Efficient Progressive Sampling

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

David Jensen and Tim Oates
Computer Science Department
University of Massachusetts
Amherst, MA 01003-4610
jensen,oates@cs.umass.edu

March 5, 1999

Abstract

Having access to massive amounts of data does not necessarily imply that induction algorithms must use them all. *Samples* often provide the same accuracy with far less computational cost. However, the correct sample size is rarely obvious. We analyze methods for *progressive sampling*—starting with small samples and progressively increasing them as long as model accuracy improves. We show that a simple, geometric sampling schedule is efficient in an asymptotic sense. We then explore the notion of optimal efficiency: what is the absolute best sampling schedule? We describe the issues involved in instantiating an "optimally efficient" progressive sampler. Finally, we provide empirical results comparing a variety of progressive sampling methods. We conclude that progressive sampling often is preferable to analyzing all data instances.

Keywords: Scaling up, inductive learning, sampling, learning curves
Paper ID: 442

Contact author: Foster Provost
Email: provost@acm.org
Phone: (914) 644-2169
FAX: (914) 644-2237

1 Introduction

Induction algorithms face competing requirements for accuracy and efficiency. The requirement for accurate models often demands the use of large data sets which allow algorithms to discover subtle and complex structure. The requirement for efficient induction demands the use of small data sets, because the computational complexity of even the most efficient induction algorithms is linear in the number of instances, and most are considerably less efficient.

In this paper we study progressive sampling methods, which attempt to maximize accuracy as efficiently as possible. Progressive sampling starts with small samples and progressively increases their size until model accuracy no longer improves. A central component of progressive sampling is a *sampling schedule* $S = \{n_0, n_1, n_2, \ldots, n_k\}$ where $n_i < n_j$ for $i < j$. Each n_i specifies the size of a sample to be provided to an induction algorithm.

We show that schedules in which the n_i increase geometrically are efficient in an asymptotic sense. Next we explore the question of optimal efficiency in an absolute sense: what is the most efficient sampling schedule? We then investigate empirically how a variety of schedules perform on large benchmark data sets, and how they compare to using the entire data set. Next, we address a crucial practical issue: how can a method for progressive sampling determine that model quality no longer increases? We describe an interaction between the sampling schedule and the method of convergence detection, and we demonstrate a practical alternative that avoids the worst aspects of the tradeoffs this interaction requires. Finally, we discuss why progressive sampling is especially beneficial in cases where sampling from a large database is inefficient.

We conclude that, in a wide variety of realistic circumstances, progressive sampling is preferable to analyzing all instances from a database. Surprisingly, it can be competitive even with knowing the optimal sample size in advance.

2 Progressive Sampling

A *learning curve* (Figure 1a) depicts the relationship between sample size and model accuracy. The horizontal axis represents n, the number of instances in a given training set,

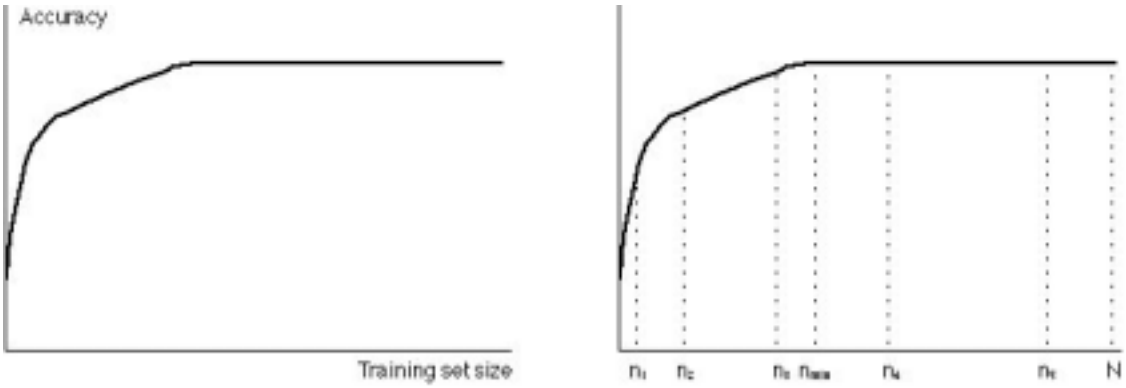

Figure 1: Learning curves and progressive samples

which can vary between zero and N, the total number of available instances. The vertical axis represents the accuracy of the model produced by an induction algorithm when given a training set of size n.

Learning curves typically have a steeply sloping portion early in the curve, a more gently sloping middle portion, and a plateau late in the curve. The gently sloping middle portion can be extremely large in some curves (e.g., (Catlett, 1991a, 1991b; Harris-Jones & Haines, 1997)) and almost entirely missing in others. The plateau occurs when adding additional data instances does not improve accuracy. The plateau, and even the entire middle portion, can be missing from curves when N is not sufficiently large. Conversely, the plateau region can constitute the majority of curves when N is very large. For example, in a recent study of two large business data sets, Harris-Jones and Haines (1997) found that learning curves reach a plateau quickly for some algorithms, but small accuracy improvements continue up to N for other algorithms.

When a learning curve reaches its final plateau, we say it has *converged*. We denote the training set size at which convergence occurs as n_{min}.

Definition 1 *Given a data set, a sampling procedure, and an induction algorithm, n_{min} is the size of the* smallest sufficient training set. *Models built with smaller training sets have lower accuracy, and models built with larger training sets have no higher accuracy, than models built with from training sets of size n_{min}.*

Compute schedule $S = \{n_0, n_1, n_2, \ldots, n_k\}$ of sample sizes
$n \leftarrow n_0$
$M \leftarrow$ model induced from n instances
while not *converged*
    recompute S if necessary
    $n \leftarrow$ next element of S larger than n
    $M \leftarrow$ model induced from n instances
end while
return M

Figure 2: Progressive sampling

Figure 1b shows an example sampling schedule and its relation to a learning curve. Empirical estimates are necessary to determine n_{min} because the precise shape of a learning curve represents a complex interaction between the statistical regularities present in a given data set and the abilities of an induction algorithm to identify and represent those regularities. In general, these characteristics are not known in advance, nor is their interaction well understood. Thus, n_{min} is nearly impossible to determine from theory.

Figure 2 is a generic algorithm that defines the family of progressive sampling methods. An instance of this family has particular methods for computing a schedule, for revising a schedule, and for determining convergence. In the next two sections, we present methods for computing and for revising schedules. We then analyze their efficiency empirically. We assume for these sections that progressive sampling is able to detect convergence and we assume that this detection can be performed efficiently (its worst-case run-time complexity is not worse than that of the underlying induction algorithm). In Section 6 we relax these assumptions. For this paper, we consider only drawing random samples from the larger data set. We believe that the results will generalize to other methods of sampling, but have not yet studied the general case.

3 Computing schedules

3.1 Arithmetic schedules

We know of one prior study of progressive sampling. John and Langley (1996) define a method we will call *arithmetic sampling*, using the schedule $S_a = n_0 + (i \cdot n_\delta) = \{n_0, n_0 + n_\delta, n_0 + 2n_\delta, \ldots, N\}$. For example, one arithmetic schedule is $\{100, 200, 300, \ldots, N\}$. John and Langley compare arithmetic sampling with the most distinct alternative—estimating n_{min} based on a subsample's statistical similarity to the entire sample—an approach they call *static sampling*. Their experiments show that progressive sampling produces more accurate models than does static sampling.

However, arithmetic sampling has an obvious drawback. If n_{min} is a large multiple of n_δ, then the approach will require very many runs of the underlying induction algorithm. For example, if $n_{min} = 200,000$ and $n_0 = n_\delta = 100$, then 2000 runs will be necessary—more than half with $n > 100,000$ instances. John and Langley partially escape this difficulty by specifying the use of an incremental induction algorithm (e.g., a simple Bayesian classifier), whose run time depends only on the additional instances, rather than having to reprocess all prior instances as well. Unfortunately, S_a can be extremely inefficient for the vast majority of induction algorithms, which are not incremental (as we show in section 5).

3.2 Geometric schedules

One way to escape the limitations of arithmetic sampling is to use progressive sampling with a geometric schedule. That is,

$$S_g = a^i \cdot n_0 = \{n_0, a \cdot n_0, a^2 \cdot n_0, a^3 \cdot n_0, \ldots, N\}$$

for some constants n_0 and a.[1] For example, one geometric schedule is $\{100, 200, 400, 800, \ldots, N\}$. We now show that geometric sampling is an *asymptotically optimal* schedule. That is, in terms of worst-case time complexity, geometric sampling is equivalent to knowing n_{min}.

[1] n_0 can be 1, but in practice will be larger to compensate for the fixed overhead of running the induction program.

Definition 2 *A progressive sampling schedule is asymptotically optimal if its worst-case, asymptotic run-time complexity is no worse than that of running the algorithm on the smallest sufficient training set.*

We assume that the asymptotic run-time complexity of induction with n instances, $C(n)$, is no better than $O(n)$. Since most inductive algorithms have $\Omega(n)$ run-time complexity, and many are strictly worse than $O(n)$, this assumption does not seem problematic. Under these assumptions, geometric sampling is asymptotically optimal.[2]

Theorem 1 *For inductive algorithms with time complexity $C(n)$ no better than $O(n)$, if convergence also can be detected in $C(n)$, then geometric sampling is an asymptotically optimal method for employing induction algorithms on large data sets.*

Proof: *Let the run-time complexity of induction (with the algorithm in question) be $C(n)$, where n is the size of the data set. As specified in Definition 1, let n_{min} be the size of the smallest sufficient training set. The run-time complexity of induction from the smallest sufficient training set is therefore $C(n_{min})$. Notably, the "optimal" run time grows with n_{min} rather than with the total number of instances N.*

By the definition of S_g, geometric sampling processes subsets of size $a^i \cdot n_0$ for $i = 0, 1, \ldots, k'$, assuming convergence is detected after k' samples, where $k' < k$. The run-time complexity of progressively running the induction algorithm on the k' subsets, including running the convergence-detection procedure, is $O(\sum_{i=0}^{k'} C(a^i \cdot n_0))$, which is $O(C(a^{k'} \cdot n_0)$, because C(n) is at best O(n).

Now, we assume convergence is well detected, so $a^{k'-1} \cdot n_0 < n_{min} \leq a^{k'} \cdot n_0$. This means that the overall time complexity is $O(C(a \cdot n_{min}))$, which is $O(C(n_{min}))$, so progressive sampling is asymptotically no worse than running the induction algorithm on the smallest sufficient training set. □

Because it is simple and asymptotically optimal, we propose that geometric sampling, for example with $a = 2$, is a good default schedule for mining large data sets. We discuss further

[2]We follow the reasoning of Korf, who shows that progressive deepening is an optimal schedule for conducting depth-first search when the smallest sufficient depth is unknown (Korf, 1985) (Provost, 1993).

reasons in Section 8, and provide empirical results testing this proposition in Section 5. First we explore further the topic of optimal schedules.

3.3 Computing optimal schedules with dynamic programming

A guarantee of asymptotic optimality is encouraging, but can we say anything about sampling schedules that are absolutely optimal? This section describes a method for identifying the schedule with the minimum expected cost of convergence. This seems a daunting task. For each value of n, from 1 to N, a model can either be built or not, leading to 2^N possible schedules. However, identification of the optimal schedule can be cast in terms of dynamic programming, yielding an algorithm that requires $O(N^2)$ space and $O(N^3)$ time.

Let $f(n)$ be the cost of building a model with n instances and determining whether accuracy has converged. Let $\Phi(n)$ be the probability that convergence requires more than n instances. Clearly, $\Phi(0) = 1$. If we let $n_0 = 0$, then the expected cost of convergence by following schedule S is given by the following equation:

$$C = \sum_{i=1}^{k} \Phi(n_{i-1}) f(n_i)$$

To better understand what this equation captures, consider a simple example in which $N = 10$ and $S = \{3, 7, 10\}$. The value of C is as follows:

$$C = \sum_{i=1}^{k} \Phi(n_{i-1}) f(n_i) = \Phi(0) f(3) + \Phi(3) f(7) + \Phi(7) f(10)$$

With probability 1 ($\Phi(0) = 1$), an initial model will be built with 3 instances at a cost of $f(3)$. If more than 3 instances are required for convergence, an event that occurs with probability $\Phi(3)$, a second model will be built with 7 instances at a cost of $f(7)$. Finally, with probability $\Phi(7)$ more than 7 instances are required for convergence and a third model will be built with all 10 instances at a cost of $f(10)$. The expected cost of a schedule is the simply the sum of the cost of building each model times the probability that the model will actually need to be constructed.

Schedule	Cost
$S_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$	121.0
$S_2 = \{10\}$	100.0
$S_3 = \{2, 6, 10\}$	72.8

Table 1: Expected costs of various schedules given $N = 10$, $f(n) = n^2$ and a uniform prior.

In many cases there may be no prior information about the likelihood of convergence occurring for any given n. In that case, assuming a uniform prior over all n yields $\Phi(n) = (N - n)/N$.[3] Consider another example in which $N = 10$, the uniform prior is used, and $f(n) = n^2$. The costs for three different schedules is shown in Table 1. The first schedule, in which a model is constructed for each possible data set size, is the most expensive of the three shown. The second schedule, in which a single model is built with all of the instances, is also not optimal in the sense of expected cost given a uniform prior. The third schedule shown has the lowest cost of all $2^{10} = 1024$ possible schedules for this problem.

Given N, f and Φ, we want to determine the schedule S that minimizes C. That is, we want to find the optimal schedule. As noted previously, a brute force approach to this problem would explore exponentially many schedules. We take advantage of the fact that optimal schedules are composed of optimal sub-schedules to apply dynamic programming to this problem, resulting in a polynomial time algorithm. Let $m[i, j]$ be cost of the optimal schedule for $i \leq n \leq j$ instances and that requires models to be constructed on $n = i$ instances and $n = j$ instances. The cost of the optimal schedule given a data set containing N instances is then $m[0, N]$. The following recurrence can be used to compute $m[0, N]$:

$$m[i, j] = \min \begin{cases} \Phi(i) f(j) \\ \min_{i<k<j} m[i, k] + m[k, j] \end{cases}$$

Both the bottom-up table-based and top-down memoized implementations of this equation require $O(N^2)$ space to store the values of m and $O(N^3)$ time.

The results of applying dynamic programming to determine the optimal schedules for three different values of N and various $f(n)$ are shown below in Tables 2 and 3. Both

[3]This implies that $\Phi(N) = 0$ and thus that $n_{min} < N$. We relax this assumption later in this section and in section 4.1.

$N = 100$		
$f(n)$	Schedule	Cost
n	{100}	100
$n^{1.5}$	{6, 50, 100}	847
n^2	{19, 62, 100}	7271
n^3	{3, 15, 41, 75, 100}	560,790

Table 2: Optimal schedules for $N = 100$ and various $f(n)$ given a uniform prior.

$N = 500$		
$f(n)$	Schedule	Cost
n	{500}	500
$n^{1.5}$	{27, 248, 500}	9,470
n^2	{10, 99, 312, 500}	181,775
n^3	{2, 17, 77, 204, 375, 500}	70,096,920

Table 3: Optimal schedules for $N = 500$ and various $f(n)$ given a uniform prior.

tables contain the optimal schedules for each $f(n)$ at a given value of N along with the costs associated with those schedule. These tables have several interesting features. First, note that for a given level of N, the optimal schedule is highly dependent on $f(n)$. In general, the larger the asymptotic complexity of $f(n)$ the more frequently the schedules indicate that models should be constructed. Second, although the interval between the n_i in any given schedule seems to be geometrically increasing, the multiplicative factor is by no means a constant. In fact, the factor seems to decrease dramatically near the end of the schedule.

As stated earlier, the running time of the algorithm that determines the optimal schedule is $O(N^3)$. This is clearly impractical for data sets with millions of instances. Fortunately, the running time actually is cubic in the number of data set sizes at which a model can be built, which can be a small fraction of N if one is willing to sacrifice precision in the placement of model construction points (for example, looking only at multiples of 100 or 1000 instances).

If there is information on the likely location of the minimum data set set size required for convergence, then schedules with much lower costs can be produced. For example, we know that learning curves almost always have the characteristic shape shown in Figure 1, and that in many cases $n_{min} \ll N$. Rather than assume a uniform prior, it would be more

$N = 500$		
$f(n)$	Schedule	Cost
n	{57, 143, 285, 500}	183
$n^{1.5}$	{36, 93, 180, 318, 500}	2,355
n^2	{16, 50, 108, 191, 318, 500}	33,473
n^3	{4, 23, 50, 93, 149, 231, 348, 500}	9,026,006

Table 4: Optimal schedules for $N = 500$ and various $f(n)$ given a log-normal prior.

reasonable to assume a more concentrated distribution, with low values for very small n, and low values for very large n. For example, data from Oates and Jensen (1997, 1998) show that the distribution of the number of instances needed for convergence over a large set of the UCI databases (Merz & Murphy, 1997) is roughly log-normal with a mean of 2000. Of the 18 datasets that reached convergence prior to N, four reached convergence between 100 and 1000 instances, 10 reached convergence between 1000 and 10,000, and 4 reached convergence between 10,000 and 100,000.

Non-uniform priors can strongly affect the schedules produced by dynamic programming. Tables 4 and 5 show optimal schedules for $N = 500$ and various $f(n)$ using priors different from those in the previous tables. Table 4 is based on a log-normal prior such that $\Phi(\log(x)) = 1 - N(x)$ where $N(x)$ is the cumulative density of a normal distribution with mean $\log(50)$ and standard deviation 1. Despite the fact that the schedules in this table call for more models to be constructed than schedules based on the uniform prior (see Table 3), the expected costs are lower because more precise information about the location of n_{min} is available. Table 5 is based on a uniform prior where $\Phi(n) = (1.25N - n)/1.25N$. That is, convergence is equally likely to occur for any data set size ranging from 1 to $1.25N$. The corresponding schedules call for fewer models to be constructed when compared to the case in which convergence is guaranteed with N instances (again, see Table 3). That is because there is a non-zero probability that more than N instances are required for convergence.

Using dynamic programming to compute the optimal schedule has several advantages over using a geometric schedule with a fixed factor a. First, the schedules produced via dynamic programming are optimal in a well-defined sense. Second, these schedules specify the data set size for which the first model should be built. Third, schedules produced via

$N = 500$		
$f(n)$	Schedule	Cost
n	{500}	500
$n^{1.5}$	{3, 144, 500}	10,329
n^2	{36, 212, 500}	208,850
n^3	{3, 28, 114, 286, 500}	8.8×10^7

Table 5: Optimal schedules for $N = 500$ and various $f(n)$ given a uniform prior out to $1.25N$.

dynamic programming determine, in a principled manner, the multiplicative constant to use between the n_i and that constant varies as needed.

We call progressive sampling based on an optimal schedule as determined by this procedure Dynamic Programming sampling, or DP sampling.

4 Revising schedules

Determining optimal schedules for DP sampling requires a model of the probability of convergence and a model of the run-time complexity of the underlying induction algorithm. In the previous section we assumed that these were known in advance, but our prior knowledge may be less than perfect. We now show that a progressive sampling procedure can build both of these models adaptively. The key insight is that a progressive sampling algorithm can obtain substantial amounts of information cheaply by including small samples in its schedule.

4.1 Modeling the probability of convergence

As we argued above, the assumption of uniform probabilities of convergence for all n is probably incorrect. However, progressive sampling algorithms can model the convergence probability dynamically. For example, a progressive sampling algorithm might assume that the accuracy of a particular algorithm on a particular data set can be modeled by a power law. A simple power law is shown by Frey and Fisher (1999) to model learning curves better than a variety of alternatives, and a similar approach is used by John and Langley (1996) to

determine convergence (see Section 6). Such an modeling approach could allow a progressive sampling procedure to adaptively improve the efficiency of its schedule during execution.

This enhances the benefits of geometric schedules that take many small samples early, when the learning curve has the most variation, but while running the induction algorithm is inexpensive and the impact of a suboptimal schedule is low. When the cost of running the induction algorithm increases, the model of the convergence probabilities will be much better, and thus the actual performance closer to the true optimal.

4.2 The cost of running the induction algorithm

The second assumption of DP sampling is that we have an accurate model of the run-time complexity (in n) of the underlying induction algorithm. Run-time complexity models are not always easy to obtain. For example, our empirical results below use the decision-tree algorithm C4.5 (Quinlan, 1993). The growth of C4.5 run time is often claimed to be linear in the number of instances, for non-numeric data sets. This claim is based on an analysis by Utgoff, where he shows the asymptotic time complexity to be $O(n)$ (Utgoff, 1989). With numeric data, sorting adds a $\log n$ term at each node.

However, C4.5 has been observed to be worse than $O(n^2)$ (Catlett, 1991a). One explanation for the discrepancy is that Utgoff actually shows that the complexity of C4.5 is $O(nt)$ where t is the number of tree nodes. Oates and Jensen (1997, 1998) have shown that the number of tree nodes often grows as $O(n)$. Therefore, the complexity for non-numeric data would be $O(n^2)$, for numerics $O(n^2 \log n)$, and for mixed-type data, somewhere in between.

In addition, DP sampling requires the *actual* run-time complexity for the problem in question, rather than the *worst-case* complexity. We obtained empirical estimates of the complexity of C4.5 on the LED and WAVEFORM data sets (used below), and found the former to be $O(n^{1.22})$ and the latter to be $O(n^{1.37})$.[4]

As with learning curve estimation, progressive sampling can dynamically determine the

[4] We follow a similar procedure to (Frey & Fisher, 1999) and assumed that the running time could be modeled by: $y = a * n^b$, gathered samples of CPU time required to build trees on 1,000 to 100,000 instances in increments of 1,000, then took the log of both the CPU time and the number of instances, ran linear regression, and used the resulting slope as an estimate of b.

actual run-time complexity as the sampling progresses. As before, early in the schedule, with small samples, suboptimal scheduling due to an incorrect time-complexity model will have little overall effect. As the samples grow and bad estimates would be costly, the time-complexity model becomes more accurate.

5 Empirical comparison of sampling schedules

We have shown that, in principle, progressive sampling can be optimal—both in an asymptotic and in an absolute sense. The goal of our empirical analysis is to evaluate whether progressive sampling can be used for practical scaling. We hypothesize that geometric sampling and DP sampling are considerably less expensive than using all the data when convergence is early, and not too much more expensive when convergence is late. We further hypothesize that, for large data sets and non-incremental algorithms, these versions of progressive sampling are significantly better than arithmetic progressive sampling. We also investigate how well simple geometric sampling compares with DP sampling, and how they each compare to the oracle-determined schedule $S_O = \{n_{min}\}$.

We compare progressive sampling with several different schedules: $S_N = \{N\}$, a single sample with all the instances; $S_O = \{n_{min}\}$, a single sample with the smallest sufficient training set determined by an omniscient oracle; $S_a = 100 + (100 \cdot i)$, arithmetic sampling with $n_0 = n_\delta = 100$; $S_g = 100 \cdot 2^i$, geometric sampling with $n_0 = 100$ and $a = 2$; and S_{dp}, DP sampling with schedule recomputation after dynamic estimation of priors and run-time complexity.

Of the three progressive sampling methods, only DP sampling revises its schedules. In order to determine the probability of convergence, with S_{dp} progressive sampling estimates the learning curve dynamically by applying linear regression to estimate the parameters of a power law. We also dynamically estimate $C(n)$, the actual time complexity of the underlying induction algorithm, under the assumption that $C(n) = a * n^b$, based on the observed run time on the samples taken so far. Progressive sampling with S_{dp} first builds models on 100, 200, 300, 400 and 500 instances to get an initial estimate of the learning curve and actual run-time complexity. From these, the method estimates the convergence probability distribution and the complexity of the algorithm. Then, as described in Figure 2 it iteratively checks for

Data set	Full	Arith	Geo	DP	Oracle
LED	16.40	1.77	0.55	0.56	0.18
CENSUS	16.33	59.68	5.57	5.41	2.08
WAVEFORM	425.31	1230.00	41.84	50.57	22.12

Table 6: Computation time for several progressive sampling methods

convergence, rebuilds the schedule by recomputing the distribution and the complexity with the latest information, and then produces a new classifier.

For the experiments in this section, we assume that convergence can be detected accurately and without cost. The progressive sampling algorithms each had access to a function that returns FALSE if $n < n_{min}$ and TRUE if $n \geq n_{min}$. The function could only be accessed as a way of testing convergence, not as a method of schedule construction. We address the challenges of practical convergence detection in section 6.

For the purposes of this experiment, we determined n_{min} empirically by analyzing the full learning curve for arithmetic sampling and applying a technique developed in (Oates & Jensen, 1997). The technique takes sets of three adjacent points on a learning curve, averages their accuracy, and then compares that accuracy with the accuracy on all N instances. We select the first set (the set furthest to the left of the learning curve) for which average accuracy is not less than 1% less than the accuracy on all instances. The middle point of that set is n_{min}.

For our experiments we used 3 large data sets from the UCI repository (Merz & Murphy, 1997): LED, WAVEFORM and CENSUS (adult). For LED and WAVEFORM we used 100,000 instances, and for CENSUS 32,000. Based on the technique above, we found that for LED $n_{min} = 2,000$, for WAVEFORM $n_{min} = 12,000$, and for CENSUS $n_{min} = 8,000$.

The results of the experiments are summarized in table 6, which shows running times in seconds averaged over 10 runs. Before each run the order of the instances was randomized.

6 Determining convergence

The key assumption behind all the progressive sampling procedures we have discussed is that convergence can be detected accurately and efficiently. While we present some preliminary

results below, we believe that convergence detection remains is an open problem on which significant research effort should be focused.

In their paper on arithmetic sampling, John and Langley outline how to model a learning curve as sampling progresses. From this learning curve, they estimate whether the accuracy that would be achieved with all instances ($acc(N)$) will be very different from the accuracy achieved with the current sample ($acc(n)$). To formalize this notion, they define the Probably Close Enough (PCE) criterion, modeled after the Probably Approximately Correct (PAC) criterion of Valiant (1984). The PCE criterion states that convergence is reached when $Pr((acc(N) - acc(n)) > \epsilon) \leq \delta$, where $acc(x)$ is the accuracy of the model that an algorithm produces after seeing x instances, ϵ refers to the maximum acceptable decrease in accuracy, and δ is a probability that the maximum accuracy difference will be exceeded on any individual run. John and Langley set ϵ to 2% and δ to 5%.

Convergence detection is essentially a statistical judgment, irrespective of the specific convergence criterion or the method to estimate whether than criterion has been met. For example, John and Langley estimate the parameters of a learning curve based on the cross-validated accuracy of models produced with arithmetically larger subsamples, thus allowing estimation of the accuracy difference.

However, statistical estimation of complete learning curves is fraught with difficulties. Actual learning curves often require a complex functional form to estimate accurately. The curve shown in figure 1a has three regions of behavior—a primary rise, a secondary rise, and a plateau. Most simple functional forms (e.g., those used by Frey and Fisher (1999) and John and Langley (1996)) generally cannot capture all three regions of behavior, often causing the estimated curves to converge too quickly or to never converge. Estimating convergence is generally more challenging than fitting earlier parts of the curve, and even fairly small errors are damaging to progressive sampling methods.

More importantly, the need for accurate statistical estimation is fundamentally at odds with the goal of progressive sampling. Statistical estimates of convergence will be aided by increasing the number of points in a schedule, but this directly impairs efficiency. Even worse, determining convergence is most aided by samples for which $n_i > n_{min}$, because these points will most assist the statistical determination that a plateau has been reached. Of course,

Data set	DP-LRLS	DP-Free
LED	72.33	72.91
CENSUS	84.87	85.38
WAVEFORM	77.20	76.36

Table 7: Mean accuracy for DP with LRLS and with optimal convergence detection

these are the very sample sizes that, for efficiency reasons, progressive sampling schedules should avoid.

The most promising approach we have yet identified for statistical convergence detection increases complexity by essentially a constant factor. Despite this, preliminary tests indicate it can approximate n_{min} with consistently high accuracy, though at the cost of significant increases in absolute run time. This method—*linear regression with local sampling* (LRLS)—begins at the latest scheduled sample size n_i and samples r additional points in the local neighborhood of n_i. These points are then used to estimate a linear regression line, whose slope is compared to zero. If the slope is sufficiently close to zero, convergence is detected. LRLS takes advantage of a common property of learning curves: the slope of the line tangent to the curve constantly decreases. If LRLS ever estimates that the slope is zero, it is unlikely to ever become non-zero for larger n.

6.1 Empirical analysis of convergence detection

We applied LRLS to estimate convergence of DP schedules. For these experiments, $r = 20$ and convergence was detected the first time that the 95% confidence interval on the slope of a regression line included zero. As before, each value represents the average of 10 runs where instances were randomized prior to each run.

LRLS identified convergence accurately. In most cases, LRLS correctly identified $n_{k'}$, and in nearly all other cases convergence was identified in a sample for which $n_i > n_{k'}$. In no data set was the mean accuracy at estimated convergence statistically distinguishable from the accuracy on n_{min} instances, the point of true convergence. Mean accuracies are shown in table 7.

However, LRLS has a large effect on absolute computation time. The additional sam-

Data set	Full	DP-LRLS	DP-Free	Oracle
LED	16.40	24.05	0.56	0.18
CENSUS	16.33	54.46	5.41	2.08
WAVEFORM	425.31	424.92	50.57	22.12

Table 8: Computation time for DP with LRLS and with free convergence detection

pling and executions of the induction algorithm created a large factor increase in the total computational cost. The computational cost for WAVEFORM would have been substantially lower except for a single run which took 1654 seconds. Without this run, the mean run time reduces to 288 seconds.

These costs could be reduced by more efficient estimation of convergence. In addition, the costs of running on the full data set would increase dramatically if N increased, while this would not affect the computational cost of any form of progressive sampling. Thus, as the total size of data sets increases, progressive sampling becomes more attractive.

7 When is progressive sampling desirable?

The experiments above indicate the potential for progressive sampling to be more efficient than other approaches, given particular algorithms and data sets and provided that efficient convergence detection method can be identifed. However, what are the general conditions under which progressive sampling is desirable compared full sampling? Are those conditions sufficiently general that progressive sampling could be used routinely?

We constructed a simple analytical model to answer these questions. The model uses three parameters: a, the ratio of N to the sample size at which convergence is detected ($a = N/n_b$); b, the number of schedules executed prior to detecting convergence; and c, the exponent of the cost function ($C(n) = n^c$). In the data sets used in our experiments above, $a_{led} = 50$, $a_{waveform} = 8.33$, and $a_{census} = 4$. Recall that $c_{led} = 1.22$ and $c_{waveform} = 1.37$.

Based on these parameters, we can define the conditions under which the computational cost of progressive sampling and the cost of using all N instances are equal. That is, the conditions under which:

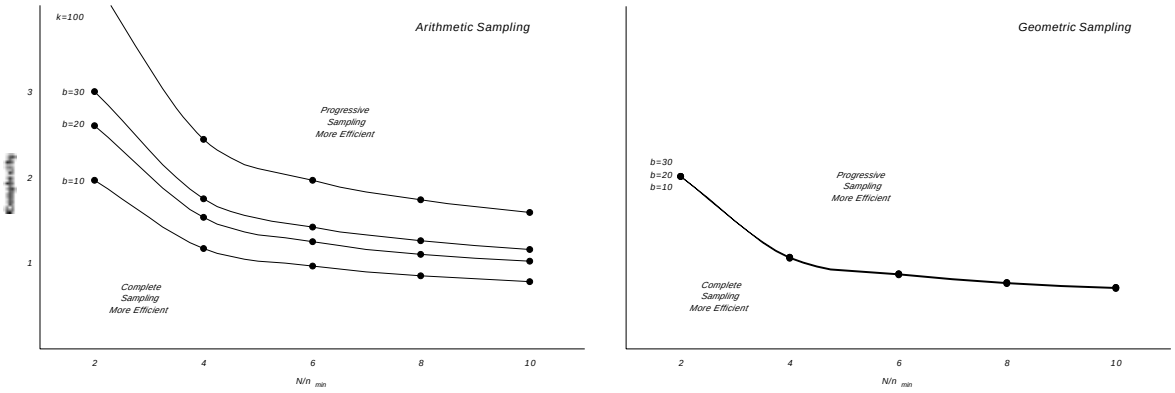

Figure 3: Regions of efficiency for arithmetic and geometric sampling

$$N^c = {n_0}^c + {n_1}^c + {n_2}^c + \ldots + {n_b}^c$$

where $a = N/n_b$ and the relationship among the elements of the partial schedule $\{n_0, n_1, n_2, \ldots, n_b\}$ are determined by the given progressive sampling method (e.g., arithmetic or geometric).

Sets of parameter values that satisfy the equation above are shown in figure 3. The curves show the boundaries between regions where it is more efficent to use progressive sampling and regions where it is more efficient to use all N instances. Above the curves, progressive sampling is more efficient; below the curves, complete sampling is more efficient. The right- and left-hand figures show curves for arithmetic and geometric sampling, respectively.

The curves show that progressive sampling will be efficient under a wide variety of circumstances. For example, arithmetic sampling that requires 30 samples to be tried before convergence is detected can still be efficient if the computational complexity is greater than n^2 and the final sample size is less than one-quarter of N. If, however, the final sample size changes to one-half of N, arithmetic sampling will not be efficient unless only 10 or fewer samples are needed prior to detecting convergence.

Geometric sampling is far more forgiving. If computational complexity is n_2 or worse, then geometric sampling is efficient as long as convergence can be detected with samples smaller than $N/2$. The number of samples prior to convergence is almost irrelevant. This is the essential feature of geometric sampling that makes it efficient. If a schedule contains

a large number of samples prior to the sample when convergence is detected, those samples will almost all be quite small. In contrast, increasing the number of samples in an arithmetic schedule adds samples along the schedule's entire range, and this results in a much larger computational cost.

8 Discussion

We have presented our final result as partially negative, but in doing so we downplay the benefits of progressive sampling. Our analysis assumes that sampling from a large database is instantaneous. Actually, as this assumption is relaxed, the relative efficiency of progressive sampling becomes better. Progressive sampling can take advantage of data as they arrive, effectively creating a pipelined induction process. For standard induction based on slow data access, the CPU sits idle and waits for the sampling to complete, and then runs the induction algorithm on the resultant data set. Progressive sampling can immediately get started on the first sample points, computing its first estimates of the learning curve and C(n). Thereafter, sampling first fills up a test-set buffer, so that when induction each subset is finished, the test-set buffer (containing data for the next subset) is used first to estimate the accuracy. Then the test set buffer can be shifted into the training buffer, and so on. Moreover, the slower the sampling, the more work can be done on convergence detection. With very slow sampling, the efficiency of progressive sampling will be the same as if n_{min} were known a priori.

9 Conclusion

With this work we have made substantial progress toward efficient progressive sampling. We have shown that if convergence detection can be done efficiently, then progressive sampling is far better than learning from all the data, and almost as efficient as being given the minimum sufficient training set by an oracle. We have also shown that convergence detection can be done effectively. What is left is a well-defined challenge for future KDD research: increase the efficiency of convergence detection.

We know of neither theoretical argument nor empirical evidence that is a relationship

between the size of a massive data set and the size of the minimum sufficient training set. In fact, although multi-gigabyte and even terabyte databases are becoming more common, we are not aware of evidence that any real-world induction task requires more than a million examples—and efficient induction on hundreds of thousands of examples is no longer unusual (Provost & Kolluri, 1999). The existence of an efficient progressive sampling procedure would take a giant step toward solving one of the basic challenges of KDD: classifier induction from massive data sets.

References

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.

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

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

John, G., & Langley, P. (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.

Korf, R. (1985). Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*, 97–109.

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

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

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

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*, pp. Menlo Park, CA 749-755. AAAI Press.

Provost, F., & Kolluri, V. (1999). A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*. To appear.

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

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.