to these large data sets in order to discover useful knowledge. The question of scalability asks whether the algorithm can process large data sets efficiently, while building from them the best possible models. However, the existence of very large data sets alone is not sufficient to motivate non-trivial scaling efforts. Why not just select a small subset of the data for data mining?

The most commonly cited reason for scaling up is that increasing the size of the training set often increases the accuracy of learned classification models (Catlett 1991b). In many cases, the degradation in accuracy when learning from smaller samples stems from overfitting due to the need to allow the program to learn *small disjuncts* (Holte, Acker, and Porter 1989), elements of a class description that cover few data items. In some domains small disjuncts make up a large portion of the class description (Danyluk and Provost 1993). In such domains, high accuracy depends on the ability to learn small disjuncts to account for these special cases. The existence of noise in the data further complicates the problem, because with a small sample it is impossible to tell the difference between a special case and a spurious data point.

Overfitting from small data sets also may be due to the existence of a large number of features describing the data. Large feature sets increase the size of the space of models. Searching through and evaluating more candidate models increases the likelihood that, by chance, the program will find a model that fits the data well (Jensen and Cohen 1999), and thereby increases the need for larger example sets (Haussler 1988). Things get particularly difficult when there are many features *and* there is the need to learn small disjuncts. Specifically, because large feature sets lead to large and often sparsely populated model spaces, a program biased to search for models covering special cases can be inundated with small disjuncts from among which it cannot choose.

Some data mining applications are concerned not with predictive modeling, but with the discovery of interesting knowledge from large databases. In such cases, increasing accuracy may not be a primary concern. However, scaling up may still be an issue. For example, the ability to learn small disjuncts well often is of interest to scientists and business analysts, because small disjuncts often capture special cases that were unknown previously (the analysts often know the common cases). As with classifier learning, in order not to be swamped with spurious small disjuncts it is essential for a data set to be large enough to contain enough instances of each special case from which to generalize with confidence (Provost and Aronis 1996).

It should be clear that scaling up to very large data sets implies, in part, that fast learning algorithms must be developed. There are, of course, other motivations for fast learners. For example, interactive induction (Buntine 1991), in which an inductive learner and a human analyst interact in real time, requires very fast

learning algorithms in order to be practicable. Wrapper approaches, which for a particular problem and algorithm iteratively search for feature subsets or good parameter settings (Kohavi and Sommerfield 1995; Kohavi 1996; Provost 1992; Provost and Buchanan 1995), also require very fast learners because such systems run the learning algorithms multiple times, evaluating them under different conditions. Furthermore, in a wrapper approach, each evaluation may involve multiple runs to produce performance statistics (e.g., with cross-validation). Experimenting with many learning biases also requires a large data set to avoid overfitting due to bias selection (DesJardins and Gordon 1995). As a final example, the popular practice of learning multiple models and combining their predictions (Dietterich 1997) also multiplies the run time.

3. How large is "very large"?

The KDD community includes researchers and practitioners from diverse backgrounds, including machine learning, statistics, and databases. Researchers in machine learning are accustomed to dealing with flat files and algorithms that run in minutes or seconds on a desktop platform. For them, 100,000 instances with a couple dozen features is the beginning of the range of "very large" data sets.

The database community deals with gigabyte databases. "Very large" to a database practitioner usually means databases (warehouses) of 100 gigabytes or larger (Agrawal and Srikant 1994). Of course, it is unlikely that all the data in a data warehouse would be mined simultaneously. In practice, data preprocessing techniques often reduce by orders of magnitude the size of the data set presented to algorithms. However, the need for data reduction as a preprocess may be more a restriction on our view of learning algorithms than a fundamental restriction on data mining. Nevertheless, because this survey concentrates on the algorithms for mining the data, we will take an algorithmic perspective on the issue of "very large." For most published work on algorithms, one million examples is considered to be a very large data set (100Mbyte-1Gbyte range). This agrees with Huber's assessment from a statistical perspective, given in his KDD-97 invited talk (Huber 1997): "Somewhere around data sizes of 100 megabytes or so, qualitatively new, very serious scaling problems begin to arise, both on the human and on the algorithmic side" (p. 306).

4. What is "scaling up"?

For all its theoretical considerations, the issue of scaling up is inherently pragmatic. For scaling up learning algorithms, the issue is not as much one of speeding up a slow algorithm as one of turning an impracticable algorithm into a practicable one. The crucial issue is seldom "how fast" you can run on a certain problem, but instead "how large" a problem can you (feasibly) deal with. From the point of view of complexity analyses, for most scaling problems the limiting factor of the data set has been the number of examples. A large number of examples introduces potential problems with both time and space complexity. For time complexity,

the appropriate algorithmic question is: what is the growth rate of the algorithm's run time as the number of examples increases? Also important are the number of attributes describing each example and the number of values for each attribute.

As may be expected, time-complexity analyses do not tell the whole story. As the number of instances grows, certain space constraints become critical—most importantly, the absolute size of the main memory with which the computing platform is equipped. Except as described below, almost all existing implementations of learning algorithms operate with the training set entirely in main memory. Furthermore, many algorithms achieve reduced run-time complexity with bookkeeping that increases the space used. No matter what the run-time computational complexity of the algorithm, if exceeding the main memory limitation leads to virtual memory thrashing, the algorithm will not scale well (Provost and Hennessy 1996).

Finally, the goal of the learning must be considered. Evaluating the effectiveness of a scaling technique becomes complicated if a degradation in the quality of the learning is permitted. The vast majority of work on learning algorithms uses classification accuracy as the metric by which different algorithms are compared. In such cases, we are most interested in methods that scale up without a substantial decrease in accuracy. For problems that require mining regularities from the data for purposes other than classification, metrics should be devised by which effectiveness can be measured (and compared) as the system scales up (Srikant and Agrawal 1996).

5. A high-level characterization of methods for scaling up

Many diverse techniques have been proposed and implemented for scaling up inductive algorithms. The similarities among the techniques become apparent when they are categorized into three main approaches. In most cases, techniques from separate categories are independent and can be applied simultaneously. The three main approaches are:

- design a fast algorithm
- partition the data
- use a relational representation

The **fast algorithm** approach includes a wide variety of algorithm design techniques for reducing the asymptotic complexity, for optimizing the search and representation, for finding approximate solutions instead of exact solutions, or for taking advantage of the task's inherent parallelism.

The **data partitioning** approach involves breaking the data set up into subsets, learning from one or more of the subsets, and possibly combining the results. Data partitioning is useful to avoid the thrashing by memory management systems that occurs when algorithms try to process huge data sets in main memory. Also, if a learning algorithm's time complexity is worse than linear in the number of examples, processing small, fixed-size data subsets sequentially can make it linear, with the

constant term dependent on the size of the subsets (Domingos 1996a). In either case, it may be possible to use a system of distributed processors to mine the subsets concurrently. An approach orthogonal to the selection of example subsets is to select subsets of relevant features upon which to focus attention.

The **relational representation** approach addresses data that cannot feasibly be treated as a flat file, including any large relational database, as well as other large relational structures such as those used for knowledge representation in artificial intelligence. In the literature, such techniques have been framed either as learning in first-order logic, as learning from relational databases (without flattening them out), or as flat-file learning augmented with relational background knowledge.

Figure 1 summarizes the general methods that make up each of the three broad approaches to scaling up inductive algorithms. We discuss the constituent methods in detail in the next section.

Scaling Methods	
Main Approach	*General Method*
Fast algorithm	Restricted model space
	Powerful search heuristics
	Algorithm/programming optimizations
	Parallelization
Data partitioning	Select an instance subset
	Select a feature subset
	Process subsets sequentially
	Process subsets concurrently
Relational representations	Represent data relationally
	Integrate data mining with database management

Figure 1. Methods for scaling up inductive algorithms

6. A comparison of methods

Grouping methods into these three broad categories illustrates that certain techniques not previously considered to be related are in fact very similar. We will now summarize the methods within each category in order to highlight their similarities, their differences, their strengths, and their weaknesses.

The survey is not exhaustive. It is representative of the current state of the art, and places recent work in the context of a number of historically important examples that have had lasting impact. We begin with the design of fast learning algorithms.

6.1. Fast algorithms

The most straightforward approach to scaling up inductive learning is to produce more efficient algorithms or to increase the efficiency of existing algorithms. Of

Fast Algorithms	
General Method	*Example Technique*
Restricted model space	decision stump, two-level tree
Powerful search heuristics	greedy, divide & conquer
	avoid decision-tree post-processing
	search-space pruning
Algorithm/programming optimizations	efficient data structures
	dynamic search-space restructuring
	bookkeeping strategies
	optimized computing infrastructure
Parallelization	search-space parallelization
	parallel matching

Figure 2. Methods for designing fast inductive algorithms

course, for very large problems, even a fast linear-time algorithm may not be *sufficient* for practicable data mining. However, it is usually the case that for very large problems, even with the use of sampling, feature selection, and relational representations, a fast algorithm is still *necessary*. Just how fast inductive algorithms must be depends, of course, on the problem. Most work on fast algorithms strives for near-linear time complexity in the number of examples (e). This is in line with Huber's observation that $O(e^{3/2})$ is the maximum tolerable time complexity (Huber 1997).

For a discussion of learning-algorithm design, it is necessary to choose an analytical framework that facilitates discussing different algorithms. We will adopt the commonly used "induction as search" framework, within which data mining is framed as a search through a space of models for a model that performs well with respect to some criteria (Simon and Lea 1973) (Mitchell 1982). The use of this framework naturally partitions fast-algorithm design into two categories of methods (see Figure 2). First, one can restrict the space of models to be searched, based on the straightforward (and sometimes untrue) principle that a small model space will be faster to search than a large one. Second, for a large model space, one can develop powerful search heuristics, where "powerful" means that the heuristics are efficient, yet they often find competitive models. Next we will discuss some particular examples of each of these methods that have proven to be both effective and efficient. We then discuss several algorithm/program optimizations that have been detailed in the literature. Finally, we discuss approaches to the use of parallelism to speed up inductive algorithms.

6.1.1. Restricted model space One approach to designing a fast learning algorithm is to restrict it to search an "easy" model space. The clearest examples of effective restricted model-space learners are the long-lived and still viable linear-discriminant methods for learning classifiers (Duda and Hart 1973). Complex machine learning methods typically are justified by noting that they can capture com-

plex, non-linear relationships from data. Nevertheless, research on both symbolic and neural learning has shown that simple models perform well on many problems. For example, Shavlik et al. (1991) show that with certain qualifications, "the accuracy of the perceptron is hardly distinguishable from the more complicated learning algorithms." *One-level decision trees*, also known as *decision stumps*, are simple mappings from the values of one attribute to class labels. Decision stumps also have been shown to achieve moderately high accuracy on many common benchmark databases (Iba and Langley 1992) (Holte 1993). Because of their restricted model spaces, these simple learning algorithms can be trained very quickly. Haussler (1988) relates "easy" model spaces both to the machine-learning notion of inductive bias (Mitchell 1980), and to Valiant's theoretical framework (Valiant 1984). The gist is that a model space can be easy to search either because it is simply small (as with decision stumps) or because some special structure weakens its power of expression (as with linear discriminants).

The tone of research on inductive algorithms changed markedly with the acceptance (or re-acceptance) of restricted hypothesis space algorithms as legitimate competitors. One reason for this change is that these simple, fast algorithms facilitate straightforward, competitive benchmarking. More importantly, from a scaling-up perspective, the competitive run-time performance of the simple classifiers makes it more difficult to justify very complex algorithms. Interest has developed in other simple classifiers that also perform well. For example, in subsequent work Auer et al. (1995) introduce a theoretically founded algorithm, called T2, for learning two-level decision trees. They show that for eight out of fifteen data sets, T2 produces two-level trees which rival or surpass the de facto standard C4.5 (see below) (Quinlan 1993). Interestingly, in practice C4.5 is considerably faster than T2 (Lim, Loh, and Shih 1999).

6.1.2. Powerful Search Heuristics Certainly, in some domains there is leverage to be gained by searching for more complex models. The size and structure of the space of models, the size of the sample necessary to learn well, and the computational complexity of algorithms that search the space are intimately related. Typically, searching for more complex models is harder. However, as Haussler points out, "by using a larger hypothesis space than is strictly necessary, it may be computationally easier to find a consistent hypothesis . . . On the other hand, by using a larger hypothesis space . . . (more) examples will be required" (Haussler 1988).

Consider the (very large) space of formulae in disjunctive normal form (DNF). Many inductive algorithms designed for efficiency, including those that learn decision trees, decision lists, and rule sets, search the space of some variant of DNF formulae (Pagallo and Haussler 1990). Because the model space is vast and there is little structure to facilitate search, powerful heuristics are necessary for navigating it efficiently (Haussler 1988).

For a vast model space, it is unusual for learning algorithms to search the space directly (i.e., by generating many alternative models and choosing one). In most cases, a single model is built up by evaluating its components. For example, by evaluating individual conjunctions, a DNF class description can be built. However,

even the space of conjunctions can be infeasibly large, especially when learning from a large data set, because the data set is used in the evaluation of the individual conjunctions. In practice, to scale to large data sets the run-time complexity of the learning algorithm must be close to linear in the number of examples.

Algorithm designers have had much success with greedy, divide-and-conquer approaches to building class descriptions. We chose decision-tree learners (made popular by ID3 (Quinlan 1986) and CART (Breiman, Friedman, Olshen, and Stone 1984)) for this survey, because they are relatively fast and typically they produce competitive classifiers. In fact, the decision tree generator C4.5 (Quinlan 1993), a successor to ID3, has become a de facto standard for comparison in machine learning research, because it produces good classifiers quickly. For non-numeric data sets, the growth of the run time of ID3 (and C4.5) is linear in the number of examples. Specifically, its asymptotic time complexity is $O(ea^2)$ (Utgoff 1989), where e is the number of examples in the training set and a is the number of attributes. However, since numeric data typically require repetitive sorting, their inclusion adds a $\log e$ factor at each node.

The practical run-time complexity of C4.5 has been determined empirically to be worse than $O(e^2)$ on some data sets (Catlett 1991a). One possible explanation is based on the observation of Oates and Jensen (1998) that the size of C4.5 trees increases linearly with the number of examples (even after accuracy stabilizes). One of the factors of a in C4.5's run-time complexity corresponds to the tree depth, which can not be larger than the number of attributes. Tree depth is related (logarithmically) to tree size, and thereby to the number of examples. For practical analyses during which tree size is still growing linearly with e, this adds yet another $\log e$ factor to the run-time complexity. Other empirical determinations on large data sets have established C4.5's practical time complexity to be substantially better than quadratic.[2]

The decision trees built by this greedy heuristic have been criticized for their lack of comprehensibility; in many situations rule sets are desired instead because of their modularity and increased comprehensibility (Catlett 1991a). The most common technique for producing high-accuracy rule sets, known as *reduced-error pruning*, is to grow rules via one algorithm or another, and then prune the rules in order to increase accuracy (Quinlan 1987). Unfortunately, reduced-error pruning systems generally do not scale well. For example, the rule-learning variant of C4.5, C4.5rules, has been reported sometimes to require $O(e^3)$ time (Cohen 1995) (Domingos 1996b). Some algorithms effective at finding high-accuracy rule sets have $O(e^4)$ time complexity in noisy domains (Cohen 1993). Kufrin (1997) describes speeding up C4.5rules with parallel processing, which we discuss later.

Fürnkranz and Widmer (1994) show, with their *incremental reduced error pruning (IREP)* algorithm, that significant speedups can be obtained by pruning each rule as it is learned and then applying a separate-and-conquer strategy based on the *pruned* rule. Their formal analysis predicts a computational complexity of $O(e \log^2 e)$, which has been verified empirically by Cohen (1995). Unfortunately, the accuracy of the class descriptions learned by IREP often is lower than the accuracy of those learned with the slower C4.5rules. Cohen details several mod-

ifications to improve IREP's accuracy, including different rule-evaluation criteria, different stopping criteria, and a post-processing optimization, producing the algorithm RIPPER. He shows that RIPPER is competitive with C4.5rules in terms of error rate and that it maintains the $O(e \log^2 e)$ time complexity of IREP. (Cohen also estimates the time complexity empirically.)

A different style of rule learning can be traced back to the search-based data mining program MetaDENDRAL (Buchanan, Smith, White, Gritter, Feigenbaum, Lederberg, and Djerassi 1976) (Buchanan and Feigenbaum 1978). Examples of MetaDENDRAL-style rule learning include the Brute programs (Riddle, Segal, and Etzioni 1994; Segal and Etzioni 1994a), PVM (Weiss, Galen, and Tadepalli 1990), ITRULE (Smyth and Goodman 1992), the RL programs (Clearwater and Provost 1990; Provost and Buchanan 1995; Fawcett and Provost 1997), SE-trees (Rymon 1993), and even Schlimmer's determination-learning algorithm (Schlimmer 1993). These programs view rule learning as an explicit search of the rule space rooted at the rule with no conditions in the antecedent, with rules becoming more specific (by adding conditions) as they get further from the root (described in detail by Webb (1995)). To allow for massive searches of very large rule spaces, the search space is reduced with depth-bounding and various forms of pruning.

6.1.3. Algorithm/programming optimizations Algorithm optimization by using efficient data structures (e.g., bit vectors, hash tables, binary search trees) and clever programming techniques is good engineering practice, complements the other methods of scaling up, and in practice often can give very large speedups. These optimizations differ from "powerful search heuristics" in that they concentrate on eliminating redundant or unnecessary computations—the models induced will not be affected. Some such optimizations are remarkable enough to have appeared in published work.

Some of the rule-space pruning techniques used in MetaDENDRAL-style rule learners can be guaranteed not to discard good rules (Clearwater and Provost 1990; Segal and Etzioni 1994b; Webb 1995). Webb (1995) takes this idea even further, introducing techniques for dynamic search-space restructuring to maximize the amount of search space removed with each pruning. He shows that it is possible to search exhaustively for the rule that optimizes the Laplace accuracy estimate for (at the time) every categorical attribute-value benchmark data set in the UCI repository (Merz and Murphy 1997). BruteDL's search algorithm was optimized carefully (Segal and Etzioni 1994b). Segal and Etzioni report that with 500 training examples, BruteDL can process 100,000 rules per second, when running on a SPARC-10 processor. They also note that significant additional speedups are not expected because BruteDL's speed is within an order of magnitude of the machine's clock rate.

Domingos (1996b) proposes to improve rule-learning efficiency by not growing each rule to its full length in the first place. He points out that the commonly used separate-and-conquer methods induce rules by evaluating each rule by itself, without regard to the effect of other rules. To avoid superfluous growth, as each rule is grown Domingos's CWS algorithm evaluates it in the context of the currently

held rule set. However, a straightforward recomputation of the accuracy of the whole rule set for each rule modification is very expensive. Domingos details an optimized procedure that carefully eliminates redundant computation, yielding a procedure with run-time complexity $O(eavcs)$, where v is the average number of values, c is the number of classes, and s is the total number of antecedents in the resultant rule set. In principle, s can be $O(e)$, but Domingos verifies empirically that in practice s is independent of e.

Another notable optimization is the pre-sorting procedure used by the decision-tree learner SLIQ (Mehta, Agrawal, and Rissanen 1996). As described above, repetitive sorting reduces the efficiency of decision-tree learners when dealing with numeric attributes. SLIQ sorts the training data just once for each numeric attribute at the beginning of tree growth.

As mentioned above, most inductive algorithms load all data into main memory; therefore if a data set is too large, either the algorithm will not run, or virtual memory thrashing will render it useless. An alternative approach is never to load all the data into memory, instead accessing them on secondary storage as needed. Since secondary storage devices typically do not provide random access to data, algorithms must be designed to process data via sequential scans—and as few as possible. SLIQ takes advantage of the need for only a single pass through the data for each level of a decision tree, if the tree is grown breadth first. However, one of the data structures that SLIQ uses during its pre-sorting step has size proportional to the number of input records, and therefore the size of this memory-resident structure becomes the limiting factor for this approach. These limitations are addressed in the SPRINT system (Shafer, Agrawal, and Mehta 1996), which does not use any monolithic, memory-resident data structures.

SPRINT is regarded widely as the reigning standard in scalable decision-tree building (Dietterich 1997). However, with this honor comes the increased scrutiny that leads to further advances. SPRINT maintains augmented vertical partitions of the data, copied into auxilliary data structures (*attribute lists*). Because of this, it has been criticized for several reasons. For example, maintaining the data structures can be costly, including a potential tripling of the size of the database (Gehrke, Ramakrishnan, and Ganti 1998) and an associated significant increase in scan cost (Graefe, Fayyad, and Chaudhuri 1998).

This brings us to perhaps the most remarkable data mining optimization, a simple bookkeeping technique that has been pointed out recently by several independent research groups. The main insight is that matching hypotheses against the data is not necessary: for most of the processing, statistics from which the results of matching can be inferred are sufficient (cf. learning from statistical queries (Kearns 1993)). Separating the generation of the sufficient statistics from their use in the evaluation of hypotheses allows each to be treated separately—first using the data to populate the statistics data structure and then operating only on the data structure—which affords both optimized use of memory and improved run-time complexity.

More specifically, for most of the critical data mining operations, such as choosing nodes when constructing decision trees, one must tally for all the examples (at a

particular point in the search) the class labels associated with the different values of each attribute. A straightforward data structure to store such statistics is a contingency table of example counts for each attribute, indexed by attribute-value and class. For av attribute values and e examples, as long as $av \ll e$ the combined size of the sufficient-statistics data structures is much smaller than the size of the data set itself. John and Lent (1997) point out that such data structures are returned by SQL `GROUP BY` queries.

By using such techniques, inductive algorithms must pass through the example set only once per node expansion—indeed, an algorithm must pass through the example set only once per level in separate-and-conquer decision-tree learning—rather than once per attribute-value pair. Tremendous run-time efficiencies can be achieved when attributes have large value sets (Aronis and Provost 1997). Even fast contemporary rule-space search algorithms (Segal and Etzioni 1994b) (Domingos 1996b) generate a conjunct for each attribute-value pair, and match each against the example set to compute statistics. Thus, the run-time complexity depends on v, the average number of values of an attribute. Domingos reports a time complexity of $O(eav)$ for the CWS algorithm (discussed above). These bookkeeping techniques can reduce rule-learning complexity to $O(ea)$ (Aronis and Provost 1997). Aronis and Provost go on to show that similar techniques can be used to speed-up learning with hierarchically structured data (Almuallim, Akiba, and Kaneda 1995), to which we will return when we discuss relational representations.

Among the various decision tree programs, C4.5 has been shown to be comparably fast (Lim, Loh, and Shih 1999)—remarkably so considering the programs' similarity. An analysis of its code shows that when evaluating node splits, C4.5 first builds a sufficient-statistics contingency table, and then uses it to decide on the best split. Kufrin (1997) notes that preliminary experiments with additional optimizations to C4.5 show substantial additional speedups (on a single processor), and indeed C4.5's successor C5 has been observed to be substantially faster than its predecessor (Harris-Jones and Haines 1997).

Gehrke, Ramakrishnan and Ganti (1998) provide a thorough treatment of the use of sufficient statistics to build decision trees when the size of the data set exceeds main memory. They discuss the options available if even the sufficient-statistics data structure is too big for main memory, and present the RAINFOREST family of algorithms. Moore and Lee (Moore and Lee 1998) present specialized data structures (*ADtrees*) designed to take best advantage of sufficient statistics for speeding up inductive algorithms. According to Moore and Lee, it is doubtful that the cost of building ADtrees will be worthwhile for individual runs of fast learning algorithms. However, they are obvious candidates for more computationally intensive algorithms, and for systems that run algorithms many times on the same data (such as interactive systems, wrapper systems, and multiple-models systems, discussed above).

Not only can the inductive program itself be optimized, the computing infrastructure can too. Data mining programs read in large amounts of data; in fact, reading in the data can take longer than mining it (Provost and Aronis 1996). Parallel I/O

systems and optimized data layout can make a considerable difference (Grossman and Bailey 1998).

6.1.4. Parallelization The process of inductive learning is decomposable at two levels, illustrated by the two main methods for parallel learning, namely, *search-space parallelization* and *parallel matching*.

As we discuss above, inductive learning can be viewed as the search of a very large space. In search-space parallelization, the search space is decomposed such that different processors search different portions of the space in parallel (Cook and Holder 1990), similar to the parallelization of other forms of heuristic search (Kumar and Rao 1987) (Rao and Kumar 1987). Load balancing and interprocess communication add additional complexity and overhead. In general, this type of parallelization does not address the problem of very large data sets, because each processor will have to deal with all the data (or subsample, which we discuss below). However, recently Zaki et al. (1999) have had success with search-space parallelization of a decision-tree algorithm; by taking advantage of a shared memory multiprocessor, they are able to avoid replicating or communicating the entire data set among the processors. Using shared memory also allows the development of effective load balancing techniques. Galal, Cook, and Holder also have had success recently using search space parallelism to scale up other data mining algorithms (Galal, Cook, and Holder 1999).

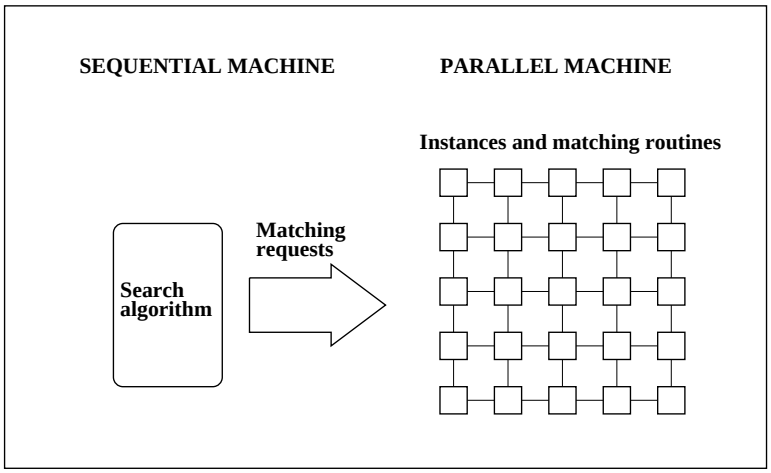

Figure 3. Parallel matching

Parallel learning has been more successful when a lower-level decomposition is used. The parallel matching approach is based on the observation that search for

inductive learning is different from most other search problems. In inductive learning the cost of evaluating a node is very high, but also highly decomposable. Nodes in the search space (e.g., partial rules or decision tree branches) are hypothesized and each is matched against many examples to gather statistics. In the parallel matching approach, depicted in Figure 3, this compute-intensive matching process is farmed out by migrating the example set and matching routines to a parallel machine, while the main learning algorithm (the *master*) may run on a sequential front end.

Parallel matching has been used by Lathrop et al. (1990), by Provost and Aronis (1996), and in the parallelization of the SPRINT algorithm (Shafer, Agrawal, and Mehta 1996). The former two efforts use a straightforward parallelization of the matching routines. In the SPRINT work, each processor builds a sublist of each attribute list, and for each decision-tree node sends the master a portion of the statistics needed to determine the best split. Impressive speedups are reported for parallel matching: less than a minute to learn from one million examples on a CM-2 Connection Machine with 8192 bit-slice processors (Provost and Aronis 1996); 400 seconds to learn from 1.6 million examples on an IBM SP2 with 16 processors (Shafer, Agrawal, and Mehta 1996).[3] Kufrin (1997) uses parallelization to speed up C4.5's transformation of decision trees to rules (C4.5rules), using parallel matching for two phases of rule-set postprocessing, and dividing up the rule set itself for a third. He also reports impressive speedups (efficiencies averaging more than 0.9 for four learning tasks and up to eight processors). The drawback to the parallel matching approach is that it is not always easy to obtain access to massively parallel hardware.

Zaki (1998) points out that shared-memory multiprocessor (SMP) systems are much more common, and presents a parallel matching approach to the design of an SMP version of SPRINT. Instead of distributing the instances, they process (in parallel) vertical partitions corresponding to SPRINT's attribute lists. The attribute lists are divided equally among the processors, which return the matching statistics to the master.

Access to parallel hardware may not be a concern if the data are already resident in a data warehouse with a parallel infrastructure. Freitas and Lavington (1996) take such an approach, making use of existing parallel database server technology. Their approach is similar to that shown in Figure 3, except the implementation-specific parallel data representation is replaced by an existing parallel database system. Also, for communication between the front- and back-end, implementation-specific matching requests are replaced with SQL queries. Commercial data mining system vendors often cite this approach when confronted with the issue of scaling, but technical details on specific vendor-supplied data mining systems are elusive. We will discuss the use of database systems and SQL queries for scaling up in more detail in Section 6.3.

A third approach to the use of parallelization is to partition the data into subsets, and then run learners concurrently on the subsets. This approach is described in the next section. Parallel data mining is treated in more detail in a recent book by Freitas and Lavington (1997).

6.2. Data partitioning

The previous section addressed the design of algorithms that are fast enough to run on very large example sets. An orthogonal approach is to partition the data, avoiding the need to run algorithms on very large data sets. Data partitioning

Data Partitioning	
General method	*Example technique*
Select an instance subset	random sampling
	duplicate compaction
	stratified sampling
	peepholing
Select a feature subset	use relevance knowledge
	use statistical indications
	use subset studies
Processing subsets sequentially	independent multi-subset learning
	sequential multi-subset learning
Processing subsets concurrently	learn multiple models, pick best
	combine class descriptions
	combine predictions
	cooperative learning

Figure 4. Data partitioning methods

techniques can be categorized based on whether they separate subsets of examples or subsets of features. Figure 4 illustrates that there are several techniques for selecting a single subset from which to learn. Furthermore, multiple subsets can be chosen and they can be processed in sequence or concurrently.

Figure 5 depicts a general model showing the similarities among partitioned-data approaches. Systems using these approaches select one or more subsets $S_1, \ldots, S_n$ of the data based on a *selection procedure*. Learning algorithms $L_1, \ldots, L_n$ are run on the corresponding subsets, producing concept descriptions $C_1, \ldots, C_n$. Then the concept descriptions are processed by a *combining procedure*, which either selects from among $C_1, \ldots, C_n$ or combines them to produce a final concept description.

The systems differ in the particular procedures used for selection and combining. They also differ in the amount and style of interaction among the learning algorithms and learned concept descriptions.

6.2.1. Select a subset of the instances The most common approach for coping with the infeasibility of learning from very large data sets is to select a single sample from the large data set. Referring to Figure 5, sampling is a degenerate form of a partitioned-data system: only a single subset is chosen. The differences between sampling techniques involve the particular selection procedure used.

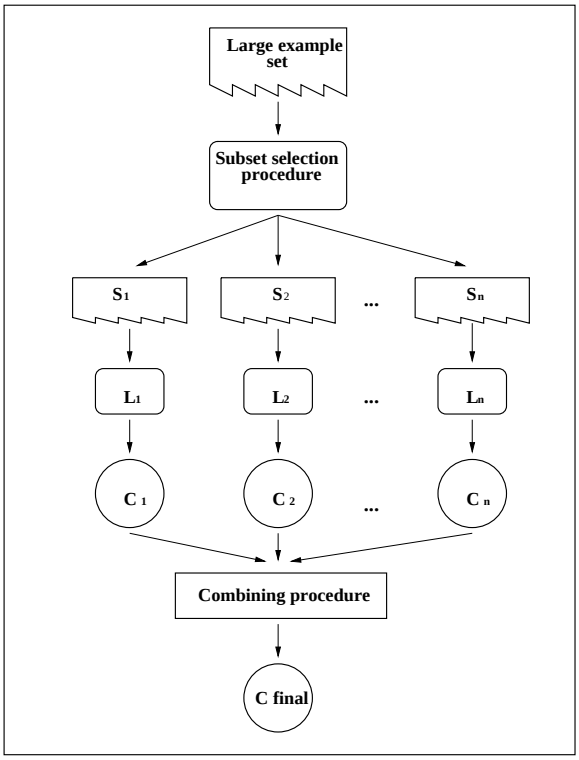

Figure 5. Learning using data partitioning

Catlett (1991a) (1991b) studied a variety of procedures for sampling instances from a large data set and compared empirically the results of using the different techniques. In particular, he studied the following.

- *Random sampling* selects a subset of examples randomly.
- *Duplicate compaction* removes duplicated instances from the database. The computational effort is proportional to the degree of completeness desired.
- *Stratified sampling* is applicable when the class values are not uniformly distributed in the training sets. Examples of the minority class(es) are selected with a greater frequency, in order to even out the distribution.

Some readers may have difficulty accepting sampling as a method for scaling up to large data sets; after all, sampling reduces the size of the data set processed. However, it is important to examine the function the algorithm is performing. Consider classifier induction algorithms. They take data sets as input and produce classification models as output. As discussed above, the question of scalability asks whether the algorithm can process large data sets efficiently, *while building from them the*

best possible models. Therefore, if (for example) using sampling produces models with lower accuracy than otherwise, its usefulness for scaling up is in question. On the other hand, if using sampling produces equivalent (or better) models, then sampling *is* an effective scaling mechanism. Sampling is well accepted by the statistics community, who observe that "a powerful computationally intense procedure operating on a subsample of the data may in fact provide superior accuracy than a less sophisticated one using the entire data base." (Friedman 1997).

Thus, for anyone wanting to mine a large data set, an important question is: must I process the whole thing? Or will sampling be effective? The answer is: it depends on the data set. Just because you have a massive data set does not imply necessarily that you must mine it all. In practice, as the amount of data grows, the rate of increase in accuracy slows, forming the familiar learning curve. Whether sampling will be effective depends on how dramatically the rate of increase slows.

It is difficult to determine in general how small a data set may be, because it depends on factors not known a priori. For example, as we discuss above, it depends on the minimum size of the special cases that a learner must discover in order to model the phenomenon effectively. However, if one is willing to bias a learner (explicitly or implicitly) against learning very small special cases, then recent work on determining sufficient sample sizes for similar data mining problems provides relevant results. For example, Toivonen (1996) and Zaki et al. (1997) discuss the determination of sufficient sample sizes for finding association rules that are no smaller than a predefined size, based on tolerances on the probability of error and the size of the error. A different view of sufficient sample size, that of *sample complexity*, is provided by Valiant's theoretical framework (Valiant 1984) (Haussler 1988), which for a given hypothesis space allows the calculation of the number of examples sufficient for learning with high probability a good approximation to the "true concept," if one exists in the hypothesis space.

Published work provides differing views of how often real-world classifier learning curves level off before massive data sets are needed. Catlett's work shows that learning from subsets of data decreases accuracy. Despite the advantages of certain sampling strategies, viz., speed-ups and improving the accuracy of the classifier over random sampling in noise-free domains, Catlett concludes that they are not a solution to the general problem of scaling up to very large data sets (Catlett 1991b). However, it should be noted that at the time of Catlett's study, "massive" data sets were much smaller than they are today, and processing times much longer. In fact, Catlett's conclusions were based on data sets which had fewer than 100,000 instances. Every data set in his study would fit in the main memory of a modern desktop PC. The study of KDD would benefit from a replication of Catlett's analyses, taking into consideration the current state of computing, to see if his conclusions stand after a decade of technological improvements.

In a more recent study, Harris-Jones and Haines (1997) analyze the relationship between data set size and accuracy for two large business data sets (up to 300,000 instances), by estimating learning curves empirically. They found that while some algorithms level off quite early, in some cases algorithms (decision-tree learner C4.5 and its successor C5, in particular) continue to show accuracy increases across the

entire range of data set sizes. However, the improvements in accuracy at the upper size limit have become quite small, and it is difficult to conclude that they would continue with another order of magnitude increase in data set size. The authors note that a more important question is whether the benefit of further improvements is worth the associated cost (Haines 1998).

Neither these results nor Catlett's provide ample justification for mining data outside of main memory. The data set sizes are not "massive" by modern standards. They can be processed in the main memory of a PC.[4] Our field would benefit from a few prominent examples of the need to scale up beyond reasonable main memory limits.

Oates and Jensen (1997) studied decision tree induction for nineteen data sets, and looked specifically at the number of examples necessary before the learning curves reached a plateau. They regard a plateau to have been reached when an accuracy estimate is within a certain tolerance of the maximum (specifically, one percent, in their experiments). Surprisingly, for these nineteen data sets, as well as some others (Jensen 1998), a plateau was reached after very few training examples.

Of course, when there exists a massive volume of data, some sampling may be necessary, whether or not it decreases accuracy. For example, in a famous application of inductive learning, Fayyad et al. (1993) used sampling techniques (among others) to reduce more than three terabytes of raw data. Therefore, it is important to consider whether it is *possible* to sample efficiently. Consider that if it is necessary to scan the entire data set in order to produce a random sample, much of the advantage of sampling will be lost. We will return to this point later, when we address database support for scaling up data mining.

Heretofore we have discussed what may be called "passive" sampling, for which the size and content of the training set are determined before induction begins. Inductive algorithms can also sample actively, based on intermediate results, as induction progresses. The notion of induction as the simultaneous search of two spaces, the space of possible concepts and the space of possible instances, was introduced by Simon and Lea (1973) and was elaborated by Provost and Buchanan (Provost 1992; Provost and Buchanan 1995).

For scaling up, Catlett (1991a) (1991b) studied the active, tactical use of sampling to reduce complexity as learning algorithms process large data sets. In particular, the search for good split values for numeric attributes dominates decision-tree inducers' computation, because the values must be sorted. Catlett found that by looking at subsets of examples (called *peepholes*) when searching for good split values for numeric attributes, the run time of decision-tree learners can be reduced substantially without sacrificing accuracy. In subsequent work, Musick et al. (1993) introduced information-theoretic measures to assess the risk of using peepholes for the evaluation of attributes in decision-tree induction. In particular, they show how to determine whether the choice of attribute can be made confidently within a given error tolerance, and, if not, how to determine how large a peephole is required to do so. In Section 7 we discuss a similar technique for determining the minimum number of training examples sufficient for satisfactory learning, namely,

progressively sampling larger subsets until model performance no longer improves (John and Langley 1996; Frey and Fisher 1999; Provost, Jensen, and Oates 1999).

6.2.2. Select a subset of the features So far, our discussion of data partitioning has focused on selecting a subset of the examples. Let us now turn to the problem of selecting a subset of features. It is important to consider the symmetry with selecting instance subsets: one method selects rows of a data table; the other selects columns. The space tradeoff is symmetric because the amount of space needed to store the table is the product of the number of rows and the number of columns. From the point of view of scaling up, the same observations apply to both. Operating on a subset reduces induction time and space requirements. Multiple subsets can be operated on independently. The results of induction with one subset may help to determine the next, and learned models can be built from components learned from different subsets. This symmetry is discussed in more detail by Provost and Buchanan (Provost 1992; Provost and Buchanan 1995).

A full treatment of feature selection is beyond the scope of this paper. Data engineering is less visible in the literature than algorithmic issues of induction, but feature selection is one data-engineering issue that has received more than just a superficial treatment (Devijver and Kittler 1982) (Miller 1990) (Wettschereck, Aha, and Mohri 1997). However, the majority of the existing work on feature selection has *not* focused directly on scaling. Instead it has focused on the phenomenon that reducing the size of the feature set, when done well, often can increase the accuracy of the resultant class description.

For the purposes of this survey, it is important to clarify these two closely related reasons for selecting feature subsets. As discussed in Section 2, as the size of the feature set grows, so do the chances that an induction program will overfit the training set—especially with a small training set. Thus, if one can select a good subset of the features, one often can increase accuracy. Ironically, as the number of examples is increased (and thereby feature selection becomes less necessary from a data-fitting perspective), feature selection becomes more necessary from a run-time perspective. As described above, the run time of inductive algorithms grows with the number of attributes, often at a rate worse than linear. Therefore, selecting a subset of the features may be important for practical algorithm application, independent of whether the selection increases the accuracy.

Selecting a subset of features is such a common method for reducing problem size that it often is neglected in discussions of scaling. When setting up a learning problem, only a small set of the possibly relevant variables are chosen for representation. Sometimes this restriction is based on the data collection apparatus, but often it is based on knowledge of relevance. Interaction with domain experts can indicate that it is unlikely for certain variables to be useful, so they are not included. Moreover, for each variable describing a problem, there are often auxiliary databases that provide related information. For example, a zip-code field might link to a massive database of demographic information. In practice, additional fields are added only if there is a reason to believe that they are relevant.

The use of prior relevance knowledge is not the only method for selecting a subset of the possible features. Another approach is to describe the problem with as many features as possible, and then to do inexpensive empirical studies to select a subset. Little has been published about using statistical indications to reduce the number of features for the purpose of scaling up, although some techniques may be viewed as too straightforward to include in publications. For example, many practitioners compute correlations of individual features to the target concept, and select a practically manageable subset of features with high correlations (Kaufman and Michalski 1996) or with high information gain (Wettschereck and Dietterich 1995).

Of course, such simple methods may miss features that are only useful in combination. Chen and Yu (1995) address this problem with a combination of instance subsetting and feature subsetting, similar to the peepholing of Catlett and of Musick et al. described above. Chen and Yu propose a two-phase method for attribute extraction to improve the efficiency of deriving classification rules in a large training data set. During the first phase, known as the *feature extraction phase*, a subset of the training data set is analyzed to identify a relevant subset of features. During the second phase, the *feature combination phase*, those extracted features are evaluated in combination, and multi-attribute predicates with strong inference power are identified (Chen, Han, and Yu 1997). The RELIEF-F algorithm (Kononenko 1994) uses experiments with randomly drawn examples and a nearest-neighbor representation to identify (even highly interdependent) relevant features (Kononenko, Simec, and Robnik-Sikonja 1997). In the next section we discuss feature selection methods that process subsets sequentially.

6.2.3. Processing Subsets Sequentially Several efforts have addressed learning from multiple subsets and combining the results. We will first consider those approaches where subsets are processed sequentially. In these cases, the differences between methods involve how the concept description learned in the previous iteration is used, and how the combining procedure operates. Unless otherwise noted, for each of the approaches described in this section and in the following, the selection procedure partitions the data set randomly into n subsets.

Figure 5 shows a general model of partitioned-data learning. More precisely, this figure shows a model of *independent multi-subset learning*, because there is no interaction between the n learning runs; the C_i are formed independently, and then combined. Fayyad et al. (1993) use a sequential independent multi-subset approach in which the L_i are decision-tree learners; the C_i are rule sets extracted from the decision trees, and the combination procedure is a greedy covering algorithm.

When multiple subsets are being processed sequentially, it is possible to take advantage of knowledge learned in one iteration to guide learning in the next iteration. Figure 6 and Figure 7 show two approaches to *sequential multi-subset learning*. *Model-guided instance selection*, shown in Figure 6, is an iterative, active sampling technique, with which class description C_i helps in determining S_{i+1}. In *incremental batch learning*, shown in Figure 7, class description C_i is taken as input to the learner and used in building C_{i+1}.

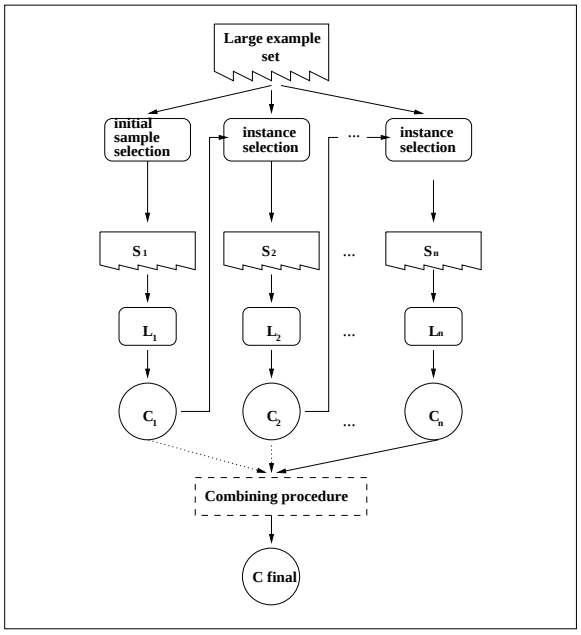

Figure 6. Sequential multi-subset learning: model-guided instance selection

Sequential multi-subset techniques have been used by several researchers to address learning from large data sets. Quinlan (1983) used a model-guided instance selection approach, called *windowing*. The selection procedure begins by choosing candidate examples either randomly or by stratification. S_{i+1}, called the *window*, is augmented by examples that C_i classifies incorrectly. The combining procedure simply chooses C_n as the final concept description. Catlett (1991b) studied windowing on several learning problems, and found the following. The effect of windowing on learning time varied from problem to problem, from a factor of three speedup to a factor of 20 slowdown. Severe slowdowns occur when the data are noisy. He concluded that windowing is a scaling solution for noise-free data sets only. If continuous attributes are present, windowing can also improve accuracy.

Incremental batch learners (Clearwater, Cheng, Hirsh, and Buchanan 1989) are hybrids of sampling and incremental learning. Class description C_i is given as "prior knowledge" to learning algorithm L_{i+1}, along with subset S_{i+1}. The learning algorithm uses S_{i+1} to evaluate C_i, and uses C_i as a basis for building C_{i+1}. As with windowing, the combining procedure chooses C_n as the final concept description, but with incremental batch learning the C_n is constructed across the n learning runs. Incremental batch learning approaches have been used to scale up to example sets that are too large for pure batch processing because of limits on

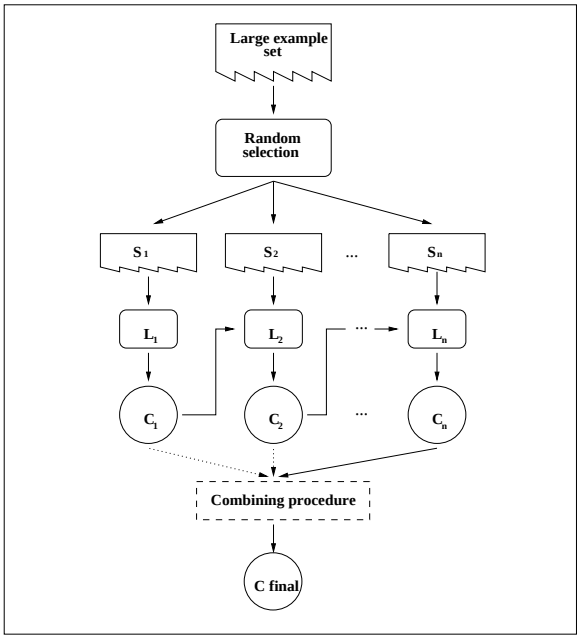

Figure 7. Sequential multi-subset learning: incremental batch learning

main memory, leading to increased accuracy over simple sampling (Provost and Buchanan 1995). Incremental batch learning offers speedups because, as discussed above, even for learners that theoretically scale up linearly in the number of examples, if the entire example set does not fit in main memory, operating system page thrashing can render the learner useless. An incremental batch learning approach was used by Domingos (1996a) to transform an algorithm whose run-time complexity is quadratic in the size of the example set to a linear algorithm. Incremental batch learning has also been called "multi-layer incremental induction" (Wu and Lo 1998).

Historically, windowing has been used with decision-tree learners and incremental batch learning has been used with rule learners. This coincidence is not accidental. Because of their modularity, rules can be evaluated individually and rule sets constructed easily from multiple learning runs; this is much more difficult with decision trees. Furthermore, separate-and-conquer rule learning internally does model-guided instance selection as induction progresses: the existing rule set is used to reduce the set of examples used in subsequent learning. Therefore, if a partial rule set is provided as input to a separate-and-conquer learner, it can (internally) restrict subsequent search to rules that are not yet covered. Fürnkranz (Fürnkranz 1998) presents a technique integrating model-guided instance selection and incremental batch learning, which he calls *integrative windowing.*

Fürnkranz also presents an insightful analysis of sequential multi-subset rule learning, pointing to a variety of other related work and providing crisp explanations of important observations. For example, windowing fails for noisy domains because a good classifier will misclassify mostly noisy examples, so subsequent windows will have increasing levels of noise, thereby decreasing subsequent learning performance. He also explains, and shows empirically, that sequential multi-subset learning improves the efficiency of rule learning more than decision-tree learning, because as with incremental batch learning, the rules need not be learned again on each iteration.

Not unexpectedly, as with single subsets, sequential multi-subset techniques may degrade classification accuracy as compared to learning from the entire data set at once. On the other hand, especially with model-guided instance selection, these techniques also may increase accuracy.

All of these approaches incrementally process instance subsets. Similarly, *feature* subsets can be processed iteratively. Sequential feature selection is not new, and is common in statistical treatments of classifier formation (Devijver and Kittler 1982). Two common methods are *sequential forward selection* and *sequential backward elimination*. However, as noted above, this work typically addresses increasing accuracy, rather than scaling up. Sequential backward elimination provides a simple illustration of the difference: the first iteration runs the inductive algorithm with all the features. However, techniques like sequential forward selection are useful either for increasing accuracy or for scaling up.

Wrapper approaches (Kohavi 1996; Kohavi and John 1997; Provost 1992; Provost and Buchanan 1995) are notable because they unify iterative example selection and iterative feature selection (and other iterative approaches). As mentioned above, wrapper approaches run an underlying inductive algorithm within different contexts, in an attempt to maximize some criteria. Wrapper approaches for feature selection fit well into the framework for data partitioning depicted in Figure 5; selection procedures select columns instead of rows. Kohavi and John (1997) use a wrapper to implement forward selection and backward elimination in order to maximize accuracy. In order to scale up past the limits of their computational platform, Provost and Buchanan (1995) implement various ad hoc feature selection strategies in the same programmable wrapper used to implement incremental batch learning.

Sequential feature selection techniques fall into the same two categories as sequential instance selection techniques. Specifically, as in Figure 6, some approaches, such as sequential forward selection, use C_i to influence the selection of S_{i+1}, in this case selecting columns instead of rows. Alternatively, as in Figure 7, other approaches use C_i in the construction of C_{i+1}. For example, combining class descriptions learned with different feature subsets has been found to be effective (Provost and Buchanan 1995).

6.2.4. Process Subsets Concurrently To further increase efficiency, partitioned-data approaches can be parallelized by distributing the subsets to multiple processors, learning concept descriptions in parallel, and then combining them. We differentiate this approach from parallel matching (described above) by the degree

of autonomy afforded the individual learners. Rather than simply parallelizing a subprocedure of an existing algorithm, and returning results to the master, these techniques are loosely coupled collections of otherwise independent algorithms. Recently this type of algorithm has been called "distributed data mining," and was the subject of a KDD-98 workshop (Kargupta and Chan 1998).

Concurrency precludes partitioned-data approaches where a prior concept description is needed as input to a subsequent learning stage, such as incremental batch learning. However, for independent multi-subset approaches, as shown in Figure 5, the C_i can be learned concurrently (even with different learning algorithms). Combining the C_i can take place as a sequential post-process, or can be parallelized (as by Kufrin (1997)). Hall et al. (1998) discuss this approach for learning decision trees. Similar to a distributed version of the approach of Fayyad et al. (1993), their system learns trees independently from partitioned data, and the trees are converted to rules. The rule sets are merged following the method described by Williams (1990), which resolves conflicts among similar rules.

Chan and Stolfo (1993) (1997) take a concurrent approach in which the L_i can be different learning algorithms, as well as separate instantiations of the same algorithm. They take an independent multi-subset approach with a key difference from the other methods: their method forms C_f as a hybrid of the C_i. Specifically, in the combining stage, instead of constructing C_f from selected pieces of the C_i, their approach saves the C_i whole, and combines the predictions using a multiple-model approach (Ali and Pazzani 1996). Domingos (1996a) also experiments with this type of combining for incremental batch learning, finding it superior to taking a simple union of the rule sets learned from the many batches.

A potential problem with creating a multiple-model hybrid is the resulting loss of comprehensibility. Prodromidis and Stolfo (1998) study several methods for evaluating, composing and pruning hybrid classifiers that reduce their size while preserving or even improving their predictive performance. A quite different approach to creating comprehensible classifiers from ensembles is taken by Craven (1996), by Domingos (1997), and by Guo and Sutiwaraphun (1998). These authors use machine-learning algorithms to induce understandable models of complex learned classification systems (Craven 1996). Specifically, they use the predictions of the ensemble as training labels, and learn from them a decision tree that models the hybrid's performance (with comparable accuracy). The resultant single tree is more understandable than the multiple-model hybrid.

Shasha and his research group have implemented PC4.5 (Li 1998; Shasha 1998), a parallel version of C4.5 (Quinlan 1993), which uses a different instantiation of the framework of Figure 5. Specifically, the selection procedure is a random partitioning of the data. Each C_i is a decision tree learned from a different subset of examples. The combining procedure evaluates each C_i on a subset of examples (disjoint from S_i), and chooses the one with the best accuracy as the final concept description.

With these partitioned-data techniques, accuracy may be degraded as compared to running a single inductive algorithm with all the data. This may be avoided if

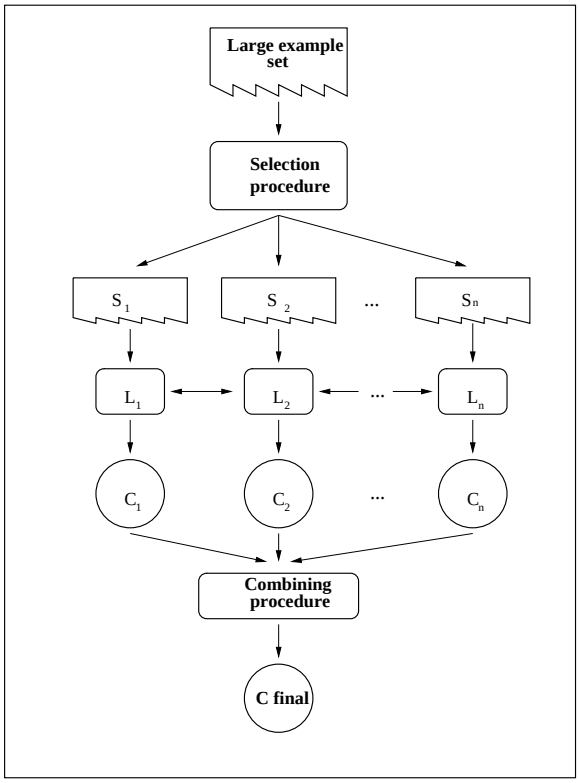

Figure 8. Cooperation among concurrent learners

the group of learners cooperates to obtain a global view of the problem, as depicted in Figure 8. The key is that the learners cooperate by sharing modules of knowledge (e.g., individual rules) that look good locally. The other learners evaluate the shared knowledge on their local data, returning or broadcasting the statistics. Provost and Hennessy (1994, 1996) take this approach for MetaDENDRAL-style rule learning, and show that it is guaranteed that each rule is considered acceptable globally if and only if it would be considered acceptable to a monolithic learner using the entire data set. Specifically, no matter how the data are partitioned, every acceptable rule will have acceptable statistics on at least one subset. Thus, over all subsets, all of the acceptable rules will be generated. The cooperation takes the form of requests (from the other learning algorithms or from a server with the entire database) for verification of statistics regarding the best discovered rules, which narrows the set of rules to only those acceptable globally. The combining procedure simply can take the union of the C_i. Because the cooperation requests are limited to rules that appear good to at least one learning program, interprocess communication is minimal. Thus, this approach has been successful scaling up to very large data sets.

A sequential version of this cooperative approach is the basis for Partition (Savasere, Omiecinski, and Navathe 1995), called one of the most efficient association-rule algorithms in terms of database operations (Toivonen 1996). Similarly, for scaling up a scientific discovery system, Galal, Cook and Holder (1999) found the concurrent/cooperative approach to be the best (among the various techniques they studied). They partition the problem and then share the best discoveries, which are then evaluated by all the processors to obtain a global perspective.

We know of no work addressing the distributed processing of feature subsets for building decision trees or rule sets, although preliminary results suggest promise for this line of inquiry. Consider a concurrent version of an independent multi-subset approach, such as that shown in Figure 7, in which many different feature subsets are selected (rather than example subsets), and the concept descriptions are subsequently combined. Results from sequential processing of feature subsets suggest that if the class description language is modular, such that useful modules (e.g., rules) can be selected from many different class descriptions, then it is possible to create an accurate class description without ever running the learner with a suitable subset of features (Provost and Buchanan 1995). Kargupta et al. (1998) consider distributed processing of feature subsets for a basis-function concept representation. We now turn to the third general approach to scaling up: using a relational representation.

6.3. Relational representations

Most existing inductive learning programs were not designed to handle very large data sets. In particular, the majority were designed under the assumption that the data set would be represented as a single, memory-resident table. Unfortunately, producing flat files from real-world, multitable, relational databases is fraught with problems. The flattening-out process can be quite time consuming, substantial storage space is needed, and keeping the flat files around leads to the problems that relational databases are designed to avoid (e.g., update and delete anomalies). Indeed, flattening may create, from otherwise manageable databases, data sets that can no longer fit in main memory. As an example, consider a database with only three tables: a customer table containing one million customers with twenty fields, including address and product preference; a state table containing fifty states with eighty fields of information on each state; and a product table containing ten products with four hundred fields of information on each product. Furthermore, let us assume that the average size of a field is five bytes. Even in this vastly oversimplified example, flattening out a 100Mbyte database results in a 2.5Gbyte flat file.[5] Flattening often demands choosing a subset of all the attributes that could be used to describe the data, which places an inflexible restriction on the unexpected discoveries that a KDD system may make.

In summary, since mining smaller data sets is typically faster, especially when they can fit in main memory, the ability of relational representations to compress data is critical. Furthermore, flattening extremely large data sets simply is not

Relational representation	
General methods	*Example Technique*
Represent data relationally	Hierarchical structures
	Multitable databases
	Knowledge structures
	Inductive Logic Programming (ILP)
Integrate data mining with database management	Access via SQL queries
	Push computation into DBMS
	Utilize parallel database engine
	Mine distributed databases

Figure 9. Methods using relational representations

feasible. In either case, we need to be able to mine relationally represented data, efficiently.

We now describe how relationally represented data can be mined directly, which scales up either because the representation is more efficient, or because the data are stored on a fast database machine.[6] We have divided the collection of methods in two, as depicted in Figure 9. First, we discuss the general issue of mining relationally represented data—regardless of how they are stored. Then we discuss mining when even the compact, relationally represented data set does not fit in main memory. In the latter case, integrating data mining with database management systems (DBMSs) is key. We treat data mining/DBMS integration as a separate approach because a unique set of issues applies. Finally, we close this section with a discussion of mining distributed databases, noting that not only does it combine two orthogonal scaling techniques, it also may be necessary because of issues such as privacy.

6.3.1. Mining relational data We have argued for the storage efficiencies afforded by relational representations, but what about mining the data once they are represented relationally? A simple form of relational data—data with hierarchical, or *tree-structured*, attributes (Almuallim, Akiba, and Kaneda 1995)—has received relatively much attention in the literature on inductive learning. The data compression afforded with tree-structured attributes can be substantial, especially with tall trees.

For example, consider geographic hierarchies ranging from fine-grained descriptors (e.g., zipcode) up to coarse-grained (e.g., country). A data miner working exclusively with flat files must either include attributes for all possible granularities, or by choosing a subset limit the possible resultant discoveries. By using tree-structured attributes, the data miner can represent the hierarchy of values separately, thereby maintaining an economical representation of the data set. For example, if each instance contains a specific location, this location can be used to index into the hierarchy. Usually, instances contain the finest granularity and the hierarchy can be used to draw more general comparisons. Efficient mining with tree-structured

attributes is treated in depth by Allmuallim et al. (1995) and improvements are described by Aronis and Provost (1997).

Tree-structured attributes allow the representation of a simple relation, the *isa* relation, between attribute-value pairs. Each such relation can be seen as a separate table in a relational database, for example a state/county table or a county/zipcode table. Expanding data mining to general multitable relational databases is an obvious next step, which has been advocated recently in inductive learning research circles (Aronis, Kolluri, Provost, and Buchanan 1997) (Kohavi 1998). The ability to handle multitable databases not only allows practitioners to compress otherwise unwieldy flat tables, it also creates possibilities for augmenting learning systems with more and more related knowledge. For each field of a learning problem, practitioners can consider whether there exist additional tables of knowledge describing that field.[7]

It should be noted that selecting just the right auxiliary databases/knowledge bases begs the very question of data mining, because it requires identifying which databases contain relevant and useful knowledge. One can envision augmenting many fields with related tables, and within the new data, augmenting additional fields with related tables, and so on. Therefore, from a scaling up perspective, it becomes necessary to be able to learn in the context of massive amounts of background knowledge,[8] creating the need for even higher degrees of scaling in data mining systems.

This view unifies learning from relational databases with learning with large amounts of background knowledge. Parallel marker-passing techniques can be used to aid in augmenting inductive learners with large networks of relational background knowledge (Aronis and Provost 1994). In the work of Aronis and Provost, the relational knowledge is used to construct new terms, which are then added to the propositional concept description language.

The field of Inductive Logic Programming (ILP) (Muggleton 1992) concentrates on mining data and knowledge expressed in a relational format. However, ILP addresses a harder problem than the type of mining we are considering. Specifically, not only are the data represented relationally, the results of the mining also may be represented relationally. Because learning relational descriptions is harder than learning propositional ones, relational algorithms are considerably slower than propositional ones, and the scaling problem is correspondingly harder.

However, Blockeel, De Raedt, Jacobs, and Demoen (1999) observe that "the full power of standard ILP is not used for most practical applications." Therefore, a general approach to speeding up learning with relational data is to avoid expensive but little-used constructs. Aronis et al. (1996) investigate induction from feature-vector-based data items linked to relational background knowledge. For the sake of efficiency, they purposely avoid n-ary and recursive relational terms. Blockeel et al. (1999) study an efficient subset of ILP known as *learning from interpretations*. In particular, they study scaling up *first-order logical decision trees* (FOLDTs), which are more expressive than propositional decision trees, but also avoid the most expensive ILP constructs. Of particular note, FOLDTs allow mining data expressed as multitable relational databases. As part of their study, Blockeel et al.

consider the application of techniques from SLIQ (Mehta, Agrawal, and Rissanen 1996) to scale up learning FOLDTs to massive data sets. Their results are encouraging: FOLDTs can be learned with run-time complexity linear in the number of examples, and very large data sets can be mined efficiently (efficiently relative to other ILP techniques—several non-trivial tasks with approximately 100,000 examples (100Mbytes) each were processed in about a day of CPU time on a Sun workstation). In another scalable-ILP project, Brockhausen and Morik (1996) describe the integration of an ILP algorithm to a DBMS, facilitating efficient learning directly from DBMS-resident data—which is the subject of the next section.

6.3.2. Data mining/DBMS integration For many applications, data are already stored in an efficient relational representation—a multitable relational database, most easily accessible via a commercial database management system (DBMS). Although many data mining systems access data stored in a commercial DBMS, most of these do not actually mine the relational data directly. Rather, they extract the data from the DBMS into a memory-resident flat file, thereby not realizing the benefits of efficient storage discussed in the previous subsection. Such approaches only take advantage of the DBMS's efficient data retrieval.

Relational data, stored in a commercial DBMS, can be mined directly by implementing the core data manipulation operations within the DBMS. As discussed in Section 6.1, the speed of inductive programs often is determined primarily by the speed of the matching or the gathering of sufficient statistics. If these operations can be cast as SQL requests for statistics (Agrawal and Shim 1995) (Agrawal and Shim 1996), a data mining program can avoid massive data uploads and problems due to main memory restrictions, and can take advantage of fast database machines optimized for query processing. In their proposal for the SQL Interface Protocol, John and Lent (1997) discuss how the basic operations for various types of data mining programs can be cast as SQL queries.

Graefe, Fayyad and Chaudhuri (1998) show that a straightforward implementation for deriving sufficient statistics from SQL databases (using `SELECT` and `UNION` operators) results in unacceptably poor performance. This poor performance stems from the manner in which the database system will implement the query; specifically, most database systems will implement a `UNION` query by performing a separate scan for each clause in the `UNION`. However, for deriving sufficient statistics the `UNION`s will be very similar. The authors propose to take advantage of this similarity by extending SQL to include a new operator (`UNPIVOT`), which minimizes the number of scans required to produce the sufficient statistics.

Figure 10 shows a schematic view of a system integrating data mining with a DBMS. The DBMiner data mining system (Han, Fu, Wang, Chiang, Gong, Koperski, Li, Lu, Rajan, Stefanovic, Xia, and Zaiane 1996) is a prototypical example of such an integrated data mining/DBMS system. Data Surveyor (Holsheimer, Kersten, and Siebes 1996) and SKICAT (Fayyad, Weir, and Djorgovski 1993) also

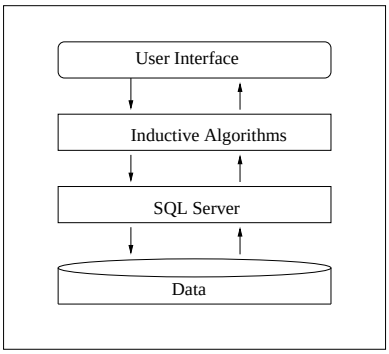

Figure 10. Data mining/DBMS integration

make use of the data mining/DBMS integration approach to achieve competitive performance on large data sets.

Sarawagi et al. (1998) discuss several alternatives for data mining/DBMS integration. Their focus is on mining association rules, but they illustrate principles that apply more generally. In particular, they point to several efforts to extend SQL to support mining operations, and discuss expressing mining algorithms in SQL. The following paragraphs parallel their high-level discussion; their paper gives more details on specific approaches.

As noted above, the most common use of a DBMS for data mining is as a simple source of records. If the DBMS and the data mining program are coupled too loosely, the cost of switching contexts between programs may become prohibitive. This is true especially if records are read individually. Block transfers make more sense.

A more tightly coupled approach pushes parts of the application program that perform intensive computations on the retrieved set of records into the database system, instead of bringing the records of the database into the application program. One method is to encapsulate the mining algorithm as a stored procedure. This approach allows the programs to share one address space, with the corresponding efficiencies, while maintaining programming flexibility. A hybrid of these approaches is to read the data once from the database into a temporary local cache, perhaps transforming it to a more efficient format. The drawback of the caching approach is the need for additional storage space.

A somewhat different approach is to represent the individual data mining operations as user-defined functions, rather than as stored procedures, placed in SQL data scan queries (which also will run in the DBMS address space) (Agrawal and Shim 1995) (Agrawal and Shim 1996). Such an approach promises to be faster, because passing records to a user-defined function is faster than passing them to a stored procedure. The disadvantage is the cost of rewriting entire mining algorithms as user-defined functions. Sarawagi et al. also consider the more general

case where a preprocessor translates data mining operations into the appropriate form for a particular environment.

Integrating data mining with the DBMS takes advantage of the storage efficiencies of relational representations, of the existence of indices, and of the fact that DBMSs typically reside on powerful platforms that are optimized for database operations. As described in Section 6.1.4, scaling can be extended further by making use of parallel database server technology to speed up data-intensive SQL operations. In one implementation, Freitas and Lavington (1996) achieved an order of magnitude speedup over a workstation of the same DBMS technology by making use of a back-end 12-processor SQL server. Data Surveyor (Holsheimer, Kersten, and Siebes 1996) also uses a parallel database engine.

Finally, the fact that data already reside in a DBMS is not the only reason to consider integrating data mining. There may be advantages to using a DBMS for other complex representations of knowledge. When faced with complex relational representations, one of the scaling problems realistic applications must address is that existing knowledge representation systems do not provide high-speed access to large, complex knowledge bases (Karp and Paley 1995). Karp, Paley and Greenberg show that one can employ a DBMS effectively as the storage subsystem for large, frame-based knowledge representations (Karp, Paley, and Greenberg 1994) (Karp and Paley 1995). Andersen, Hendler, Evett and Kettler also describe knowledge representation tools that scale to massive knowledge bases (Andersen, Hendler, Evett, and Kettler 1994) (Evett 1994). Although initially most of the efficiency gains they realized were due to massive parallelism, they too have made increasing use of DBMSs to achieve increased efficiency (while still allowing for effective parallelization). Specifically, they use DBMS techniques to support matching, inference and data management. Advances such as these in the efficient handling of large-scale knowledge bases should facilitate future efforts to mine them or to use them to augment data mining.

6.3.3. Distributed databases Enabling inductive programs to learn from multi-table relational databases makes available to data mining the vast amount of data and background knowledge distributed across a local network, or scattered about the Internet. For example, companies are interested in mining federations of similar data (Stolfo, Fan, Lee, Prodromidis, and Chan 1997), and digital library research is working to facilitate access to networked data and information (Fox, Akscyn, Furuta, and Legsett 1995). Along with the desire to take advantage of these collections comes the need to scale up to massive amounts of distributed data and background information. This scaling problem manifests all the issues discussed so far in this survey, plus some additional constraints and opportunities.

Not only do distributed data provide the opportunity for concurrent mining of different subsets, similar to the more straightforward uses of parallelism and data partitioning, distributed data may *require* distributed mining. Combining already distributed databases may be out of the question for a variety of reasons. They may simply be too big to combine on a local system. The bandwidth of the communications channel may make combining databases infeasible, because it would take too

long to download the data. Finally, privacy issues may prevent unrestricted access to the data. For any of these reasons, a database of interest may be accessible over the network, but transferring it may not be feasible.

Mining distributed databases requires a system that can operate on many separate data partitions. The information that can be transferred is limited by bandwidth and other (e.g., privacy) restrictions. For data sets where rows or columns are distributed, the methods discussed in Section 6.2.4 are for the most part appropriate. Stolfo et al. present their approach and an implemented system, and discuss how privacy concerns restrict the federation of banking data (Stolfo, Fan, Lee, Prodromidis, and Chan 1997) (Stolfo, Prodromidis, Tselepis, Fan, Lee, and Chan 1997).

In an alternative distributed data mining scenario, different database *tables* are spread about the network. Consider our simplistic multitable example above, which had three tables: customer information, geographic information, product information. In many real-world situations, different tables such as these would not reside on the same machine. Currently, much of the practical work of data mining comprises locating relevant tables in different databases, and transferring them (or carefully selected subsets) to the data mining platform. If network access were provided to these data, through SQL servers or tailored data-mining servers, an inductive algorithm could query the remote databases as necessary during data mining.

Ribeiro, Kaufman and Kersberg (1995) describe a method for performing knowledge discovery across multiple databases by using *foreign-key* values to augment tables. Specifically, they propose tracing through multiple databases following the *foreign-keys*, and learning individual *knowledge segments* for each database. The WoRLD system (Aronis, Kolluri, Provost, and Buchanan 1997) learns across multiple distributed databases spread across the network, using *spreading activation* techniques. These require only limited communication to pass sets of markers (implemented with SQL queries).

7. Discussion

By looking systematically at the body of work on scaling up inductive methods, it is clear that several areas have received relatively deep treatment. We first remark on what this work suggests you do when faced with mining a huge data set. Then we discuss where more research can provide the most help, from perspectives of statistics, databases, and machine learning.

7.1. What should you do with a huge data set?

When a large data set can fit in main memory, restricted model space learners should be tried first, because they often are effective at building competitive classifiers quickly. If the resulting simple classifiers are not satisfactory, there are several fast, effective algorithms for data sets that can fit in main memory. Extending this concept, one direction that has been suggested is to build algorithms that use a simplicity-first strategy in their search for classifiers (Holte 1993) (Provost 1993). It

is not yet clear how much leverage can be obtained through the use of simple classifiers to guide subsequent search to address specific deficiencies in their performance (Holte 1993).

So, what if a data set of interest does not fit in main memory? Or, what if the data set does fit in a machine's huge main memory, but mining it there just is not efficient enough? Although research on induction when the data set is too large to fit in main memory is not nearly as comprehensive, several techniques are clear choices when they apply. But first, we should revisit briefly the need, or lack thereof, to mine the entire data set.

For most data mining problems, the benefit of increasing the size of the data set decreases as the data set size grows, yielding the familiar concave-down learning curve. Eventually, the increase in the quality of the results becomes negligible (or simply zero). For a given problem, how can you determine where the learning curve will plateau? For most algorithms, calculating precisely, a priori, a tight bound on the size of the required data set is difficult. Computational learning theory does provide upper bounds, for many concept classes for a particular kind of learning, which should not be ignored. However, it may be that for a particular learning algorithm these bounds are weak and many fewer instances are actually needed. Of course, theoretical calculations are not the only method. Considering that the run-time complexity of inductive algorithms is at best linear in the number of examples, and often worse, relatively inexpensive experiments can be conducted on small samples in order to estimate the number of examples that are actually needed (John and Langley 1996; Frey and Fisher 1999; Provost, Jensen, and Oates 1999). In cases where the number of examples needed is much smaller than the number available, such procedures can provide substantial practical speedups.

Subsets of the examples should be sampled, using stratified sampling when one class dominates strongly. Subsets of the features should also be selected, by doing empirical studies to determine relevance.[9] Once a practitioner has chosen a good subset of examples, a good subset of features, and an algorithm with an efficient data representation, there may not be a significant increase in accuracy when learning with more data than will fit in main memory—especially with a modern computer with memory slots filled to capacity.

Once such straightforward methods have been exhausted, the best approach depends on the resources available. Massively parallel matching is an obvious choice for increased scaling, if access to a massively parallel machine and specialized programming talent are available. However, one should carefully examine the tradeoffs between matching and gathering sufficient statistics. Taking advantage of powerful, well-tuned database systems, via data mining/DBMS integration, is a good idea if cycles on the database engine are readily available, but such an approach may require a significant investment and is appropriate mostly for long-term plans to mine a set of data. Also, the loss of flexibility in choosing and modifying inductive methods cannot be ignored, since problem engineering is such a large portion of the overall KDD process. Specialized programming talent usually is required.

Independent multi-subset learning shows promise for scaling up and retaining the flexibility of desktop data mining. The ability to process the subsets concur-

rently offers to take advantage of the large number of idle workstations that are already networked in most institutions. Unfortunately, once again there is currently a dearth of available technology to facilitate such learning, so specialized programming talent is still required. A notable exception is the JAM (Java Agents for Meta-learning) software, downloadable from Stolfo's web site (Stolfo 1998).

7.2. Where will more research most help?

As with most issues in data mining, even when problem and environmental characteristics dictate a general approach, there is little guidance for choice among the various constituent methods. For each approach, several methods have been studied in isolation, but there exist few (if any) studies comparing their relative merits. For example, for partitioned-data approaches, research has only just reached the border between the proof-of-concept stage and the comparative-evaluation stage. More theoretical and empirical research is needed before we can claim a thorough understanding.

What is most needed is a better treatment of sampling. The KDD community, including researchers from all the different perspectives, should achieve a common understanding of when and how sampling should be used, and should build artifacts enabling it. For example, what are the effects of stratified sampling on predictive performance on the original distribution? (See the recent work of Chan and Stolfo (1998).) Is it time to revive the concept of the "near miss" as a way to choose data subsets intelligently, once a tentative classifier has been built? Let us consider what can be contributed from the three most commonly cited KDD component fields: (1) statistics, (2) databases, and (3) machine learning.

(1) Statistics has a long and rich history of theoretical work on sampling. Although traditionally it has focused on small samples and on hypothesis verification, it may be applicable to computer-driven discovery either directly or when tailored to the current context. KDD should embrace efforts by statisticians, based either on new or past research, to provide a common theoretical understanding of the issue.

(2) Most discussions of sampling assume that producing random samples efficiently from large data sets is not difficult. For most large databases this simply is not true. To produce a random sample from even a single-table database may require scanning the entire table. A naive implementation may be much worse. This has obvious implications for claims of algorithm efficiency with sampling; for example, the asymptotic run-time complexity will be at least linear in the *total number* of instances. Along with a better understanding of sampling must come database operations and organizations that allow for efficient sampling (Fayyad 1997).

(3) Remarkable advances have been made in learning effectively from large numbers of examples, and (separately) in learning effectively when there are large numbers of features. However, the data mining problems most in need of scaling are those with massive numbers of both. Main-memory learning algorithms are fast and effective enough that a good deal of experimentation is possible, and this type of experimentation can be automated to a large degree (Moore and Lee 1994) (Provost

and Buchanan 1995) (Kohavi and John 1997). Indeed, some existing algorithms may provide strong baselines against which new approaches can be compared. For example, the well-known algorithm WINNOW (Littlestone 1988) is not only effective at detecting irrelevant features, it is also incremental; it should be straightforward to augment the algorithm to reduce the number of features automatically as the number of examples grows (thus keeping the total size of the data set small).

Another area in need of research effort is mining large multitable relational databases/databases linked to relational background knowledge. Such research would have broad implications, affecting the development of data mining/DBMS integrated systems, algorithms for learning in main memory, and partitioned-data approaches. Given the storage economies possible with relational representations, their use promises that much larger data sets can be processed in main memory than with flat file representations. Furthermore, the ability to mine distributed, structured data and knowledge efficiently will dovetail nicely with current efforts to make available vast amounts of metadata-indexed information (Fox, Akscyn, Furuta, and Legsett 1995), bringing into view new research horizons.

8. Conclusion

We have reached some conclusions after reading and organizing more than a hundred papers on scaling up inductive algorithms. Much work has been done, and the establishment of KDD as an coherent field of study seemingly has accelerated the production of relevant results.

Because of the interdisciplinary nature of the field, often research is undertaken without the benefit of insights from substantially similar work that has taken place either in another subfield, or that has taken place as a necessary peripheral task in a research effort with a different focus. We found striking similarities between efforts that, as far as we can determine, were completely independent. This is not unexpected, and is in fact encouraging: often when "the time comes," fundamentally identical technical advances are made simultaneously by independent groups. We hope that this survey helps to establish common ground from which future efforts can reach even higher.

The design of fast algorithms stands out as a clear example of effective incremental research: there are clear chains of advances in fast rule learning and in fast decision-tree learning. Research on partitioned-data approaches and on mining relationally represented data is less mature, consisting mainly of independent work, but there are striking commonalities among the existing approaches. We believe these areas are ready for more emphasis on comparative research, followed by (hopefully significant) incremental advances.

The most glaring gaps in the literature are the lack of a common understanding of the power (or impotence) of sampling for data mining, and the dearth of convincing examples of the *need* to mine massive data sets.

9. Acknowledgements

We are indebted to many, including John Aronis, Lars Asker, Bruce Buchanan, Jason Catlett, Pedro Domingos, Phil Chan, Doug Fisher, Dan Hennessy, David Jensen, Ronny Kohavi, Rich Segal, Sal Stolfo, and anonymous referees of previous papers, who have influenced our views of scaling up through many discussions. Thanks also to the many who have pointed us to relevant work. Peter Huber gave an invited talk at KDD-97 on "From Large to Huge: A Statistician's Reactions to KDD & DM," a summary of which appears in the proceedings (along with our summary of this survey). We were delighted that many of his observations agreed with ours, and have included some of his specific statements. Tom Dietterich reviewed the state of scaling up in his recent article on trends in machine learning (Dietterich 1997). Although the development of the two reviews occurred independently and contemporaneously, we have had the benefit his insights in producing this final version. Finally, we thank Pedro Domingos, Tom Fawcett, Usama Fayyad, Rick Kufrin, Doug Metzler, Ron Musick, and our anonymous referees for comments on drafts, and Usama Fayyad for encouraging us to turn our informal summary into a formal survey. This work was partly supported by National Science Foundation grant IRI-9412549.

Notes

1. A version of this paper was available as a technical report on the web (Provost and Kolluri 1997b) and an overview appears in the proceedings of KDD-97 (Provost and Kolluri 1997a).
2. One study of large data sets shows a polynomial estimation of C4.5's run-time complexity to fall between $O(e^{1.2})$ and $O(e^{1.4})$ (Provost, Jensen, and Oates 1999).
3. These run times are given for illustration only. No comparison should be inferred.
4. In fact, the experiments of Harris-Jones and Haines were conducted on a dual-processor 133MHz Compaq computer with 256M RAM running Windows NT. The run time for C5 on 283,649 examples was fifty minutes (fifteen minutes for 99,303 examples).
5. Before flattening, the customer table requires $1,000,000$ customers * 20 fields/customer * 5 bytes/field = $100,000,000$ bytes, the state table requires 50 states * 80 fields/state * 5 bytes/field = 20,000 bytes, and the product table requires 50 products * 400 fields/product * 5 bytes/field = 100,000 bytes, for a pre-flattening total size of 100,120,000 bytes. To flatten, the appropriate state and product information is spliced into each customer record, yielding a new customer record comprising 498 fields. For the same one million customers, the new space requirement is 1,000,000 customers * 498 fields/customer * 5 bytes/field = $2.49 * 10^9$ bytes.
6. It should be noted that even with a fast database machine, mining database-resident data directly is often considerably slower than flat-file mining (Musick 1998).
7. To justify calling linked tables "knowledge," recall the direct correspondence between relational databases and knowledge representation structures (Hayes 1979).
8. Gaines (1989) analyzed the extent that prior knowledge *reduces* the amount of data needed for effective learning.
9. Remember that the size of the example set is a product of the number of examples and the number of features. Halving the number of features by eliminating irrelevant ones not only may increase the accuracy by itself, it also will allow one to double the number of examples occupying a fixed space.

References

Agrawal, R. and K. Shim (1995). Developing tightly-coupled applications on IBM DB2/CS relational database system: Methodology and experience. Research Report RJ 10005(89094), IBM Corporation.

Agrawal, R. and K. Shim (1996). Developing tightly-coupled data mining applications on a relational database system. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 287–290. AAAI Press.

Agrawal, R. and R. Srikant (1994). Fast algorithms for mining association rules. Research Report RJ9839, IBM Corporation.

Ali, K. M. and M. J. Pazzani (1996). Error reduction through learning multiple descriptions. *Machine Learning 24*(3), 173–202.

Almuallim, H., Y. Akiba, and S. Kaneda (1995). On handling tree-structure attributes in decision tree learning. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann.

Andersen, W., J. Hendler, M. Evett, and B. Kettler (1994). Massively parallel matching of knowledge structures. In H. Kitano and J. Hendler (Eds.), *Massively Parallel Artificial Intelligence*. AAAI/MIT Press.

Aronis, J., V. Kolluri, F. Provost, and B. Buchanan (1997). The WoRLD: Knowledge discovery from multiple distributed databases. In *Proceedings of Florida Artificial Intelligence Research Symposium (FLAIRS-97)*.

Aronis, J. and F. Provost (1994). Efficiently constructing relational features from background knowledge for inductive machine learning. In *Working Notes of the AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle WA.

Aronis, J. and F. Provost (1997). Increasing the efficiency of data mining algorithms with breadth-first marker propagation. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Newport Beach, CA.

Aronis, J., F. Provost, and B. Buchanan (1996). Exploiting background knowledge in automated discovery. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 355–358. AAAI Press.

Auer, P., R. Holte, and W. Maas (1995). Theory and applications of agnostic pac-learning with small decision trees. In *Proceedings of the Twelveth International Conference on Machine Learning*, pp. 21–29.

Blockeel, H., L. De Raedt, N. Jacobs, and B. Demoen (1999). Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*. To appear.

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

Brockhausen, P. and K. Morik (1996). Direct access of an ILP algorithm to a database management system. In *Proceedings of MLnet Sponsored Familiarization Workshop Data Mining with Inductive Logic Programming*.

Buchanan, B. and E. Feigenbaum (1978). DENDRAL and META-DENDRAL: Their applications dimensions. *Artificial Intelligence 11*, 5–24.

Buchanan, B. G., D. H. Smith, W. C. White, R. Gritter, E. A. Feigenbaum, J. Lederberg, and C. Djerassi (1976). Applications of artificial intelligence for chemical inference, xxii. automatic rule formation in mass spectrometry by means of the META-DENDRAL program. *Journal of the American Chemical Society 96:6168*.

Buntine, W. (1991). *A theory of learning classification rules*. Ph. D. thesis, School of Computer Science, University of Technology, Sydney, Australia.

Catlett, J. (1991a). Megainduction: A test flight. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 596–599. Morgan Kaufmann.

Catlett, J. (1991b). *Megainduction: Machine learning on very large databases*. Ph. D. thesis, School of Computer Science, University of Technology, Sydney, Australia.

Chan, P. and S. Stolfo (1993). Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Workshop Knowledge Discovery in Databases*, pp. 227–240.

Chan, P. and S. Stolfo (1997). On the accuracy of meta-learning for scalable data mining. In *Journal of Intelligent Information Systems*, Volume 8, pp. 5–28.

Chan, P. and S. Stolfo (1998). Towards scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 164–168.

Chen, M., J. Han, and P. Yu (1997). Data mining: An overview from database perspective. In *IEEE Transactions on Knowledge and Data Engineering*.

Chen, M.-S. and P. Yu (1995). Using multi-attribute predicates for mining classification rules. Technical report, IBM Research Report.

Clearwater, S., T. Cheng, H. Hirsh, and B. Buchanan (1989). Incremental batch learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, San Mateo CA:, pp. 366–370. Morgan Kaufmann.

Clearwater, S. and F. Provost (1990). RL4: A tool for knowledge-based induction. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*, pp. 24–30. IEEE C.S.Press.

Cohen, W. W. (1993). Efficient pruning methods for separate-and-conquer rule learning systems. In *Thirteenth International Joint Conference on Artificial Intelligence*, pp. 988–994. Morgan Kaufmann.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123.

Cook, D. and L. Holder (1990). Accelerated learning on the connection machine. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*, San Mateo CA, pp. 366–370. Morgan Kaufmann.

Craven, M. W. (1996). *Extracting Comprehensible Models from Trained Neural Networks*. Ph. D. thesis, University of Wisconson – Madison. Technical Report No. 1326.

Danyluk, A. and F. Provost (1993). Small disjuncts in action: Learning to diagnose errors in the telephone network local loop. In P. Utgoff (Ed.), *Machine Learning: Proceedings of the Tenth International Conference*, pp. 81–88. Morgan Kaufmann Publishers, Inc.

DesJardins, M. and D. F. Gordon (1995). Special issue on bias evaluation and selection. *Machine Learning 20*(1/2).

Devijver, P. A. and J. Kittler (1982). *Pattern recognition: A statistical approach*. Prentice Hall.

Dietterich, T. G. (1997). Machine learning research: Four current directions. *AI Magazine 18(4)*, 97–136.

Domingos, P. (1996a). Efficient specific-to-general rule induction. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 319–322. AAAI Press.

Domingos, P. (1996b). Linear time rule induction. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 96–101. AAAI Press.

Domingos, P. (1997). Knowledge acquisition from examples via multiple models. In D. H. Fisher (Ed.), *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-97)*, pp. 98–106. San Francisco, CA:Morgan Kaufmann.

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York: John Wiley.

Evett, M. (1994). *PARKA: A System for Massively Parallel Knowledge Representation*. Ph. D. thesis, Department of Computer Science, University of Maryland, College Park, Maryland.

Fawcett, T. and F. Provost (1997). Adaptive fraud detection. *Data Mining and Knowledge Discovery 1*(3), 291–316.

Fayyad, U. (1997). Editorial. *Data Mining and Knowledge Discovery 1(1)*, 5–10.

Fayyad, U., D. Haussler, and P. Stolorz (1996). KDD for science data analysis: Issues and examples. In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, Menlo Park, CA., pp. 50–56. AAAI Press.

Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth (1996a). Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, Menlo Park, CA., pp. 82–88. AAAI Press.

Fayyad, U., N. Weir, and S. Djorgovski (1993). SKICAT: A machine learning system for automated cataloging of large scale sky surveys. In *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann.

Fayyad, U. M., G. Piatetsky-Shapiro, and P. Smyth (1996b). From data mining to knowledge discovery: An overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthursamy (Eds.), *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press.

Fox, E. A., R. M. Akscyn, R. Furuta, and J. Legsett (1995). *Communications of the ACM*, Volume 38(4). Morgan Kaufmann.

Freitas, A. and S. Lavington (1996). Using SQL primitives and parallel DB servers to speed up knowledge discovery in large relational databases. In *Cybernetics and Systems'96: Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research*, pp. 955–960.

Freitas, A. A. and S. H. Lavington (1997). *Mining Very Large Databases with Parallel Processing*. Norwell, MA: Kluwer Academic Publishers.

Frey, L. J. and D. H. Fisher (1999). Modeling decision tree performance with the power law. In D. Heckerman and J. Whittaker (Eds.), *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*. San Francisco, CA: Morgan Kaufmann.

Friedman, J. H. (1997). Data mining and statistics: What's the connection? In *Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics*.

Fürnkranz, J. (1998). Integrative windowing. *Journal of Artificial Intelligence Research 8*, 129–164.

Fürnkranz, J. and G. Widmer (1994). Incremental reduced error pruning. In *Proceedings of the Eleventh International Machine Learning Conference*, New Brunswick. Morgan Kaufmann.

Gaines, B. (1989). An ounce of knowledge is worth a ton of data: Quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, San Mateo, CA, pp. 156–159. Morgan Kaufmann.

Galal, G., D. J. Cook, and L. Holder (1999). Exploiting parallelism in a scientific discovery system to improve scalability. *Journal of the American Society for Information Science*. To appear.

Gehrke, J., R. Ramakrishnan, and V. Ganti (1998). Rainforest - a frameword for fast decision tree construction of large datasets. In *Proceedings of the Twenty-fourth International Conference on Very Large Data Bases*, New York, NY.

Graefe, G., U. Fayyad, and S. Chaudhuri (1998). On the efficient gathering of sufficient statistics for classification of large SQL databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data-Mining*, New York, N.Y. AAAI Press.

Grossman, R. and S. Bailey (1998). A tutorial introduction to high performance data mining. Tutorial given at the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98).

Guo, Y. and J. Sutiwaraphun (1998). Knowledge probing in distributed data mining. In *Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pp. 61–69.

Haines, T. L. (1998). Private communication.

Hall, L. O., N. Chawla, and K. W. Bowyer (1998). Combining decision trees learned in parallel. In *Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pp. 10–15.

Han, J., Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaiane (1996). DBMiner: A system for mining knowledge in large relational databases. In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, Menlo Park, Ca., pp. 250–255. AAAI Press.

Harris-Jones, C. and T. L. Haines (1997). Sample size and misclassification: Is more always better? Working Paper AMSCAT-WP-97-118, AMS Center for Advanced Technologies.

Haussler, D. (1988). Quantifiying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence 36*, 177–221.

Hayes, P. (1979). The logic of frames. In D. Metzing (Ed.), *Frame Conceptions and Text Understanding*, pp. 46–61. de Gruyter.

Holsheimer, M., M. Kersten, and A. Siebes (1996). Data Surveyor: Searching the nuggets in parallel. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthursamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 447–467. Menlo Park: AAAI Press.

Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning 3*, 63–91.

Holte, R., L. Acker, and B. Porter (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, San Mateo, CA, pp. 813–818. Morgan Kaufmann.

Huber, P. (1997). From large to huge: A statistician's reaction to KDD and DM. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 304–308. AAAI Press.

Iba, W. and P. Langley (1992). Induction of one-level decision trees. In *Proceedings of Ninth International Conference on Machine Learning*, pp. 233–240. Morgan Kaufmann.

Jensen, D. (1998). Private communication.

Jensen, D. and P. R. Cohen (1999). Multiple comparisons in induction algorithms. *Machine Learning*. To appear.

John, G. and P. Langley (1996). Static versus dynamic sampling for data mining. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 367–370. AAAI Press.

John, G. and B. Lent (1997). SIPping from the data firehose. In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 199–202. AAAI Press.

Kargupta, H. and P. Chan (Eds.) (1998). *KDD-98 Workshop on Distributed Data Mining*.

Kargupta, H., E. Johnson, E. R. Sanseverino, B.-H. Park, L. D. Silvestre, and D. Hershberger (1998). Scalable data mining from distributed, vertically partitioned feature space using collective mining and gene expression based genetic algorithms. In *Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pp. 70–91. http://www.eecs.wsu.edu/~hillol/pubs/bodhi.ps.Z.

Karp, P. D. and S. M. Paley (1995). Knowledge representation in the large. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Menlo Park, CA, pp. 751–758. AAAI Press.

Karp, P. D., S. M. Paley, and I. Greenberg (1994). A storage system for scalable knowledge representation. In *Proceedings of the Third International Conference on Information and Knowledge Management*.

Kaufman, K. and R. Michalski (1996). A method for reasoning with structured and continuous attributes in the INLEN-2 knowledge discovery system. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 232–237. AAAI Press.

Kearns, M. (1993). Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth ACM Symposium on the Theory of Computing*, New York, NY, pp. 392–401. ACM Press.

Kohavi, R. (1996). *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. Ph. D. thesis, Dept. of Computer Science, Stanford University, Palo Alto, CA.

Kohavi, R. (1998). Crossing the chasm: From academic machine learning to commercial data mining. Invited talk for the Fifteenth International Conference on Machine Learning.

Kohavi, R. and G. John (1997). Wrappers for feature subset selection. *Artificial Intelligence 97(1-2)*, 273–324.

Kohavi, R. and D. Sommerfield (1995). Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA. AAAI Press.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of Relief. In F. Bergadano and L. D. Raedt (Eds.), *Proceedings of the European Conference on Machine Learning*.

Kononenko, I., E. Simec, and M. Robnik-Sikonja (1997). Overcoming the myopia of inductive learning algorithms with RELIEFF. *Applied Intelligence 7*, 39–55.

Kufrin, R. (1997). Generating C4.5 production rules in parallel. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 565–670. Menlo Park, CA:AAAI Press.

Kumar, V. and V. Rao (1987). Parallel depth-first search, part 2: Analysis. *International Journal of Parallel Programming 16*, 501–519.

Lathrop, R., T. Webster, T. Smith, and P. Winston (1990). ARIEL: A massively parallel symbolic learning assistant for protein structure/function. In P. Winston and S. Shellard (Eds.), *AI at MIT: Expanding Frontiers*, Cambridge, MA. MIT Press.

Li, B. (1998). *Free Parallel Data Mining*. Ph.d. thesis, Department of Computer Science, New York University.

Lim, T.-J., W.-Y. Loh, and Y.-S. Shih (1999). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*. To appear.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning 2*, 285–318.

Mehta, M., R. Agrawal, and J. Rissanen (1996). SLIQ: A fast scalable classifier for data mining. In *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT)*, Avignon, France.

Merz, C. J. and P. M. Murphy (1997). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Miller, A. (1990). *Subset selection in regression*. Chapman and Hall.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence 18*(2), 203–226.

Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical Report Report CBM-TR-117, Rutgers University, New Brunswick, NJ.

Moore, A. and M. Lee (1994). Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann.

Moore, A. and M. Lee (1998). Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research 8*, 67–91.

Muggleton, S. (1992). *Inductive Logic Programming*. London: Academic Press Ltd.

Musick, R. (1998). Supporting large-scale computational science. Technical Report UCRL-ID-129903, Center for Applied Scientific Computing, Lawrence Livermore National Lab.

Musick, R., J. Catlett, and S. Russell (1993). Decision theoretic subsampling for induction on large databases. In *Proceedings of the Tenth International Conference on Machine Learning*, San Mateo, CA, pp. 212–219. Morgan Kaufmann.

Oates, T. and D. Jensen (1997). The effects of training set size on decision tree complexity. In D. Fisher (Ed.), *Machine Learning: Proceedings of the Fourteenth International Conference*, pp. 254–262. Morgan Kaufmann.

Oates, T. and D. Jensen (1998). Large data sets lead to overly complex models: an explanation and a solution. In R. Agrawal and P. Stolorz (Eds.), *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, pp. 294–298. Menlo Park, CA: AAAI Press.

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

Piatetsky-Shapiro, G., R. Brachman, T. Khabaza, W. Kloesgen, and E. Simoudis (1996). An overview of issues in developing industrial data mining and knowledge discovery applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA., pp. 89–95. AAAI Press.

Prodromidis, A. L. and S. J. Stolfo (1998). Pruning meta-classifiers in a distributed data mining system. In *Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pp. 22–30.

Provost, F. (1992). *Policies for the Selection of Bias in Inductive Machine Learning*. Ph. D. thesis, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA.

Provost, F. and J. Aronis (1996). Scaling up inductive learning with massive parallelism. *Machine Learning 23*, 33–46.

Provost, F. and B. Buchanan (1995). Inductive policy: The pragmatics of bias selection. *Machine Learning 20*, 35–61.

Provost, F. and D. Hennessy (1996). Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, CA. AAAI Press.

Provost, F., D. Jensen, and T. Oates (1999). Efficient progressive sampling. Technical report 99-14, Department of Computer Science, University of Massachusetts/Amherst.

Provost, F. and V. Kolluri (1997a). Scaling Up inductive algorithms: An overview. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining.*, Menlo Park, CA, pp. 239–242. AAAI Press.

Provost, F. and V. Kolluri (1997b). A survey of methods for scaling up inductive learning. Technical Report ISL-97-3, Intelligent Systems Laboratory, University of Pittsburgh, Pittsburgh, PA. http://www.pitt.edu/~uxkst/survey-paper.ps.

Provost, F. J. (1993). Iterative weakening: Optimal and near-optimal policies for the selection of search bias. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 749-755, pp. Menlo Park, CA. AAAI Press.

Provost, F. J. and D. Hennessy (1994). Distributed machine learning: scaling up with coarse-grained parallelism. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*.

Quinlan, J. (1983). Learning efficient classification procedures and their application to chess endgames. In R. Michalski, C. J., and T. Mitchell (Eds.), *Machine Learning: An AI approach*. Los Altos, CA: Morgan Kaufmann.

Quinlan, J. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies 27*, 221–234.

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

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

Rao, V. and V. Kumar (1987). Parallel depth-first search, part 1: Implementation. *International Journal of Parallel Programming 16*, 479–499.

Ribeiro, J., K. Kaufmann, and L. Kerschberg (1995). Knowledge discovery from multiple databases. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, pp. 240–245. AAAI Press.

Riddle, P., R. Segal, and O. Etzioni (1994). Representation design and brute-force induction in a boeing manufacturing domain. *Applied Artificial Intelligence 8*, 125–147.

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

Sarawagi, S., S. Thomas, and R. Agrawal (1998). Integrating association rule mining with relational database systems: Alternatives and implications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

Savasere, A., E. Omiecinski, and S. Navathe (1995). An efficient algorithm for mining association rules in large databases. In *Proceedings of Twenty-First International Conference on Very Large Data Bases*, pp. 432–444. Morgan Kaufmann.

Schlimmer, J. C. (1993). Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In P. Utgoff (Ed.), *Proceedings of the Tenth International Conference on Machine Learning*, pp. 284–290. San Mateo, CA: Morgan Kaufmann.

Segal, R. and O. Etzioni (1994a). Learning decision lists using homogeneous rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 619–625. AAAI Press.

Segal, R. and O. Etzioni (1994b). Learning decision lists using homogenous rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, pp. 619–625. AAAI Press.

Shafer, J., R. Agrawal, and M. Mehta (1996). SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, Mumbai, India.

Shasha, D. (1998). PC4.5. http://merv.cs.nyu.edu:8001/~binli/pc4.5.

Shavlik, J. W., R. J. Mooney, and G. G. Towell (1991). An experimental comparison of symbolic and connectionist learning algorithms. *Machine Learning 6(2)*, 111–143.

Simon, H. and G. Lea (1973). Problem solving and rule induction: A unified view. In Gregg (Ed.), *Knowledge and Cognition*, pp. 105–127. New Jersey: Lawrence Erlbaum Associates.

Smyth, P. and R. Goodman (1992). An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering 4*(4), 301–316.

Srikant, R. and R. Agrawal (1996). Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal.

Stolfo, S. (1998). http://www.cs.columbia.edu/~sal/JAM/PROJECT.

Stolfo, S., D. Fan, W. Lee, A. Prodromidis, and P. Chan (1997). Credit card fraud detection using meta-learning: Issues and initial results. In *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management (AAAI Technical Report WS-97-07)*, Menlo Park: CA, pp. 83–90. AAAI Press.

Stolfo, S., A. Prodromidis, S. Tselepis, D. Fan, W. Lee, and P. Chan (1997). JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the AAAI-97 Workshop on*

AI Approaches to Fraud Detection and Risk Management (AAAI Technical Report WS-97-07), Menlo Park: CA, pp. 91–98. AAAI Press.

Toivonen, H. (1996). Sampling large databases for association rules. In *Proceedings of the Twenty-fourth International Conference on Very Large Data Bases.*

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

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM 27*(11), 1134–1142.

Webb, G. (1995). OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research 3*, 383–417.

Weiss, S. M., R. S. Galen, and P. V. Tadepalli (1990). Maximizing the predictive value of production rules. *Artificial Intelligence 45*, 47–71.

Wettschereck, D., D. W. Aha, and T. Mohri (1997). A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *Artificial Intelligence Review 10*, 1–37. Also Technical Report AIC-95-012, Naval Research Laboratory.

Wettschereck, D. and T. G. Dietterich (1995). An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning 19*(1), 5–28.

Williams, G. (1990). *Inducing and Combining Multiple Decision Trees.* Ph. D. thesis, Australian National University, Canberra, Australia.

Wu, X. and W. H. Lo (1998). Multi-layer incremental induction. In *Proceedings of the Fifth Pacific Rim International Conference on Artificial Intelligence*, pp. 24–32. Springer-Verlag.

Zaki, M. (1998). *Scalable Data Mining for Rules.* Ph. D. thesis, Department of Computer Science, University of Rochester, Rochester, NY.

Zaki, M. J., C. Ho, and R. Agrawal (1999). Scalable parallel classification for data mining on shared memory multiprocessors. In *Proceedings of IEEE International Conference on Data Engineering.*

Zaki, M. J., S. Parthasarathy, W. Li, and M. Ogihara (1997). Evaluation of sampling for data mining of association rules. In *Proceedings of the Seventh International Workshop on Research Issues in Data Engineering.*