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

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 rarely is obvious. We analyze methods for *progressive sampling*—using progressively larger samples as long as model accuracy improves. We explore several notions of efficient progressive sampling. We analyze efficiency relative to induction with all instances; we show that a simple, geometric sampling schedule is asymptotically optimal, and we describe how best to take into account prior expectations of accuracy convergence. We then describe the issues involved in instantiating an efficient progressive sampler, including how to detect convergence. Finally, we provide empirical results comparing a variety of progressive sampling methods. We conclude that progressive sampling can be remarkably efficient.

1 Introduction

Induction algorithms face competing requirements for accuracy and efficiency. The requirement for accurate models often demands the use of large data sets that allow algorithms to discover complex structure and make accurate parameter estimates. 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 algorithms 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 a small sample and uses progressively larger ones 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 each n_i is an integer that specifies the size of a sample to be provided to an induction algorithm. For $i < j$, $n_i < n_j$. If the data set contains N instances in total, $n_i \leq N$ for all i. There are three fundamental questions regarding progressive sampling.

1. What is an efficient sampling schedule?
2. How can convergence (i.e., that model quality no longer increases) be detected effectively and efficiently?
3. As sampling progresses, can the schedule be adapted to be more efficient?

We discuss several ways to assess efficiency, and how various progressive sampling procedures fare with respect to each. Notably, we show that schedules in which the n_i increase geometrically are optimal in an asymptotic sense. We explore the question of optimal efficiency in an absolute sense: what is the most efficient schedule given one's prior expectations of convergence? Next, we address the crucial practical issue of convergence detection. We describe an interaction between the sampling schedule and the method of convergence detection, and we describe a practical alternative that avoids the worst aspects of the tradeoffs this interaction requires. We also discuss algorithms that schedule adaptively, based on knowledge of convergence and actual run-time complexity, obtained on the fly.

We then investigate empirically how a variety of schedules perform on large benchmark data sets. 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 when the optimal sample size is known in advance.

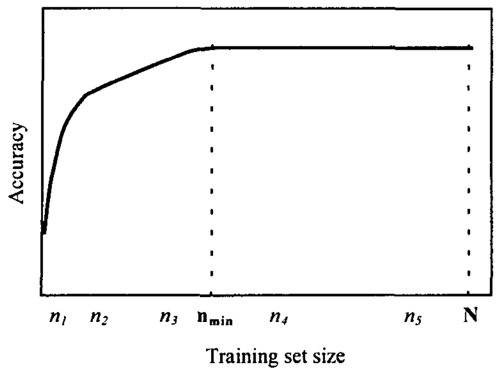

Figure 1: Learning curves and progressive samples

2 Progressive Sampling

A *learning curve* (Figure 1) depicts the relationship between sample size and model accuracy. The horizontal axis represents n, the number of instances in a given training set, that 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 middle portion can be extremely large in some curves (e.g., [2, 3, 6]) 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 [6] found that learning curves reach a plateau quickly for some algorithms, but small accuracy improvements continue up to N for other algorithms.

We assume that learning curves are well behaved. Specifically, we assume that the slope of a learning curve is monotonically non-increasing with n except for local variance. Locality is defined within a particular progressive sampling procedure.

Not all learning curves are well behaved. For example, theoretical analyses of learning curves based on statistical mechanics [7, 19] have shown that sudden increases in accuracy are possible, particularly on small samples. However, empirical studies of the application of standard induction algorithms to large data sets—those of relevance to this paper—have shown learning curves to be well behaved [3, 4, 6, 12, 13]. In addition, practical progressive sampling demands only that learning curves are well behaved at the level

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

of granularity of the sampling schedule. Given the relatively course granularity of many schedules, sudden increases in accuracy can occur without impairing progressive sampling. Our empirical results in section section 6 bear out these assumptions.

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 than models built with from training sets of size n_{min}, and models built with larger training sets have no higher accuracy.*

Figure 1 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, in many cases, n_{min} is nearly impossible to determine from theory. However, n_{min} can be approximated empirically by a progressive sampling procedure. We denote by $\hat{n}_{min}$ a procedure's approximation to n_{min}.

Figure 2 is a generic algorithm that defines the family of progressive sampling methods. An instance of this family has particular methods for selecting a schedule, for determining convergence, and for altering the schedule adaptively. The next three sections consider each of these methods in turn.

3 Determining an efficient schedule

We now discuss several alternative methods for selecting an efficient schedule. For now, we assume 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).

3.1 Simple schedules

For brevity, we call a schedule generated by a simple procedure a *simple schedule*. Several simple schedules are of interest. For example, later we will use for comparison the schedule composed of a single data set with all instances, $S_N = \{N\}$. We also will consider the simple schedule generated by an omniscient oracle, $S_O = \{n_{min}\}$.

John and Langley [8] define a progressive sampling approach we call *arithmetic sampling* using the schedule $S_a = n_0 + (i \cdot n_\delta) = \{n_0, n_0 + n_\delta, n_0 + 2n_\delta, \ldots, n_0 + k \cdot n_\delta\}$.[1] An example arithmetic schedule is $\{100, 200, 300, \ldots, n_k\}$. John and Langley compare arithmetic sampling with static sampling. *Static sampling* computes $\hat{n}_{min}$ without progressive sampling, based on a subsample's statistical similarity to the entire sample. For consistency, we consider static sampling as a degenerate progressive sampling procedure with schedule $S_{static} = \{\hat{n}_{min}\}$. John and Langley show that arithmetic sampling produces more accurate models than does static sampling. This is not surprising, given that n_{min} depends on the relationship between the data and the specific learning algorithm. In some cases, the difference between n_{min}'s for different learning algorithms is quite large [6].

Arithmetic sampling has an obvious drawback. If n_{min} is a large multiple of n_δ, then the approach will require 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. Unfortunately, S_a can be extremely inefficient for the vast majority of induction algorithms, which are not incremental. We compare arithmetic schedules to other approaches in section 6.

An alternative schedule escapes the limitations of arithmetic sampling. *Geometric sampling* uses the schedule S_g:

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

for some constants n_0 and a. An example geometric schedule is $\{100, 200, 400, 800, \ldots\}$. The next sections explain why geometric schedules are robust.

3.2 When are simple schedules efficient?

What are the general conditions under which simple schedules are more efficient than S_N? Are those conditions sufficiently general that simple progressive sampling should be used routinely?

Consider an underlying induction algorithm with a simple polynomial run-time complexity $f(n)$, and a simple analytical model with three parameters: b, the number of schedule points executed prior to the point of convergence (n_b is the first schedule point greater than $\hat{n}_{min}$); r, the ratio of the total number of instances to the size of the final sample ($r = \frac{N}{n_b}$); and c, the exponent of the run-time complexity ($f(n) = n^c$).

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:

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

where $r = \frac{N}{n_b}$ and where the relationship among the elements of the partial schedule $\{n_0, n_1, n_2, \ldots, n_b\}$ is 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 left- and right-hand figures show curves for arithmetic and geometric sampling, respectively.

The curves show that progressive sampling will be more efficient than learning with all the examples under a wide variety of circumstances. For example, arithmetic sampling that requires 30 samples to be tried before convergence is detected is better 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 is one-half of N, arithmetic sampling will not be better 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 better as long as convergence can be detected with samples smaller than $N/2$. If computational complexity is linear, then geometric sampling is better as long as convergence can be detected with samples smaller than $N/6$. The number of samples prior to convergence is nearly 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.

[1] The first and last points in the schedule, n_0 and n_k, depend on external factors such as N and the induction program's (fixed) computational overhead.

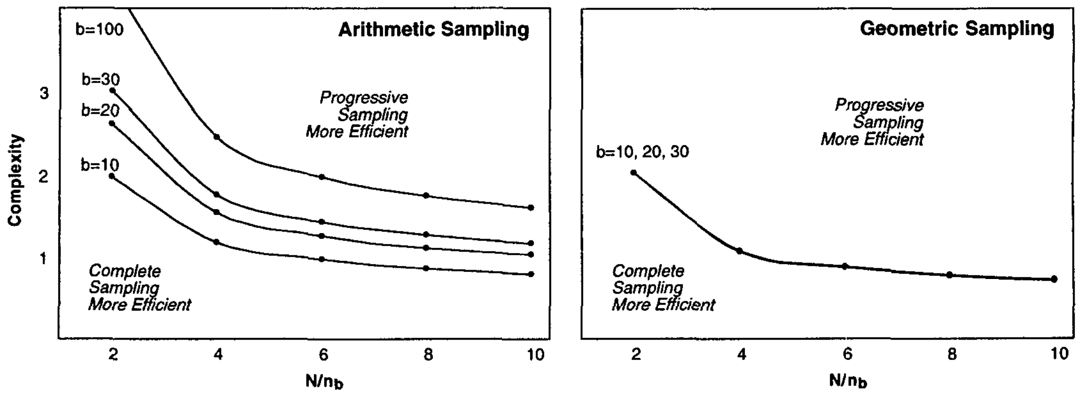

Figure 3: Regions of efficiency for arithmetic and geometric sampling

3.3 The asymptotic optimality of geometric schedules

How do simple schedules compare to the schedule of an omniscient oracle, $S_O = \{n_{min}\}$? Below we show that geometric sampling is an *asymptotically optimal* schedule. That is, in terms of worst-case time complexity, geometric sampling is equivalent to S_O.

For a given data set, let $f(n)$ be the (expected) run time of the underlying induction algorithm with n sampled instances. We assume that the asymptotic run-time complexity of the algorithm, $\Theta(f(n))$, is polynomial (no better than $O(n)$). Since most induction 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 induction algorithms with polynomial time complexity $\Theta(f(n))$, no better than $O(n)$, if convergence also can be detected in $O(f(n))$, then geometric progressive sampling is asymptotically optimal among progressive sampling methods in terms of run time.*

Proof: *The run-time complexity of induction (with the learning algorithm in question) is $\Theta(f(n))$, where n is the number of instances in the data set. As specified in Definition 1, let n_{min} be the size of the smallest sufficient training set. The ideal progressive sampler has access to an oracle that reveals n_{min}, and therefore will use the simple, optimal schedule $S_O = \{n_{min}\}$. The run-time complexity of S_O is $\Theta(f(n_{min}))$. Notably, the run time of the optimal progressive sampling procedure grows with n_{min} rather than with the total number of instances N.*

By the definition of S_g, geometric progressive sampling runs the induction algorithm on subsets of size $a^i \cdot n_0$ for $i = 0, 1, \dots, b$, where $b+1$ is the number of samples processed before detecting convergence.

Now, we assume convergence is well detected, so

$$a^{b-1} \cdot n_0 < n_{min} \leq a^b \cdot n_0 < a \cdot n_{min},$$

which means that

$$a^i \cdot n_0 < \frac{a^i}{a^{b-1}} \cdot n_{min},$$

for $i = 0, 1, \dots, b$. Since $O(f(\cdot))$ is at best linear, the run time of S_g (on all subsets, including running the convergence-detection procedure) is

$$O(f(\sum_{i=0}^{b} \frac{a^i}{a^{b-1}} \cdot n_{min})).$$

This is

$$O(f(a \cdot n_{min} \cdot (1 + \frac{1}{a} + \frac{1}{a^2} + \dots + \frac{1}{a^b})).$$

The final, finite sum is less than the corresponding infinite sequence. Because $a > 1$ for S_g, this converges to a constant that is independent of n_{min}. Since $O(f(\cdot))$ is polynomial, the overall run time of S_g is $O(f(n_{min}))$. Therefore, progressive sampling asymptotically is no worse than the optimal progressive sampling procedure, S_O, which runs the induction algorithm only 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 provide empirical results testing this proposition and discuss further reasons in section 6. First we explore further the topic of optimal schedules.

3.4 Optimality with respect to expectations of convergence

Comparisons with S_N and S_O represent two ends of a spectrum of expectations about n_{min}. Can optimal

[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 [9] [15].

schedules be constructed given expectations between these two extremes?

Prior expectations of convergence can be represented as a probability distribution. For example, let $\Phi(n)$ be the (prior) probability that convergence requires more than n instances. By using $\Phi(n)$, and the run-time complexity of the learning algorithm, $f(n)$, we can compare the expected cost of different schedules. Thus we can make statements about *expectation-based optimality*. The schedule with minimum expected cost is optimal in a practical sense.

In many cases there may be no prior information about the likelihood of convergence occurring for any given n. Assuming a uniform prior over all n yields $\Phi(n) = (N-n)/N$.[3] At the other end of the spectrum, oracle optimality can be cast as a special case of expectation-based optimality, wherein $\Phi(n) = 1$ for $n < n_{min}$, and $\Phi(n) = 0$ for $n \geq n_{min}$ (i.e., n_{min} is known).

The better the information on the likely location of n_{min}, the lower the expected costs of the schedules 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 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 [12, 13] show that the distribution of the number of instances needed for convergence over a large set of the UCI databases [10] is roughly log-normal.

Given such expectations about n_{min}, is it possible to construct 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 [1], 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. As described above, 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 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.

Consider another example in which $N = 10$; the uniform prior is used ($\Phi(n) = (N-n)/N$), and $f(n) = n^2$. The costs for three different schedules are 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, also is 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 the cost of the minimum expected-cost schedule of all samples in the size range $[i, j]$, with the requirement that samples of i and of j instances be included in $m[i, j]$. 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 $O(N^2)$

[3] This implies that $\Phi(N) = 0$ and thus that $n_{min} \leq N$.

$N = 500$		
$f(n)$	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 2: Optimal schedules for $N = 500$ and various $f(n)$ given a uniform prior.

$N = 500$		
$f(n)$	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 3: Optimal schedules for $N = 500$ and various $f(n)$ given a log-normal prior.

space to store the values of m and $O(N^3)$ time.

The results of applying dynamic programming to determine the optimal schedules for $N = 500$ and various $f(n)$ are shown in Table 2, which shows the optimal schedules for each $f(n)$ along with the associated costs. Note that the optimal schedule depends on $f(n)$. In general, the larger the complexity of $f(n)$, the more frequently the schedules indicate that models should be constructed. Second, although the n_i in any given schedule are quasi-geometrically increasing, the multiplicative factor is by no means a constant. In fact, the factor seems to decrease dramatically near the ends of the schedules.

As stated earlier, the running time of the algorithm that determines the optimal schedule is $O(N^3)$. This clearly is 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).

Non-uniform priors can strongly affect the schedules produced by dynamic programming. Table 3 shows optimal schedules 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 2), the expected costs are lower because more precise information about the location of n_{min} is available.

We call progressive sampling based on the optimal schedule determined by this dynamic programming procedure, DP sampling.

4 Detecting convergence

A key assumption behind all the progressive sampling procedures discussed above is that convergence can be detected accurately and efficiently. We present some preliminary results below, and we believe that convergence detection remains an open problem on which significant research effort should be focused.

Convergence detection is fundamentally a statistical judgment, irrespective of the specific convergence criterion or the method to estimate whether that criterion has been met. In their paper on arithmetic sampling, John and Langley [8] model the learning curve as sampling progresses. They determine convergence using a stopping criterion modeled after the work of Valiant [18]. Specifically, convergence is reached when $Pr((acc(N) - acc(n_i)) > \epsilon) \leq \delta$, 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. A model of the learning curve is used to estimate $acc(N)$.

Statistical estimates of convergence face several challenges. First, 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 1 has three regions of behavior—a primary rise, a secondary rise, and a plateau. Most simple functional forms (e.g., the power laws used by Frey and Fisher [4] and by John and Langley [8]) generally cannot capture all three regions of behavior, often causing the estimated curves to converge too quickly or never to converge. Estimating convergence is generally more challenging than fitting earlier parts of the curve, and even fairly small errors can mislead progressive sampling methods. For example, a power law may fit the early part of the curve well, but will represent a long, final plateau as a long (perhaps slight) incline.

More important, the need for accurate statistical estimates must be balanced against the goal of computational efficiency. Statistical estimates of convergence are aided by increasing the number of points in a schedule, but this directly impairs efficiency. Even worse, determining convergence is aided most 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, these are the very sample sizes that, for efficiency reasons, progressive sampling schedules should avoid.

The most promising approach we have yet identified for convergence detection—*linear regression with local sampling* (LRLS)—begins at the latest scheduled sample size n_i and samples l 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.

LRLS increases progressive sampling's complexity by a constant factor. In section 6 we show that it approximates n_{min} with consistently high accuracy, though at the cost of significant increases in absolute run time.

5 Adaptive scheduling

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.

5.1 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 [4] to model learning curves better than a variety of alternatives, and a similar approach is used by John and Langley [8] to determine convergence (see section 4). Such a modeling approach could allow a progressive sampling procedure to improve the efficiency of its schedule adaptively during execution.

This enhances the benefits of quasi-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 will be closer to the true optimal.

5.2 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 [17], for which reported time complexity varies widely.

Moreover, DP sampling requires the *actual* run-time complexity for the problem in question, rather than a *worst-case* complexity. We obtained empirical estimates of the complexity of C4.5 on the data sets used below, and found $O(n^{1.22})$ for LED, $O(n^{1.37})$ for WAVEFORM, and $O(n^{1.38})$ for CENSUS.[4]

As with learning curve estimation, progressive sampling can determine the actual run-time complexity dynamically 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.

6 Empirical comparison of sampling schedules

We have shown that, in principle, progressive sampling can be efficient. We now 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.

We compare progressive sampling with several different schedules: $S_N = \{N\}$, a single sample with all the instances; $S_O = \{n_{min}\}$, the optimal schedule 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

[4]We followed a similar procedure to Frey and Fisher [4] and assumed that the running time could be modeled by: $y = c_0 \cdot n^c$, 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 c.

sampling estimates the learning curve dynamically by applying linear regression to estimate the parameters of a power law. We also dynamically estimate $f(n)$, the actual time complexity of the underlying induction algorithm, under the assumption that $f(n) = c_0 \cdot n^c$, 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 convergence, rebuilds the schedule by recomputing the distribution and the complexity with the latest information, and then produces a new classifier.

For our first set of experiments, 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.

To instantiate the oracle and the costless convergence detection procedure, we determined n_{min} empirically by analyzing the full learning curve and applying a technique developed by Oates and Jensen [12]. The technique takes sets of three adjacent points on a learning curve (we used points an arithmetic schedule with $n_\delta = 1000$), averages their accuracies, and then compares that accuracy with the accuracy on all N instances. The oracle chooses the first set (the set furthest to the left of the learning curve) for which average accuracy is not less than 1% of 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 [10]: LED, WAVEFORM (with 10% noise), 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 $n_{min} = 8,000$.

Table 4 shows running times in seconds averaged over 10 runs on each of the three data sets. Before each run the order of the instances was randomized. All runs were performed on a 400MHz DEC alpha.

The results confirm our hypotheses regarding the relative efficiency of progressive sampling. Geometric progressive sampling is between three and thirty times faster than learning with all the data. It also is between three and thirty times faster than arithmetic progressive sampling. On the other hand, geometric progressive sampling is only two to three times slower than the oracle-optimal procedure, S_O.

Perhaps surprisingly, DP progressive sampling is not faster than geometric sampling. Constructing the DP schedule takes less than $\frac{1}{10}$th of a second, so the DP overhead is not responsible. DP would be faster with more precise knowledge of n_{min}. Indeed, non-adaptive DP with uniform priors consistently produces poor schedules on these data sets. This is because the time complexity of C4.5 is rather low ($O(n^{1.22})$ for LED $O(n^{1.37})$ for WAVEFORM, and $O(n^{1.38})$ for CENSUS) (cf. the schedule for linear complexity with uniform priors in Table 2), but these data sets have relatively large values for $r^* = N/n_{min}$; specifically, $r^*_{census} = 4$, $r^*_{waveform} = 8.33$, and $r^*_{led} = 50$. Adaptive modeling of the probability of convergence is critical, but apparently we must devise a more accurate technique if we want to beat geometric sampling.[5]

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 4: Computation time for several progressive sampling methods

Now we move on to the detection of convergence. We used LRLS to estimate convergence for adaptive DP sampling. For these experiments, at each schedule point, ten samples were taken for convergence estimation, $l = 10$, and convergence was indicated the first time that the 95% confidence interval on the slope of the regression line included zero. Table 5 compares the accuracy of the models built using LRLS convergence detection, with the accuracy using optimal convergence detection from above (i.e., it can query the oracle); each value represents the average of 10 runs where instances were randomized prior to each run.

LRLS identifies convergence accurately. In most cases, LRLS correctly identifies n_b as the first schedule point after n_{min}, and in nearly all other cases convergence is identified in a sample for which $n_b > n_{min}$. 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.

However, LRLS has a large effect on absolute computation time. The additional sampling and executions of the induction algorithm create a large (constant) factor increase in the total computational cost. Table 6 compares the run time with LRLS to running with "free" convergence detection, to running on the full data set, and to knowing n_{min} in advance.

[5] Experiments with increasingly precise knowledge of n_{min} show DP sampling to be less efficient than geometric sampling even for prior distributions centered on n_{min}, if the distributions are wide. Of course, if the distributions are very narrow, DP sampling comes very close to S_O, the optimal schedule.

Data set	DP-LRLS	DP-Free
LED	73.02	72.91
CENSUS	84.83	85.38
WAVEFORM	76.88	76.36

Table 5: Mean accuracy for DP with LRLS and with optimal convergence detection

Data set	Full	DP-LRLS	DP-Free	Oracle
LED	16.40	11.27	0.56	0.18
CENSUS	16.33	9.22	5.41	2.08
WAVEFORM	425.31	142.92	50.57	22.12

Table 6: Computation times with LRLS and with free convergence detection

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.

Our presentation does not highlight all the benefits of progressive sampling. For example, our analysis assumes that sampling from a large database is instantaneous. 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 $f(n)$. 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.

7 Other related work

The method of Musick et al. [11] for determining the best attribute at each decision-tree node can be seen as an instance of the generic progressive sampling algorithm shown in figure 2, if we regard each node of the decision tree as an individual induced model. Specifically, based on an analysis of information gain and its statistical properties, they compute an estimate of the sample size needed to have less than a specified loss in information. However, because this estimate can overshoot $\hat{n}_{min}$ greatly, they then calculate n_δ for an efficient arithmetic schedule, and revise the estimate after executing each schedule point. Other sequential multi-sample learning methods [14] are degenerate instances of progressive sampling, typically using fixed arithmetic schedules and treating convergence detection simplistically, if at all.

For this paper, we have considered 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. Methods for active sampling, choosing subsequent samples based upon the models learned previously, are of particular interest. A classic example of active sampling is windowing [16], wherein subsequent sampling chooses instances for which the current model makes errors. Active sampling changes the learning curve. For example, on noisy data, windowing learning curves are notoriously ill behaved: subsequent samples contain increasing amounts of noise, and performance often decreases as sampling progresses [5]. It would be interesting to examine more closely the use of the techniques outlined above in the context of active sampling, and the potential synergies.

8 Conclusion

With this work we have made substantial progress toward efficient progressive sampling. We have shown that if convergence detection can be done very 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 shown that convergence detection can be done effectively and moderately efficiently. We have also shown that geometric sampling is remarkably robust: its efficiency is insensitive to the number of points in the schedule (unlike arithmetic sampling); it is asymptotically no worse than knowing the point of convergence in advance, and in practice it performs as well as much more complicated adaptive scheduling.

What is left are two well-defined challenges for future KDD research: increase the efficiency of convergence detection, and devise an accurate method for estimating the point of convergence from a partial learning curve. One of the defining problems of KDD is classifier induction from massive data sets [14]. The existence of an efficient progressive sampling procedure would take a giant step toward solving it.

9 Acknowledgements

This research is supported in part by DARPA and AFOSRF under contract numbers DARPA/AFOSRF 49620-97-1-0485 and DARPAN66001-96-C-8504. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of DARPA, AFOSRF or the U.S. Government.

References

[1] BELLMAN, R. E. *Dynamic programming.* Princeton University Press, 1957.

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

[3] CATLETT, J. *Megainduction: Machine learning on very large databases.* PhD thesis, School of Computer Science, University of Technology, Sydney, Australia, 1991.

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

[5] FÜRNKRANZ, J. Integrative windowing. *Journal of Artificial Intelligence Research 8* (1998), 129–164.

[6] HARRIS-JONES, C., AND HAINES, T. L. Sample size and misclassification: Is more always better? Working Paper AMSCAT-WP-97-118, AMS Center for Advanced Technologies, 1997.

[7] HAUSSLER, D., KEARNS, M., SEUNG, H. S., AND TISHBY, N. Rigorous learning curve bounds from statistical mechanics. *Machine Learning 25* (1996), 195–236.

[8] JOHN, G., AND LANGLEY, P. Static versus dynamic sampling for data mining. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (1996), AAAI Press, pp. 367-370.

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

[10] MERZ, C. J., AND MURPHY, P. M. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1997.

[11] MUSICK, R., CATLETT, J., AND RUSSELL, S. Decision theoretic subsampling for induction on large databases. In *Proceedings of the Tenth International Conference on Machine Learning* (San Mateo, CA, 1993), Morgan Kaufmann, pp. 212–219.

[12] OATES, T., AND JENSEN, D. The effects of training set size on decision tree complexity. In *Machine Learning: Proceedings of the Fourteenth International Conference* (1997), D. Fisher, Ed., Morgan Kaufmann, pp. 254–262.

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

[14] PROVOST, F., AND KOLLURI, V. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery 2* (1999).

[15] PROVOST, F. J. 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, 1993), AAAI Press, pp. Menlo Park, CA.

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

[17] QUINLAN, J. R. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, California, 1993.

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

[19] WATKIN, T., RAU, A., AND BIEHL, M. The statistical mechanics of learning a rule. *Reviews of Modern Physics 65* (1993), 499–556.