Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 348 (2005) 3 – 14

Theoretical Computer Science

www.elsevier.com/locate/tcs

Sequential sampling techniques for algorithmic learning theory

Osamu Watanabe

Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo 152-8552, Japan

Abstract

A *sequential sampling algorithm* or *adaptive sampling algorithm* is a sampling algorithm that obtains instances sequentially one by one and determines from these instances whether it has already seen enough number of instances for achieving a given task. In this paper, we present two typical sequential sampling algorithms. By using simple estimation problems for our example, we explain when and how to use such sampling algorithms for designing *adaptive* learning algorithms.

Keywords: Sequential sampling; Adaptive sampling; Adaptive learning algorithm; Chernoff bound; Hoeffding bound

1. Introduction

Random sampling is an important technique in computer science for developing efficient randomized algorithms. A task such as estimating the proportion of instances with a certain property in a given data set can often be achieved by randomly sampling a relatively small number of instances. *Sample size*, i.e., the number of sampled instances, is a key factor for sampling, and for determining appropriate sample size, so-called concentration bounds or large deviation bounds have been used (see, e.g., [9]). In particular, the Chernoff bound [2] and the Hoeffding bound [14] have been used commonly in theoretical computer science because they derive a theoretically guaranteed sample size sufficient for achieving a given task with given accuracy and confidence. There are some cases, however, where these bounds can provide us with only overestimated or even unrealistic sample size. In this paper, we show that "sequential sampling algorithms" are applicable for some of such cases to design *adaptive* randomized algorithms with theoretically guaranteed performance.

A *sequential sampling algorithm* or *adaptive sampling algorithm* is a sampling algorithm that obtains instances sequentially one by one and determines from these instances whether it has already seen enough number of instances for achieving a given task. Intuitively, from the instances seen so far, we can more or less obtain some knowledge on the input data set, and it may be possible to estimate an appropriate sample size. Recently, we have proposed [7,8] a sequential sampling algorithm for a general hypothesis selection problem (see also [6] for some preliminary versions). Our main motivation was to scale up various known learning algorithms for practical applications such as data mining (see, e.g., discussions in [8]). While some applications and extensions of our approach towards this direction have been reported [1,4,20], it has been also noticed [3,5] that sequential sampling allows us to add "adaptivity" to learning algorithms while keeping their worst-case performance. In this paper, we use some simple examples and explain when and how to use sequential sampling for designing such adaptive learning algorithms.

E-mail address: watanabe@is.titech.ac.jp.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.09.003

The idea of "sampling on-line" is quite natural, and it has been studied in various contexts. First of all, statisticians made significant accomplishments on sequential sampling during World War II [21]. In fact, from their activities, a research area on sequential sampling—sequential analysis—has been formed in statistics. Thus, it may be quite likely that some of the algorithms explained here have been already found in their contexts. (For recent studies on sequential analysis, see, e.g., [10,11].) In computer science, sequential sampling techniques have been studied in the database community. Lipton and Naughton [17] and Lipton et al. [16] proposed adaptive sampling algorithms for estimating query size in relational databases. Later Haas and Swami [13] proposed an algorithm that performs better than the Lipton–Naughton algorithm in some situations. More recently, Lynch [18] gave a rigorous analysis to the Lipton–Naughton algorithm. Roughly speaking, the spirit of sequential sampling is to use instances observed so far for reducing a current and future computational task. This spirit can be found in some of the learning algorithms proposed in machine learning community. For example, the Hoeffding race proposed by Maron and Moore [19] attempts to reduce a search space by removing candidates that are determined hopeless from the instances seen so far. A more general sequential local search has been proposed by Greiner [12].

All the above approaches more or less share the same motivation. That is, they attempt to design "adaptive algorithms" that can make use of the advantage of the situation to reduce sample size (or in general, computation time) whenever such reduction is indeed possible. We believe that some of these approaches can be formally discussed so that we can propose *adaptive* learning algorithms with theoretically guaranteed performance.

This paper has some overlap with the author's previous survey paper on sequential sampling [22].

2. Our problem and statistical bounds

In this paper, we fix one simple estimation problem for our basic example, and discuss sampling techniques on this problem or its variations. Let us specify our problem. Let D be an input data set; here it is simply a set of instances. Let B be a Boolean function defined on instances in D. That is, for any $x \in D$, $B(x)$ takes either 0 or 1. Our problem is to estimate the probability p_B that $B(x) = 1$ when x is given at random from D; in other words, the ratio of instances x in D such that $B(x) = 1$ holds.

Clearly, the probability p_B can be computed by counting the number of instances x in D for which $B(x) = 1$ holds. In fact, this is only the way if we are asked to compute p_B *exactly*. But we consider the situation where D is *huge* and it is impractical to go through all instances of D for computing p_B. A natural strategy that we can take in such a situation is random sampling. That is, we pick up some instances of D randomly and estimate the probability p_B on these selected instances. Without seeing all instances, we cannot hope for computing the exact value of p_B. Also due to the "randomness", we cannot always obtain a desired answer. Therefore, we must be satisfied if our sampling algorithm yields a *good approximation* of p_B with *reasonable probability*. In this paper, we will discuss this type of approximate estimation problem.

Our estimation problem is completely specified by fixing an "approximation goal" that defines the notion of "good approximation". We consider the following one for our first approximation goal. (In the following, we will use $\widetilde{p_B}$ to denote the output of a sampling algorithm (for estimating p_B); thus, it is a random variable and the probability below is taken w.r.t. this random variable.)

Approximation Goal 1 (*Absolute error bound*). *For given* $\delta > 0$ *and* ε, $0 < \varepsilon < 1$, *the goal is to have*

$$\Pr[|\widetilde{p_B} - p_B| \leqslant \varepsilon] > 1 - \delta. \tag{1}$$

As mentioned above, the simplest sampling algorithm for estimating p_B is to pick up instances of D randomly and estimate the probability p_B on these selected instances. Fig. 1 gives the precise description of this simplest sampling algorithm, which we call *Batch Sampling* algorithm. Here the only assumption we need (for using the statistical bounds explained below) is that we can easily pick up instances from D uniformly at random and independently.

The description of Batch Sampling algorithm of Fig. 1 is still incomplete since we have not specified the way to determine n, the number of iterations or sample size. Of course, to get an accurate estimation, the larger the n the better; on the other hand, for the efficiency, the smaller the n the better. We would like to achieve a given accuracy with as small sample size as possible.

Batch Sampling
begin
  $m \leftarrow 0$;
  for n **times do**
    get x uniformly at random from D;
    $m \leftarrow m + B(x)$;
  output m/n as an approximation of p_B;
end.

Fig. 1. Batch Sampling.

To determine appropriate sample size, we can use several statistical bounds, upper bounds of the probability that a random variable deviates far from its expectation. Here, we explain the Hoeffding bound [14] and the Chernoff bound [2] that have been used in computer science. (In practice, the bound derived from the Central Limit Theorem gives a better (i.e., smaller) sample size. But the Central Limit Theorem holds only asymptotically, and, furthermore, the difference is within a constant factor. Thus, it is omitted here (see, e.g., [9,22]).)

For explaining these bounds, let us prepare some notations. Let $X_1, \ldots, X_n$ be independent trials, which are called *Bernoulli trials*, such that, for $1 \leqslant i \leqslant n$, we have $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for some p, $0 < p < 1$. Let X be a random variable defined by $X = \sum_{i=1}^{n} X_i$. Then its expectation $E[X] = np$; hence, the expected value of X/n is p. The following two bounds, respectively, give an upper bound of the probability that X/n differs from p, say, ε.

Remarks on Notations: In order to distinguish absolute and relative error bounds, we will use, throughout this paper, symbols ε and ε for absolute and relative error bounds, respectively. We use below $\exp(x)$ to denote e^x, where e is the base of the natural logarithm.)

Theorem 1 (*The Hoeffding Bound [14]*). *For any* ε, $0 < \varepsilon < 1$, *we have the following relations*:

$$\Pr\left[\frac{X}{n} > p + \varepsilon\right] \leqslant \exp(-2n\varepsilon^2), \quad \Pr\left[\frac{X}{n} < p - \varepsilon\right] \leqslant \exp(-2n\varepsilon^2).$$

Theorem 2 (*The Chernoff Bound [2]*). *For any* ε, $0 < \varepsilon < 1$, *we have the following relations*:

$$\Pr\left[\frac{X}{n} > (1+\varepsilon)p\right] \leqslant \exp\left(-\frac{pn\varepsilon^2}{3}\right), \quad \Pr\left[\frac{X}{n} < (1-\varepsilon)p\right] \leqslant \exp\left(-\frac{pn\varepsilon^2}{2}\right).$$

By using these bounds, we calculate "safe" sample size, the number n of examples, so that Batch Sampling satisfies our approximation goals. Here we consider Goal 1, i.e., bounding the absolute estimation error. It is easy to prove that the following bounds work. (The proof is easy and it is omitted.)

Theorem 3. *For any* $\delta > 0$ and ε, $0 < \varepsilon < 1$, *if Batch Sampling uses sample size n satisfying one of the following inequalities*, *then it satisfies bound* (1).

$$n > \frac{1}{2\varepsilon^2} \ln\left(\frac{2}{\delta}\right), \tag{2}$$

$$n > \frac{3p_B}{\varepsilon^2} \ln\left(\frac{2}{\delta}\right). \tag{3}$$

This theorem shows that the simplest sampling algorithm, Batch Sampling, can be used to achieve the Approximation Goal 1 with a reasonable sample size. Let us see how the above (sufficient) sample size grows depending on given parameters. In both bounds (2) and (3), n grows proportional to $1/\varepsilon^2$ and $\ln(1/\delta)$. Thus, it is costly to reduce the (absolute) approximation error. On the other hand, we can reduce the error probability (i.e., improve the confidence) quite a lot without increasing the sample size so much.

3. Absolute error vs. relative error

For another typical approximation goal, we consider the following one.

Approximation Goal 2 (*Relative Error Bound*). *For given $\delta > 0$ and ε, $0 < \varepsilon < 1$, the goal is to have*

$$\Pr[|\widetilde{p_B} - p_B| \leqslant \varepsilon p_B] > 1 - \delta. \tag{4}$$

Here again we try our Batch Sampling algorithm to achieve this goal. Since the Chernoff bound is stated in terms of relative error, it is immediate to obtain the following sample size bound. (We can get a similar but less efficient sample size bound by using the Hoeffding bound.)

Theorem 4. *For any $\delta > 0$ and ε, $0 < \varepsilon < 1$, if Batch Sampling uses sample size n satisfying the following inequality, then it satisfies* (4).

$$n > \frac{3}{\varepsilon^2 p_B} \ln\left(\frac{2}{\delta}\right). \tag{5}$$

The above size bound is similar to (3). But it does not seem easy to use because p_B, the probability what we want to estimate, is in the denominator of the bound (cf. In the case of (3), we can safely assume that $p_B = 1$). Nevertheless, there are some cases where a relative error bound is easier to use and the above size bound (5) provides a better analysis to us. We show such examples below.

We consider some variations of our estimation problem. First one is the following problem:

Problem 1. *Let $\delta_0 > 0$ be any constant and fixed. For a given p_0, determine (with confidence $> 1 - \delta_0$) whether $p_B > p_0$ or not. We may assume that either $p_B > 3p_0/2$ or $p_B < p_0/2$ holds.*

That is, we would like to "approximately" compare p_B with p_0. Note that we do not have to answer correctly when $p_0/2 \leqslant p_B \leqslant 3p_0/2$ holds.

First we use our sample size bound (2) for Approximation Goal 1. It is easy to see that the requirement of the problem is satisfied if we run Batch Sampling algorithm with sample size n_1 computed by using $\varepsilon = p_0/2$ and $\delta = \delta_0$, and compare the obtained $\widetilde{p_B}$ with p_0. That is, we can decide (with high confidence) that $p_B > p_0$ if $\widetilde{p_B} > p_0$ and $p_B < p_0$ otherwise. Note that the sample size n_1 is $2c/p_0^2$, where $c = \ln(2/\delta_0)$.

On the other hand, by using the sample size bound (5), we can take the following strategy. Let $n_2 = 48c/p_0$, the sample size computed from (5) with $\varepsilon = 1/2$, $p_B = p_0/2$, and $\delta = \delta_0$, where $c = \ln(2/\delta_0)$ as above. Run Batch Sampling with this n_2 and let $\widetilde{p_B}$ be the obtained estimation. Then compare $\widetilde{p_B}$ with $3p_0/4$. We can prove that with probability $1 - \delta_0$, we have $p_B > p_0$ if $\widetilde{p_B} \geqslant 3p_0/4$ and $p_B < 3p_0$ otherwise.

Comparing two sample size n_1 and n_2, we note that $n_1 = \mathrm{O}(1/p_0^2)$ and $n_2 = \mathrm{O}(1/p_0)$; that is, n_2 is asymptotically better than n_1. One reason for this difference is that we could use large ε (i.e., $\varepsilon = 1/2$) for computing n_2.

Next consider the problem of estimating the product probability. Instead of estimating one probability p_B, we consider here a sequence of probabilities $p_1, \ldots, p_T$, where each p_t is defined as the probability that $B_t(x)$ holds for instance x randomly chosen from its domain D_t. Now our problem is to estimate their product $P_T = \prod_{t=1}^{T} p_t$ within a given absolute error bound. That is, the following problem:

Problem 2. *Let $\delta_0 > 0$ be any constant and fixed. For a given ε_0, obtain an estimation $\widetilde{P_T}$ of P_T such that*

$$\Pr[|\widetilde{P_T} - P_T| \leqslant \varepsilon_0] > 1 - \delta_0. \tag{6}$$

This is a simplified version of the problem solved by Kearns and Singh [15] for approximating some Markov decision process, and the following improvement is due to Domingo [3].

We may assume that, for each t, $1 \leqslant t \leqslant T$, it is easy to pick up instances from D_t uniformly at random and independently. Thus, by using Batch Sampling, we can get an approximate estimation $\widetilde{p_t}$ of each p_t. Here again we use sample size bounds for two approximation goals.

The strategy used by Kearns and Singh [15] is essentially based on the bound (2) for Approximation Goal 1. Their algorithm is outlined as follows:

Step 1: Check whether there is some t, $1 \leqslant t \leqslant T$, such that $p_t < \varepsilon_0$. (For this, we can use almost the same strategy used for solving Problem 1 by Batch Sampling; we omit the detail.) If $p_t < \varepsilon_0$, then we can simply estimate $\widetilde{P_T} = 0$, which satisfies the requirement because $P_T \leqslant \varepsilon_0$.

Step 2: Otherwise, for some ε specified later and $\delta = \delta_0/T$, compute the sample size n_1 for achieving Goal 1 by Batch Sampling. (Let us use bound (2) here.) Run Batch Sampling algorithm with sample size n_1 on D_t for each t, $1 \leqslant t \leqslant T$, getting the estimate $\widetilde{p_t}$ of p_t. Then obtain the estimation $\widetilde{P_T}$ of P_T as the product of all $\widetilde{p_t}$.

Here, we show that the obtained $\widetilde{P_T}$ satisfies the desired accuracy and confidence; we also estimate the number of examples used in this algorithm. (We only consider Step 2 and omit the analysis of Step 1 that is essentially the same as the one we did for Problem 1.)

It follows from our choice of n_1 that the following holds with probability $1 - \delta_0$. (We also have a lower bound inequality, which can be treated symmetrically.)

$$\widetilde{P_T} = \prod_{t=1}^{T} \widetilde{p_t} \leqslant \prod_{t=1}^{T} (p_t + \varepsilon).$$

But since $p_t \geqslant \varepsilon_0$, we have

$$\prod_{t=1}^{T} (p_t + \varepsilon) \leqslant \prod_{t=1}^{T} p_t \left(1 + \frac{\varepsilon}{\varepsilon_0}\right) = \left(1 + \frac{\varepsilon}{\varepsilon_0}\right)^T \prod_{t=1}^{T} p_t = \left(1 + \frac{\varepsilon}{\varepsilon_0}\right)^T P_T.$$

Then by letting $\varepsilon = \varepsilon_0^2/(2T)$, we have the desired bound, i.e., $\widetilde{P_T} \leqslant P_T + \varepsilon_0$.

Next consider the number N_1 of examples used at Step 2. Let $c = \ln(T/\delta_0)$, then N_1 is estimated as follows. (Precisely speaking, c is not constant but $\mathrm{O}(\ln T)$ even if we assume that δ_0 is constant, but we omit this small factor for the simplicity.)

$$N_1 = T \cdot n_1 = T \cdot ((c/2) \cdot (2T/\varepsilon_0^2)^2) = 2c \cdot (T^3/\varepsilon_0^4).$$

Now let us see that the argument becomes much simpler if we compute sample size using bound (5) for Approximation Goal 2. The outline of algorithm is similar to the above. In fact, Step 1 is exactly the same; thus, we state only Step 2.

Step 2: For some ε specified later and $\delta = \delta_0/T$, compute the sample size n_2 for achieving Goal 2 by Batch Sampling. For computing n_2 with bound (5), we need to know p_t (for each t); but, we can simply use ε_0 for p_t since we can assume here that $p_t \geqslant \varepsilon_0$ for all t. Again run Batch Sampling algorithm with sample size n_2 on each D_t, $1 \leqslant t \leqslant T$, to get estimate $\widetilde{p_t}$ of p_t. Then obtain the estimation $\widetilde{P_T}$ of P_T as the product of all $\widetilde{p_t}$.

To see that this algorithm solves Problem 2, we first note that the following holds with probability $1 - \delta_0$.

$$\widetilde{P_T} = \prod_{t=1}^{T} \widetilde{p_t} \leqslant \prod_{t=1}^{T} p_t(1 + \varepsilon) = (1 + \varepsilon)^T \prod_{t=1}^{T} p_t = (1 + \varepsilon)^T P_T.$$

Then the accuracy condition is satisfied by letting $\varepsilon = \varepsilon_0/(2T)$.

On the other hand, the number N_1 of examples used at Step 2 is estimated as follows. (Here, c is the same as the one used for estimating N_1.)

$$N_2 = T \cdot n_2 = T \cdot ((3c/\varepsilon_0) \cdot (2T/\varepsilon_0)^2) = 12c \cdot (T^3/\varepsilon_0^3).$$

Thus, $N_1 = \mathrm{O}(T^3/\varepsilon_0^4)$ and $N_2 = \mathrm{O}(T^3/\varepsilon_0^3)$. That is, N_2 is asymptotically better than N_1.

4. Adaptive sampling for bounding the relative error

In the previous section, we have seen some examples such that we can design an asymptotically better algorithm by bounding the relative error (instead of the absolute error) in the approximation problem. On the other hand, for computing the size bound (5), we need to know p_B or its appropriate lower bound, which is not easy in some cases. Even if we can use a lower bound p_0 for p_B, the actual p_B may be usually much larger than p_0, and we almost always

Adaptive Sampling
begin
  $m \leftarrow 0$; $n \leftarrow 0$;
  while $m < A$ **do**
    get x uniformly at random from D;
    $m \leftarrow m + B(x)$; $n \leftarrow n + 1$;
  output m/n as an approximation of p_B;
end.

Fig. 2. Adaptive Sampling.

have to use unnecessarily large sample sets. For example, for solving Problem 2 in the previous section, we used ε_0 for p_t (for each t) because we could assume that $p_t \geqslant \varepsilon_0$. This gives us the sample size $n_2 = \mathrm{O}(T^2/\varepsilon_0^3)$. But among all p_t, $1 \leqslant t \leqslant T$, if many of them are much larger than ε_0, then this sample size is unnecessarily big.

One way to avoid this problem is to perform presampling. By running our sampling algorithm, e.g., Batch Sampling, with small sample size and obtain some "rough" estimate of p_B. Although it may not be a good approximation of p_B, we can use it to determine appropriate sample size for main sampling. This is the strategy often suggested in statistics textbooks. But further note that we do not have to separate presampling and main sampling. In the course of sampling, we can improve our knowledge on p_B; why don't we use it! This is the key idea of our "adaptive sampling" techniques. Technically, what we need for implementing this approach is a stopping condition that determines whether it has already seen enough number of examples by using the current estimation of p_B.

Lipton et al. [16,17] realized this intuitive idea and proposed adaptive sampling algorithms for query size estimation and related problems for relational database. Our approximate estimation of p_B is a special case of estimating query sizes. Thus, their algorithm is immediately applicable to our problem. (On the other hand, the proof presented here is for the special case, and it may not be used to justify the original adaptive sampling algorithm proposed by Lipton et al., see [18].)

Fig. 2 is the outline of the adaptive sampling algorithm of [16]. Though it is simplified, the adaptive sampling part is essentially the same as the original one. As we can see, the structure of the algorithm is simple. It runs until it sees more than A examples x with $B(x) = 1$.

To complete the description of the algorithm, we have to specify the way to determine A. Here we use the Chernoff bound and derive the following formula for computing A. (The proof is given in Appendix.)

Theorem 5. *For any* $\delta > 0$ and ε, $0 < \varepsilon < 1$, *if Adaptive Sampling uses the following* A, *then it satisfies* (4) *with probability* $> 1 - \delta$.

$$A > \frac{3(1+\varepsilon)}{\varepsilon^2} \ln\left(\frac{2}{\delta}\right).$$

Furthermore, with probability $> 1 - \delta/2$, *the sample size* n *satisfies*

$$n \leqslant \frac{3(1+\varepsilon)}{(1-\varepsilon)\varepsilon^2 p_B} \ln\left(\frac{2}{\delta}\right). \tag{7}$$

Compare the sample size given by (5) and (7). Since ε is usually small, the difference is within some constant factor. That is, the sample size of this Adaptive Sampling algorithm is almost optimal; it is almost the same as the best case where the precise p_B is given. Therefore, if our target algorithm is designed with the bound (5) for Goal 2, then we can add "adaptivity" to the algorithm without (drastically) changing the worst-case performance of the algorithm. For example, consider the previous Problem 2 of estimating the product probability P_T. We can modify the second strategy by replacing Batch Sampling with Adaptive Sampling. Then with some small constant $c' > 0$, new sample size N_3 is (with high probability) bounded by

$$N_3 \leqslant c' \cdot (12c) \cdot (T^3/(p_0\varepsilon_0^2)),$$

where $p_0 \geqslant \varepsilon_0$ is a lower bound for $p_1, \ldots, p_T$. In the worst case (i.e., $p_0 = \varepsilon_0$), $N_3 = \mathrm{O}(T^3/\varepsilon_0^3)$, which is the same order as N_2. On the other hand, if the situation is favorable and p_0 is large, say, $p_0 > 1/2$, then N_3 gets decreased

and we have $N_3 = O(T^3/\varepsilon_0^2)$. That is, we could add "adaptivity" to our new strategy without changing the worst-case performance.

5. Adaptive sampling for general utility functions

We have seen two ways for estimating p_B within either an absolute or a relative error bound. But in some applications, we may need the other closeness conditions, or in more general, we might want to estimate not p_B but some other "utility function" computed from p_B. For example, let us consider the following problem:

Problem 3. *Let $\delta_0 > 0$ be any constant and fixed. Determine (with confidence $> 1 - \delta_0$) whether $p_B > 1/2$ or not. Here we may assume that either $p_B > 1/2 + \sigma_0$ or $p_B < 1/2 - \sigma_0$ holds for some σ_0.*

Problem 3 is similar to Problem 1, but these two problems have different critical points. That is, Problem 1 gets harder when p_0 gets smaller, whereas Problem 3 gets harder when σ_0 gets smaller. In other words, the closer p_B is to 1/2, a more accurate estimation is necessary, and hence more samples are needed. Thus, for solving Problem 3, what we want to estimate is not p_B itself but the following value:

$$u_B = p_B - \frac{1}{2}.$$

More specifically, the above problem is easily solved if the following approximation goal is achieved. (In the following, we use $\widetilde{u_B}$ to denote the output of a sampling algorithm for estimating u_B. Note that u_B is not always positive.)

Approximation Goal 3. *For given $\delta > 0$ and ε, $0 < \varepsilon < 1$, the goal is to have*

$$\Pr[|\widetilde{u_B} - u_B| \leqslant \varepsilon |u_B|] > 1 - \delta. \tag{8}$$

The problem of estimating u_B arises when implementing "boosting" techniques in some framework. Boosting is a technique to improve accuracy of a given (weak) learning algorithm by running it several times by modifying distribution over the instance space. For this modification, it is necessary (in some boosting framework) to estimate the "advantage" of an obtained hypothesis, h, which is defined as $p_h - 1/2$, where p_h is the probability that h answers correctly on randomly generated instance under the current distribution. It is possible to estimate this advantage by Batch Sampling. But as before, Batch Sampling requires the worst-case sample size, which is not good if the current hypothesis h has large advantage. That is, by using Batch Sampling, we would loose some "adaptivity" in the boosting process. This problem can be solved by using the adaptive sampling algorithm that we will explain here. (In fact, this is one of the motivations for us to develop this adaptive sampling algorithm [4].)

For achieving Approximation Goal 3, sequential sampling is again helpful. One might want to modify our previous Adaptive Sampling algorithm for this new approximation goal. For example, by replacing its while-condition "$m < A$" with "$m - n/2 < B$" and by choosing B appropriately, we may be able to satisfy the new approximation goal. Unfortunately, though, this naive approach does not seem to work. In the previous case, the stopping condition (i.e., the negation of the while-condition "$m < A$") was monotonic; that is, once $m \geqslant A$ holds at some point, this condition is unchanged even if we keep sampling. On the other hand, even if $m - n/2 \geqslant B$ holds at some point, the condition may be falsified later if we keep sampling. Due to this nonmonotonicity, one of the key lemmas for proving Theorem 5 (i.e., Lemma A.2) does not hold.

Fortunately, we can deal with this nonmonotonicity by using a slightly more complicated stopping condition. In Fig. 3, we state an adaptive sampling algorithm for Approximation Goal 3. Note that the algorithm does not use any information on u_B; hence, we can use it without knowing u_B at all. On the other hand, as shown in the following theorem, the algorithm estimates u_B with the desired accuracy and confidence. (The proof outline is given in Appendix.)

Theorem 6. *For any $\delta > 0$ and ε, $0 < \varepsilon < 1$, Nonmonotonic Adaptive Sampling satisfies* (8). *Furthermore, with probability more than $1 - \delta$, we have*

$$\textit{sample size} \lesssim \frac{2(1+2\varepsilon)^2}{(\varepsilon|u_B|)^2} \ln\left(\frac{1}{\varepsilon\delta|u_B|}\right).$$

Nonmonotonic Adaptive Sampling
begin
$m \leftarrow 0;\ n \leftarrow 0;$
$u \leftarrow 0;\ \alpha \leftarrow \infty;$
while $|u| < \alpha(1 + 1/\varepsilon)$ **do**
get x uniformly at random from D;
$m \leftarrow m + B(x);\ \ n \leftarrow n + 1;$
$u \leftarrow m/n - 1/2;$
$\alpha \leftarrow \sqrt{(1/2n)\ln(n(n+1)/\delta)};$
output u as an approximation of u_B;
end.

Fig. 3. Nonmontonic Adaptive Sampling.

To see the advantage of Nonmonotonic Adaptive Sampling, let us go back Problem 3 and solve it using this algorithm.

Note first that Problem 3 is solvable by Batch Sampling; we can use one of the bounds (2) and (3) for achieving Approximation Goal 1 by Batch Sampling. For example, let n'_1 be the sample size computed by using bound (2) with $\varepsilon = \sigma_0$ and $\delta = \delta_0$. Then we can determine $p_B > 1/2$ (with the desired confidence) if the estimate $\widetilde{p_B}$ obtained by running Batch Sampling with n'_1 is larger than $1/2$. Here the sample size n'_1 is $O(1/\sigma_0^2)$.

On the other hand, to use our Nonmonotonic Adaptive Sampling for Problem 3, we need to execute the algorithm with $\varepsilon = 1/2$ and $\delta = \delta_0$. Then with probability $\geqslant 1 - \delta_0$, the algorithm yields $\widetilde{u_B}$ of u_B within relative error bound $\varepsilon = 1/2$. Recall that $|u_B| > \sigma_0$ (from the condition given in Problem 3). Hence, if $u_B > 0$ (resp., $u_B < 0$), then it must hold that $\widetilde{u_B} > \sigma_0/2$ (resp., $\widetilde{u_B} < -\sigma_0/2$). Thus, we can decide $p_B > 1/2$ (i.e., $u_B > 0$) if $\widetilde{u_B} > 0$, and $p_B < 1/2$ (i.e., $u_B < 0$) otherwise. That is, Problem 3 is solvable with confidence $> 1 - \delta_0$. On the other hand, it follows from the above theorem that the sample size is about $O(1/|u_B|^2)$ with high probability. (Here we ignore the $\ln(1/|u_B|)$ factor.)

Now compare these two sample size n'_1 and n'_2. Since we can assume that $|u_B| > \sigma_0$, the sample size $n'_2 = O(1/|u_B|^2)$ is, in the worst case, essentially the same as the sample size $n'_1 = O(1/\sigma_0^2)$ for Batch Sampling. That is, our algorithm enjoys "adaptivity" while keeping almost the same worst-case sample size. Recall the difference between the sample size n_1 and n_2 we have seen at Problem 1. One reason that n_2 is asymptotically smaller than n_1 is that we could use a relatively large ε for computing n_2, because Approximation Goal 2 was suitable for Problem 1. Here again, Approximation Goal 3 is suitable for Problem 3; thus, we could choose large constant, i.e., 1/2, for ε.

6. Concluding remarks

We have seen some examples of sequential sampling algorithms and the way they are used for designing adaptive algorithms. For our explanation, we have used a very simple probability estimation problem and its variations, but there are many other interesting problems we can solve by using sequential sampling algorithms. For example, we have originally developed sequential sampling algorithms for selecting nearly optimal hypothesis [8], and some extension of our hypothesis selection technique has been also reported in [20].

Although only a simple utility function is considered, we may be able to use various functions defined on one or more estimated probabilities. For example, estimating the entropy or some pseudoentropy function by some sequential sampling technique is an interesting and practically important problem. In our general sampling algorithm [8], we have only considered utility functions that can be approximated by some linear function, because otherwise sample size may become very large. Since the entropy function does not belong to this function family, we need to find some way to bound sample size to a reasonable level.

Acknowledgements

This paper is based on a series of joint works [6–8] with Carlos Domingo and Ricard Gavaldà. I have learned a lot from these talented researchers. In particular, I thank Carlos for supplying me with information on related works for

preparing this manuscript. I would like to thank Professor Akahira and Professor Lynch for discussion and giving me pointers to related works. This work is supported in part by Grant-in-Aid for Scientific Research on Priority Areas (Discovery Science), 1998–2000, the Ministry of Education, Science, Sports and Culture.

Appendix A. Proof outlines of Theorems 5 and 6

Here for the reader's convenience, we give proof outlines of Theorems 5 and 6. For the details and for more general results, see the original papers [16] (also [18]) for Theorem 5 and [8] for Theorem 6.

A.1. Theorem 5

Let any $\delta > 0$ and ε, $0 < \varepsilon < 1$, be fixed, and let A be any number satisfying the condition of the theorem for ε and δ. Also in the following discussion, let t denote the number of execution of the while-iterations until Adaptive Sampling halts. In other words, the algorithm has seen t examples and then the while-condition breaks. (In the following, we simply call this situation "the algorithm halts at the tth step".) Note that t is a random variable that varies depending on the examples drawn from D. Let m_t and p_t denote the value m and m/n when the algorithm halts at the tth step.

Since the while-condition breaks at the tth step, it holds that $A \leqslant m_t$. On the other hand, $m_t < A + 1$ holds because the while-condition holds before the tth step. Hence we have $A/t \leqslant p_t < (A + 1)/t$. Here in order to simplify our discussion, we assume that $p_t \approx A/t$. In fact, we will see below (Lemma A.1) that t is larger than $1/(\varepsilon^2 p_B)$ with high probability; thus, the difference $(A + 1)/t - A/t (= 1/t)$ is negligible compared with the error bound εp_B. Now assuming $p_t \approx A/t$, it is easy to see that p_t is within the desired range $[(1 - \varepsilon)p_B, (1 + \varepsilon)p_B]$ (i.e., $|p_t - p_B| \leqslant \varepsilon p_B$) if and only if

$$\frac{A}{(1+\varepsilon)p_B} \leqslant t \leqslant \frac{A}{(1-\varepsilon)p_B}$$

holds for t. Therefore, the theorem follows from the following two lemmas. (Recall that t is a random variable, and the probabilities below are taken w.r.t. this random variable.)

Lemma A.1. $\Pr[t < A/((1 + \varepsilon)p_B)] < \delta/2$.

Lemma A.2. $\Pr[t > A/((1 - \varepsilon)p_B)] < \delta/2$.

Notice that t is also the number of examples used; thus, the sample size bound (7) is immediate from Lemma A.2.

Proof of Lemma A.1. We would like to estimate the above probability, and for this purpose, we want to regard the values $B(x) \in \{0, 1\}$ of chosen examples x as the result of Bernoulli trials so that we can use the statistical bounds explained in Section 2. There is, however, one technical problem. These statistical bounds are valid for fixed number of trials, i.e., examples in this case. On the other hand, the number of examples t itself is a random variable. Here we can get around this problem by arguing in the following way.

Let $t_0 = \lfloor A/((1 + \varepsilon)p_B) \rfloor$. Then our goal is to bound the probability that the algorithm halts within t_0 steps. (For the simplicity, let us assume that t_0 is strictly smaller than $A/((1 + \varepsilon)p_B)$.) Now we modify our algorithm so that it *always* sees exactly t_0 examples. That is, this new algorithm just ignores the while-condition and repeats the while-iteration exactly t_0 times. Consider the situation that the original algorithm does halt at the tth step for some $t \leqslant t_0$. Then we have $m_t \geqslant A$ at the tth step. Though the algorithm stops here, if we continued the while-iteration after the tth step, we would clearly have $m_{t_0} \geqslant A$ at the t_0th step. That is, $\widetilde{m_0} \geqslant A$ in the modified algorithm, where $\widetilde{m_0}$ denotes the value of m when the modified algorithm halts (after seeing t_0 examples). Conversely, if $\widetilde{m_0} \geqslant A$ in the modified algorithm, then the original algorithm (under the same situation) should have halted within t_0 steps. From this observation, we have

$$\begin{aligned}&\Pr[A \text{ halts at the } t\text{th step for some } t \leqslant t_0]\\ &\quad = \Pr[\widetilde{m_0} \geqslant A \text{ in the modified algorithm}].\end{aligned}$$

On the other hand, the modified algorithm always sees t_0 examples; that is, it is Batch Sampling. Thus, we can use the Chernoff bound to analyze the right-hand side probability. By our choice of m_{t_0} and A, it is easy to prove that the right-hand side probability is at most $\delta/2$. Thus, the desired bound is proved. □

One remark on the proof. The reason that we could argue by considering only the t_0th step is because the stopping condition "$m \geqslant A$" is *monotonic*. This property does not hold when we consider more general approximation goals like Goal 3.

Proof of Lemma A.2. Let $t_1 = \lceil A/((1-\varepsilon)p_B)\rceil$. We want to bound the probability that the algorithm does not halt after the t_1th step. Again we consider the modification of the algorithm; here we modify the algorithm so that it sees exactly t_1 examples. Then as before, we can show that the original algorithm does not halt at the t_1th step if and only if $\widetilde{m_1} < A$, where $\widetilde{m_1}$ is the value of m when the modified algorithm halts. Thus, it suffices to bound $\Pr[\widetilde{m_1} < A]$ by $\delta/2$, which is again provable by using the Chernoff bound. □

B.1. Theorem 6

The proof outline is basically the same as the one for Theorem 5.

Again let t be a random variable whose value is the step when the algorithm terminates. For any $k \geqslant 1$, we use u_k and α_k to denote, respectively, the value of u and α at the kth step. Define t_0 and t_1 by

$$t_0 = \min_k\{\alpha_k \leqslant \varepsilon|u_B|\}, \quad \text{and} \quad t_1 = \min_k\{\alpha_k \leqslant \varepsilon|u_B|/(1+2\varepsilon)\}.$$

Since α_k decreases monotonically in k, both t_0 and t_1 are uniquely determined, and $t_0 \leqslant t_1$.

We first show that if $t_0 \leqslant t \leqslant t_1$, that is, if the algorithm halts no earlier than the t_0th step nor later than the t_1th step, then its output u_t is in the desired range.

Lemma A.3. *If $t_0 \leqslant t \leqslant t_1$, then we have $|u_t - u_B| \leqslant \varepsilon|u_B|$ with probability $> 1 - \delta/(2t_0)$.*

Next we show, in the following two lemmas, that with reasonable probability the algorithm halts between the t_0th and t_1th steps. Then Theorem 6 follows from these lemmas.

Lemma A.4. $\Pr[t < t_0] < \delta(1 - 1/t_0)$.

Lemma A.5. $\Pr[t > t_1] < \delta/(2t_0)$.

Proof of Lemma A.3. Since the algorithm stops at the tth step, the while-condition holds at the $(t-1)$th step; that is, we have $|u_{t-1}| < \alpha_{t-1}(1 + 1/\varepsilon)$. Here t is large enough so that we may assume that the difference between $|u_t|$ and $|u_{t-1}|$ and the difference between α_t and α_{t-1} are both negligible. Then we have

$$|u_t| \lesssim \alpha_t\left(1+\frac{1}{\varepsilon}\right) \leqslant \alpha_{t_0}\left(1+\frac{1}{\varepsilon}\right) \leqslant \varepsilon|u_B|\left(1+\frac{1}{\varepsilon}\right) = (1+\varepsilon)|u_B|, \tag{9}$$

where the last inequality is from the choice of t_0.

Similarly by using $t \leqslant t_1$ and assuming that $\alpha_{t_1} \approx \varepsilon|u_B|/(1+2\varepsilon)$, we can prove that

$$|u_t| \gtrsim (1-\varepsilon)|u_B|. \tag{10}$$

Now if u_t and u_B have the same sign, then the lemma follows from (9) and (10). On the other hand, if u_t and u_B have different signs, then they must be quite different because $|u_t| \geqslant \alpha_t(1 + 1/\varepsilon)$; the difference is far enough for us to prove, by using the Hoeffding bound, that such a situation occurs with probability $< \delta/(t(t+1)) \leqslant \delta/(2t_0)$ (since $t \geqslant t_0 \geqslant 1$). □

Proof of Lemma A.4. In order to bound $\Pr[t < t_0]$, we first consider, for any k, $1 \leqslant k < t_0$, the modification of the algorithm that sees exactly k examples. Let $\widetilde{u_k}$ denote the value of u when the modified algorithm halts (after seeing k examples), and let P_k denote the probability $\Pr[|\widetilde{u_k}| \geqslant \alpha_k(1 + 1/\varepsilon)]$. Note that if the original algorithm halts at the kth step, then we have $|\widetilde{u_k}| \geqslant \alpha_k(1 + 1/\varepsilon)$ in the modified algorithm. (The converse does not always hold.) Thus, the probability that the original algorithm halts at the kth step (i.e., $\Pr[t = k]$) is bounded by P_k.

Consider any k, $1 \leqslant k < t_0$. From the choice of t_0, we have $\alpha_k > \varepsilon|u_B|$ because $k < t_0$. Thus, we have

$$P_k = \Pr\left[|\widetilde{u_k}| \geqslant \alpha_k\left(1 + \frac{1}{\varepsilon}\right)\right] \leqslant \Pr[|\widetilde{u_k}| > |u_B| + \alpha_k].$$

This means that $P_k \leqslant \Pr[\widetilde{u_k} > u_B + \alpha_k]$ if $\widetilde{u_k} \geqslant 0$, and $P_k \leqslant \Pr[\widetilde{u_k} < u_B - \alpha_k]$ otherwise. Both probabilities are bounded by using the Hoeffding bound in the following way. (Here we only state the bound for the former case, but the latter case can be treated similarly.)

$$P_k \leqslant \Pr[\widetilde{u_k} > u_B + \alpha_k] = \Pr\left[\sum_{i=1}^{k} X_i/n - \frac{1}{2} > p_B - \frac{1}{2} + \alpha_k\right] \leqslant \exp(-2\alpha_k^2 k) = \frac{\delta}{k(k+1)}.$$

Then summing up these bounds, we have

$$\Pr[t < t_0] \leqslant \sum_{k=1}^{t_0-1} P_k \leqslant \delta\left(1 - \frac{1}{t_0}\right). \qquad \square$$

Proof of Lemma A.5. Again we consider the modification of the algorithm that sees exactly t_1 examples, and let $\widetilde{u_1}$ denote the value of u when it terminates. Note that if the original algorithm does not halt within t_1 steps (which is in fact equivalent to the event that the algorithm does not halt at the t_1th step), then we have $\widetilde{u_1} < \alpha_{t_1}(1 + 1/\varepsilon)$. On the other hand, by using the Chernoff bound, we can show that the probability $\Pr[\widetilde{u_1} < \alpha_{t_1}(1 + 1/\varepsilon)]$ is at most $\delta/(2t_0)$. $\square$

References

[1] J. Balcazár, personal communication.
[2] H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, Ann. Math. Statist. 23 (1952) 493–509.
[3] C. Dominogo, Faster near-optimal reinforcement learning: adding adaptiveness to the E3 algorithm, in: Proc. of 10th Algorithmic Learning Theory Conf. ALT'99, Lecture Notes in Artificial Intelligence, Vol. 1720, Springer, Berlin, 1999, pp. 241–251.
[4] C. Domingo, O. Watanabe, Scaling up a boosting-based learner via adaptive sampling, in: Proc. of Knowledge Discovery and Data Mining, PAKDD'00, Lecture Notes in Artificial Intelligence, Vol. 1805, Springer, Berlin, 2000, pp. 317–328.
[5] C. Domingo, O. Watanabe, MadaBoost: a modification of AdaBoost, in: Proc. of 13th Annu. Conf. on Computational Learning Theory, COLT'00, Morgan Kaufmann, Los Altos, CA, 2000, pp. 180–189.
[6] C. Domingo, R. Gavaldà, O. Watanabe, Practical algorithms for on-line selection, in: Proc. of the 1st Internat. Conf. on Discovery Science, Lecture Notes in Artificial Intelligence, Vol. 1532, Springer, Berlin, 1998, pp. 150–161.
[7] C. Domingo, R. Gavaldà, O. Watanabe, Adaptive sampling methods for scaling up knowledge discovery algorithms, in: Proc. of the 2nd Internat. Conf. on Discovery Science, Lecture Notes in Artificial Intelligence, Vol. 1721, Springer, Berlin, 1999, pp. 172–183.
[8] C. Domingo, R. Gavaldà, O. Watanabe, Adaptive sampling methods for scaling up knowledge discovery algorithms, J. Knowledge Discovery Data Mining 6 (2002) 131–152.
[9] W. Feller, An Introduction to Probability Theory and its Applications, third ed., Wiley, New York, 1968.
[10] B.K. Ghosh, P.K. Sen (Eds.), Handbook of Sequential Analysis, Marcel Dekker, New York, 1991.
[11] B.K. Ghosh, M. Mukhopadhyay, P.K. Sen, Sequential Estimation, Wiley, New York, 1997.
[12] R. Greiner, PALO: a probabilistic hill-climbing algorithm, Artif. Intell. 84 (1996) 177–204.
[13] P. Haas, A. Swami, Sequential sampling procedures for query size estimation, IBM Research Report No. RJ 9101 (80945), 1992.
[14] W. Hoeffding, Probability inequalities for sums of bounded random variables, J. Amer. Statist. Assoc. 58 (1963) 15–30.
[15] M. Kearns, S. Singh, Near-optimal reinforcement learning in polynomial time, in: Proc. of the 16th Internat. Conf. on Machine Learning, ICML'98, Morgan Kaufmann, Los Altos, CA, 1998, pp. 260–268.
[16] R.J. Lipton, J.F. Naughton, D.A. Schneider, S. Seshadri, Efficient sampling strategies for relational database operations, Theoret. Comput. Sci. 116 (1993) 195–226.
[17] R.J. Lipton, J.F. Naughton, Query size estimation by adaptive sampling, J. Comput. System Sci. 51 (1995) 18–25.

[18] J.F. Lynch, Analysis and application of adaptive sampling, in: Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS'99), ACM Press, New York, 1999, pp. 260–267.
[19] O. Maron, A. Moore, Hoeffding races: accelerating model selection search for classification and function approximation, in: Advances in Neural Information Processing Systems, Morgan Kaufmann, Los Altos, CA, 1994, pp. 59–66.
[20] T. Scheffer, S. Wrobel, A sequential sampling algorithm for a general class of utility criteria, in: Proc. of the 6th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, ACM Press, New York, 2000, pp. 330–334.
[21] A. Wald, Sequential Analysis, Wiley, New York, 1947.
[22] O. Watanabe, Simple sampling techniques for discovery science, IEICE Trans. Inform. Systems E-83D (1) (2000) 19–26.