Data Mining and Knowledge Discovery, 6, 131–152, 2002

Adaptive Sampling Methods for Scaling Up Knowledge Discovery Algorithms

CARLOS DOMINGO*
Department of Mathematics and Computer Sciences, Tokyo Institute of Technology, Tokyo, Japan

RICARD GAVALDÀ
Department of LSI, Universitat Politècnica de Catalunya, Barcelona, Spain

OSAMU WATANABE†
Department of Mathematics and Computer Sciences, Tokyo Institute of Technology, Tokyo, Japan

Editors: Mannila, Liu, Motoda

Received September 18, 1999; Revised January 15, 2001

Abstract. Scalability is a key requirement for any KDD and data mining algorithm, and one of the biggest research challenges is to develop methods that allow to use large amounts of data. One possible approach for dealing with huge amounts of data is to take a random sample and do data mining on it, since for many data mining applications approximate answers are acceptable. However, as argued by several researchers, random sampling is difficult to use due to the difficulty of determining an appropriate sample size. In this paper, we take a sequential sampling approach for solving this difficulty, and propose an adaptive sampling method that solves a general problem covering many actual problems arising in applications of discovery science. An algorithm following this method obtains examples sequentially in an on-line fashion, and it determines from the obtained examples whether it has already seen a large enough number of examples. Thus, sample size is not fixed a priori; instead, it *adaptively* depends on the situation. Due to this adaptiveness, if we are not in a worst case situation as fortunately happens in many practical applications, then we can solve the problem with a number of examples much smaller than required in the worst case. We prove the correctness of our method and estimates its efficiency theoretically. For illustrating its usefulness, we consider one concrete task requiring sampling, provide an algorithm based on our method, and show its efficiency experimentally.

Keywords: data mining, knowledge discovery, scalability, adaptive sampling, concentration bounds

1. Introduction

Scalability is a key requirement for any knowledge discovery and data mining algorithm. It has been previously observed that many well known machine learning algorithms do not scale well. Therefore, one of the biggest research challenges is to develop new methods that allow to use machine learning techniques with large amount of data.

*Partially supported by EU Science and Technology Fellowship.
†Partially supported by the Ministry of Education, Science, Sports and Culture of Japan, Grant-in-Aid for Scientific Research on Priority Areas (Discovery Science).

When facing a huge input data set, there are typically two possible ways to proceed. One way could be to redesign known algorithms so that, while maintaining almost the same performance, can be run efficiently with much larger input data sets. The second possible approach is reducing the size of the data in such a way that the reduction does not change the result drastically. This is valid because, in many data mining applications, approximately optimal answers are acceptable. Data sampling is one such a data reduction method. Thus, we could take a random sample of the instance space and do data mining on it. Several researchers (see, for instance, Wang et al., 1998) have objected to this approach due to the difficulty of determining an appropriate sample size. In this paper, we advocate for this second approach of reducing the dimensionality of the data through random sampling. For this, we propose a general problem that covers many situations arising in data mining algorithms, and a general sampling algorithm for solving it.

A typical task of knowledge discovery and data mining is to find out some "rule" or "law" explaining a huge set of examples well. It is often the case that the size of possible candidates for such rules is still manageable. Then the task is simply to select a rule among all candidates that has certain "utility" on the dataset. This is the problem we discuss in this paper, and we call it *General Rule Selection*. More specifically, we are given an input data set X of examples, a set H of *rules*, and a *utility function* U that measures "usefulness" of each rule on X. The problem is to find a nearly the best rule h, more precisely, a rule h satisfying $U(h) \geq (1-\epsilon)U(h_\star)$, where $h_\star$ is the best rule and ϵ is a given accuracy parameter. Though simple, this problem covers several key problems occurring in data mining.

We would like to solve the General Rule Selection problem by random sampling. From a statistical point of view, this problem can be solved by taking *first* a random sample S from the domain X and *then* selecting $h \in H$ with the largest $U(h)$ on S. If we choose enough number of examples from X randomly, then we can guarantee that the selected h is nearly best within a certain confidence level. We will refer to this simple method as a *batch sampling* method.

As mentioned, one of the most important issues when doing random sampling is the choice of a proper sample size, i.e., the number of examples. To do this, any sampling method must take into account problem parameters, an accuracy parameter, and a confidence parameter.

Widely used and theoretically sound tools to determine appropriate sample size for given accuracy and confidence parameters are the so called concentration bounds or large deviation bounds like the Chernoff or the Hoeffding bounds. They are commonly used in most of the theoretical learning research (see Kearns and Vazirani (1994) for some examples) as well as in many other branches of computer science. For examples of sample size calculated with concentration bounds for data mining problems, see, e.g., Kivinen and Mannila (1994) and Toivonen (1996). While these bounds usually allow us to calculate sample size needed in many situations, it is usually the case that resulting sample size is immense to obtain a reasonably good accuracy and confidence. Moreover, in most of the situations, to apply these bounds, we need to assume the knowledge of certain problem parameters that are unknown in practical applications.

It is important to notice that, in the batch sampling method, the sample size is calculated a priori and thus, it must be big enough so that it will work well in all the situations we might encounter. In other words, the sample size provided by the above bounds for the batch

sampling should be the worst case sample size and thus, it is overestimated for most of the situations. This is one of the main reasons why researchers have found that, in practice, these bounds are overestimating necessary sample size for many non worst-case situations; see, e.g., discussion by Toivonen (1996) for sampling for association rule discovery.

For overcoming this problem, we propose in this paper to do sampling in an on-line sequential fashion instead of batch. That is, an algorithm obtains examples sequentially one by one (or block by block), and it determines from those obtained examples whether it has already received enough examples for issuing the currently best rule as nearly the best with high confidence. Thus, we do not fix sample size a priori. Instead, sample size depends *adaptively* in the situation at hand. Due to this adaptiveness, if we are not in a worst case situation as fortunately happens in most of the practical cases, we may be able to detect this and use significantly fewer examples than in the worst case. Following this approach, we propose a general algorithm—AdaSelect—for solving the General Rule Selection problem, which provides us with an efficient tool for many knowledge discovery applications. This algorithm evolves from our preliminary work on adaptive sampling in Domingo et al. (1998). Furthermore, to show the applicability of our approach, we show an instantiation of the approach to a particular problem, boosting decision stumps, and present some experiments to verify the correctness and advantage of our approach. These experiments were presented in a preliminary version in Domingo and Watanabe (2000a).

The idea of adaptive sampling is quite natural, and various methods for implementing this idea have been proposed in the literature. In statistics, in particular, these methods have been studied in depth under the name of "sequential test" or "sequential analysis" (see, for instance, the pioneer book of Wald, 1947). Their main goal has been, however, to test statistical hypotheses. Thus, even though some of their methods are applicable for some instances of the General Rule Selection problem, as far as the authors know, there has been no method that is as reliable and efficient as AdaSelect for the General Rule Selection problem. More recent work on adaptive sampling comes from the database community (Lipton and Naughton, 1995; Lipton et al., 1993). The problem they address is that of estimating the size of a database query by using adaptive sampling. While their problem (and algorithms) is very similar in spirit to ours, they do not deal with selecting competing hypotheses that may be arbitrarily close in value, and this makes their algorithms simpler but not applicable to our problem. From the data mining community, the work of John and Langley (1996) and Provost et al. (1999) is related to ours although their technique for stopping sampling is based on fitting the learning curve of some assumed learning algorithm, while our stopping condition of sampling is from purely statistical bounds. The other crucial difference is that their methods are used on top of the assumed learning algorithm, whereas our method is for a much simpler problem and it is used to speed-up some important part, not a whole, of a data mining algorithm. These differences will be explained in more detail later.

The paper is organized as follows. In the next section, we explain still at some intuitive level the idea of our method and its advantage over other random sampling methods. In Section 3, we present our problem and algorithm, and prove theorems concerning its reliability and complexity. Some improvements are also discussed there. In Section 4, by using one particular application for our example, we explain how our general algorithm is instantiated and show its advantage by experimental results. Other possible

applications are briefly discussed in Section 5.We conclude in Section 6 highlighting future work.

2. Why our sampling method

In this section we explain still at some intuitive level, the advantage of our method over other random sampling methods.

To illustrate the basic idea of our adaptive sampling method and discuss its difference from the other methods, let us consider one very simple problem. The problem is to test whether the probability $p_\star$ that a given condition C holds in a database X is more than $1/2$ or not. If C held exactly in $1/2$ of the transactions in X, then we would have to check all the database to notice it; thus, let us suppose that $p_\star = 1/2 \pm \gamma_\star$ for some $\gamma_\star$, $0 < \gamma_\star \leq 1/2$. Intuitively, if $\gamma_\star$ is large, that is, C holds on either many or very few transactions of the database, then just sampling a small number of transactions should be enough to figure out that the answer is positive or negative. On the other hand, if $\gamma_\star$ is small, then testing $p_\star > 1/2$ becomes difficult and we need to see a large number of transactions. Thus, it is natural that sample size, i.e., the number of examples, grows depending on $1/\gamma_\star$, and in fact, it can be shown that $\mathcal{O}(1/\gamma_\star^2)$ sample size is necessary. Our adaptive sampling method provides an algorithm that achieves this test with roughly $\mathcal{O}(1/\gamma_\star^2)$ sample size, an almost optimal sample size.

For comparison, let us use the batch sampling method for this example problem. That is, we randomly choose some number of transactions, compute the ratio that C holds on them, and determine whether $p_\star > 1/2$ or not. The sample size can be calculated by using an appropriate large deviation bound. For example, we can use the Hoeffding bound, which has been widely used in computer science (see, e.g., Kearns and Vazirani (1994)). For any γ, the Hoeffding bound tells us that $\mathcal{O}(1/\gamma^2)$ examples are sufficient to detect, with high confidence, whether C holds more than $1/2 + \gamma$ of X or less than $1/2 - \gamma$. Thus, if we know $\gamma_\star$ in advance, we can solve our problem with the optimal sample size. In most cases, however, $\gamma_\star$ is not known in advance, and we have to choose appropriate γ. Since the high confidence is guaranteed only if $\gamma_\star \geq \gamma$, we have to use γ that is smaller than $\gamma_\star$. But if we underestimate $\gamma_\star$, then sample size $\mathcal{O}(1/\gamma^2)$ becomes unnecessarily large. In other words, the sample size required by the batch sampling method has to be always big enough to cover all possible "expected" values of $\gamma_\star$, or one has to choose γ carefully, which makes sampling difficult to use in practice. For our adaptive sampling method, on the other hand, we do not need to estimate $\gamma_\star$; the algorithm automatically detects it and uses appropriate sample size accordingly. This is the advantage of our method over the simple batch sampling.

The notion of "adaptive sampling" has been already discussed in the database (Lipton and Naughton, 1995; Lipton et al., 1993). Roughly speaking, the following problem is discussed. Given a database and a query over the databases (for instance, a selection or a joint), we want to estimate the query size, that is, the number of transactions associated with the query in the database, up to a certain error and confidence level. They designed algorithms for this task. Below we refer to their algorithms as *Adaptive Estimator*. More specifically, for a given database X, a condition C, an accuracy parameter ϵ, and a confidence parameter δ, Adaptive Estimator estimates (through random sampling) the probability $p_\star$ that transactions in X

satisfy C up to a multiplicative error ϵ with probability $> 1 - \delta$. That is, with probability $> 1 - \delta$, the algorithm yields estimate p of $p_\star$ such that $(1 - \epsilon)p_\star \leq p \leq (1 + \epsilon)p_\star$. For achieving this task, Adaptive Estimator collects examples from the database sequentially at random while checking whether collected examples are sufficient for terminating the execution with the current estimate. The number of examples used by the algorithm depends on $p_\star$, ϵ, and δ, which is roughly $O(1/\epsilon^2 p_\star)$ for some fixed δ. Here again we do not need any a priori estimate of $p_\star$. The algorithm can determine appropriate sample size. Thus, this algorithm is very similar in spirit to ours. Notice, however, Adaptive Estimator (and any algorithm in their method) is designed for estimating $p_\star$, but for our problem, it is necessary to estimate $p_\star - 1/2$. Moreover, the difficulty of our problem depends on $\gamma_\star = |p_\star - 1/2|$, i.e., the closeness of $p_\star$ to $1/2$, and not on $p_\star$ itself. Hence, for solving our problem with Adaptive Estimator, we need to set $\epsilon \approx \gamma_\star$; here again we have to estimate $\gamma_\star$! In a word, our method is applicable to a wider class of estimation problems, and this example is a typical one of those problems. (It should be remarked here that Adaptive Estimator works better than our algorithms for estimating $p_\star$; that is, their algorithms are better for this particular problem. See discussion in Section 3.)

3. The adaptive sampling algorithm

In this section we formally describe the problem we would like to solve. We then present our algorithm, prove its reliability and estimate its efficiency and give some modifications of our algorithm that make it efficient asymptotically and/or practically under certain situations.

3.1. General rule selection problem

We begin introducing some notation. Let $X = \{x_1, x_2, \ldots, x_k\}$ be a (large) set of examples and let $H = \{h_1, \ldots, h_n\}$ be a (finite, not too large) set of n functions such that $h_i : X \mapsto \mathbb{R}$. That is, $h \in H$ can be thought as a function that can be evaluated on an example x producing a real value $y_{h,x}$ as a result. Intuitively, each $h \in H$ corresponds to a "rule" or "law" explaining examples, which we call below a *rule*, and $y_{h,x}$ measures the "goodness" of the rule on x. For example, if the task is to predict a particular Boolean feature of example x in terms of its other features, then we could set $y_{h,x} = 1$ if the feature is predicted correctly by h, and $y_{h,x} = 0$ if it is predicted incorrectly. We also assume that there is some fixed real-valued and nonnegative *utility function* $U(h)$, measuring some global "goodness" of the rule (corresponding to) h on the set X. More specifically, $U(h)$ is defined by

$$U(h) = F(\text{the average value of } y_{h,x} \text{ in } X),$$

where F is some function $\mathbb{R} \mapsto \mathbb{R}$. That is, the "goodness" of h measured by the utility function U is not just the average of $y_{h,x}$, but it could be something else that is computed by F from the average of $y_{h,x}$. For the "average" of $y_{h,x}$, we simply use here the arithmetic average $\sum_{x \in X} y_{h,x} / \|X\|$, which we denote $\text{avg}(y_{h,x} : x \in X)$.

Now we are ready to state our problem.

General rule selection
Given: X, H, and ϵ, $0 < \epsilon < 1$.
Goal: Find $h \in H$ such that $U(h) \geq (1 - \epsilon) \cdot U(h_\star)$,
where $h_\star \in H$ be the rule with maximum value of $U(h_\star)$.

For any $S \subseteq X$, we define $U(h, S) = F(\text{avg}(y_{h,x} : x \in S))$. Then $U(h)$ is simply $U(h, X)$. Thus, one trivial way to solve our problem is evaluating all functions h in H over all examples x in X, hence computing $U(h, X)$ for all h, and then finding the h that maximizes this value. Obviously, if X is large, this method might be extremely inefficient. We want to solve this task much more efficiently by random sampling. That is, we want to look only at a fairly small, randomly drawn subset $S \subseteq X$, find the h that maximizes $U(h, S)$, and still be sure with *high* probability that h is *close enough* to the best one. (This is similar to the PAC learning, whose goal is to obtain a *probably approximately* correct hypothesis.)

Remark 1 (Accuracy parameter ϵ). Intuitively, our task is to find some $h \in H$ whose utility is reasonably high compared with the maximum $U(h_\star)$, where the *accuracy* of $U(h)$ to $U(h_\star)$ is specified by the parameter ϵ. Certainly, the closer $U(h)$ is to $U(h_\star)$ the better. However, depending on the choice of U, the accuracy is not essential in some cases, and we may be able to use a large ϵ. The advantage of our algorithm becomes clear in such cases. (See discussion at the end of the next subsection and the application in the next section.)

Remark 2 (Confidence parameter δ). We want to achieve the goal above by "random sampling", i.e., by using examples randomly selected from X. Then there must be some chance of selecting bad examples that make our algorithm to yield unsatisfactory $h \in H$. Thus, we introduce one more parameter $\delta > 0$ for specifying *confidence* and require that the probability of such error is bounded by δ.

Remark 3 (Distribution on X). In order to simplify our discussion, we assume the uniform distribution over X, in which case $U(h)$ is just the same as $U(h, X)$. But our method works as well on any distribution D over X so long as we can get each example independently following the distribution D. In fact, this is the case for the example we will consider in Section 4. There is no difference at all in such cases except that $U(h)$ is now defined as $U(h) = F(E[y_{h,x}])$, where $E[y_{h,x}]$ is the expectation of $y_{h,x}$ under the distribution that we assume over X.

Remark 4 (Condition on H). In order to simplify our discussion, we assume in the following that the value of $y_{h,x}$ is in $[0, d]$ for some constant $d > 0$. (From now on, d will denote this constant.)

Remark 5 (Condition on U). Our goal does not make sense if $U(h_\star)$ is negative. Thus, we assume that $U(h_\star)$ is positive. Also in order for (any sort of) random sampling to work, it cannot happen that a single example changes drastically the value of U; otherwise, we would be forced to look at all examples of X to even approximate the value of $U(h)$. Thus, we require that the function F that defines U is smooth. Formally, F need to be c-Lipschitz for some constant $c \geq 0$, as defined below. (From now on, c will denote the Lipschitz constant of F.)

Definition 1. Function $F : \mathbb{R} \mapsto \mathbb{R}$ is c-Lipschitz if for all x, y it holds $|F(x) - F(y)| \leq c \cdot |x - y|$. The Lipschitz constant of F is the minimum $c \geq 0$ such that F is c-Lipschitz (if there is any).

Observe that all Lipschitz functions are continuous, and that all differentiable functions with a bounded derivative are Lipschitz. In fact, if F is differentiable, then by Mean Value Theorem, the Lipschitz constant of F is $\max_x |F'(x)|$. Also note that from the above conditions, we have $0 \leq U(h) \leq cd$ for any $h \in H$.

3.2. *Adaptive selection algorithm*

One can easily think of the following simple batch sampling approach. Obtain a random sample S from X of a priori fixed size m and output the function from H that has the highest utility in S. There are several statistical bounds to calculate an appropriate number m of examples. While this batch sampling solves the problem, its efficiency is not satisfactory because, as discussed in the previous section, it has to choose sample size for the worst case. For overcoming this inefficiency, we take a sequential sampling approach. Instead of statically deciding the sample size, our algorithm obtains examples sequentially one by one and it stops according to some condition based on the number of examples seen and the values of the functions on the examples seen so far. That is, the algorithm *adapts* to the situation at hand, and thus if we are not in the worst case, it would be able to realize of that and stop earlier. Figure 1 shows a pseudo-code of the algorithm we propose, called AdaSelect, for solving the General Rule Selection problem.

Statistical bounds used to determine sample size for the batch sampling still plays a key role for designing our algorithm. Here we choose the Hoeffding bound. One can use any reasonable bound, but the reason that we choose the Hoeffding bound is that basically no assumption is necessary[1] for using this bound to estimate the error probability and calculate the sample size. On the other hand, for example, the bound from the Central Limit Theorem might be appropriate for some practical situations since it behaves better. We will discuss this issue in the next subsection.

Now we provide two theorems discussing the reliability and the complexity of the algorithm AdaSelect. For our analysis, we derive the following bounds from the Hoeffding bound.

Algorithm AdaSelect(X, H, ϵ, δ)
  $t \leftarrow 0$; $S_t \leftarrow \emptyset$; $n \leftarrow ||H||$
  repeat
    $x \leftarrow$ randomly drawn example from X;
    $t \leftarrow t + 1$;
    $\alpha_t = cd\sqrt{\ln(nt(t+1)/\delta)/(2t)}$;
    $S_t := S_{t-1} \cup \{x\}$;
  until $\exists h \in H\ [U(h, S_t) \geq \alpha_t \cdot (2/\epsilon - 1)]$;
  output $h \in H$ with the largest $U(h, S_t)$;

Figure 1. Pseudo-code of our on-line sampling algorithm AdaSelect.

Lemma 2. *Let $S \subseteq X$ be a set of size t obtained by independently drawing t elements from X at random. For any $h \in H$ and $\alpha \geq 0$, we have*

$$\Pr\{U(h, S) \geq U(h) + \alpha\} < \exp\left(-2\frac{\alpha^2}{(cd)^2}\, t\right).$$

The same bound holds for $\Pr\{U(h, S) \leq U(h) - \alpha\}$.

Proof: We prove the first inequality. The second one is proved symmetrically. Let g be the random variable $\mathrm{avg}(y_{h,x} : x \in S)$, i.e., the arithmetic average of t independent random variables. Observe that $U(h) = F(E[g])$, and $U(h, S) = F(g)$, where $E[g]$ is g's expectation. Then, using the fact that F is c-Lipschitz, we have $\Pr\{U(h, S) \geq U(h) + \alpha\}$ $\leq \Pr\{g \geq E(g) + \alpha/c\}$. Note that g is the average of t independent random variables with range bounded by d. Hence, by the Hoeffding bound, the above probability is less than $\exp(-2\frac{\alpha^2}{(cd)^2}\, t)$, as claimed. □

We first prove the reliability of Algorithm AdaSelect.

Theorem 3. *With probability* $1-\delta$, $\mathrm{AdaSelect}(X, H, \epsilon, \delta)$ *outputs a function* $h \in H$ *such that* $U(h) \geq (1 - \epsilon)U(h_\star)$.

Proof: For a fixed $\epsilon > 0$, we will show that, with probability less than δ, the function output by AdaSelect (X, H, ϵ, δ) is in H_{bad}, defined as $H_{\mathrm{bad}} = \{h \in H \mid U(h) < (1 - \epsilon)U(h_\star)\}$. That is, we want to bound the following error probability P_{error} by δ.

$$P_{\mathrm{error}} = \Pr\{\text{AdaSelect yields some } h \in H_{\mathrm{bad}}\}.$$

Here we regard one repeat-loop iteration as a basic step of the algorithm and measure the algorithm's running time in terms of the number of repeat-loop iterations. Note that the variable t keeps the number of executed repeat-loop iterations. Let t_0 be the integer such that the following inequalities hold. (We assume that $\alpha_0 = \infty$).

$$\alpha_{t_0-1} > U(h_\star)\frac{\epsilon}{2} \quad \text{and} \quad \alpha_{t_0} \leq U(h_\star)\frac{\epsilon}{2}.$$

Note that α_t is strictly decreasing as a function of t; hence, t_0 is uniquely determined. As we see below, the algorithm terminates by the t_0th step (i.e., within t_0th repeat-loop iterations) with high probability.

For deriving the bound, we consider the following two cases: (Case 1) Some $h \in H_{\mathrm{bad}}$ satisfies the stopping condition of the repeat-loop before the t_0th step, and (Case 2) $h_\star$ does not satisfy the stopping condition during the first t_0 steps. Clearly, whenever the algorithm makes an error, one of the above cases certainly occurs. Thus, by bounding the probability that either (Case 1) or (Case 2) occurs, we can bound the error probability P_{error} of the algorithm.

First we bound that the probability of (Case 1). Let h_{bad} be a rule in H_{bad} with the largest utility, and let $EV(h, \tilde{t})$ be the event that h satisfies the stopping condition of AdaSelect at the $\tilde{t}$th step. Then we have

$$\begin{aligned}&\Pr\{\text{(Case 1) holds}\}\\ &\quad\le \sum_{1\le\tilde{t}<t_0}\sum_{h\in H_{\mathrm{bad}}} \Pr\{EV(h,\tilde{t})\} \le \sum_{1\le\tilde{t}<t_0} n\cdot\Pr\{EV(h_{\mathrm{bad}},\tilde{t})\}.\end{aligned}$$

Note that $\Pr\{EV(h_{\mathrm{bad}}, \tilde{t})\}$ is bounded by $P_1(\tilde{t}) = \Pr\{U(h_{\mathrm{bad}}, S_{\tilde{t}}) \ge \alpha_{\tilde{t}} \cdot (2/\epsilon - 1)\}$. Now we would like to bound this $P_1(\tilde{t})$ by using Lemma 2. Note that the Hoeffding bound and hence Lemma 2 is applicable only for the case that the size of set $S \subseteq X$ is fixed in advance. On the other hand, in our algorithm t itself is a random variable. Thus, precisely speaking, we cannot use Lemma 2 directly, and we argue as follows. First fix any $\tilde{t}$, $1 \le \tilde{t} < t_0$. We modify our algorithm by replacing the stopping condition of the repeat-loop with "$t \ge \tilde{t}$?"; that is, modify the algorithm so that it always sees $\tilde{t}$ examples. Then it is easy to show that if $U(h_{\mathrm{bad}}, S_{\tilde{t}}) \ge \alpha_{\tilde{t}} \cdot (2/\epsilon - 1)$ in the original algorithm, then with the same sampling sequence we have $U(h_{\mathrm{bad}}, S_{\tilde{t}}) \ge \alpha_{\tilde{t}} \cdot (2/\epsilon - 1)$ in the modified algorithm. Thus, $P_1(\tilde{t})$ is at most $P_1'(\tilde{t})$ $= \Pr\{U(h_{\mathrm{bad}}, S_{\tilde{t}}) \ge \alpha_{\tilde{t}} \cdot (2/\epsilon - 1)$ in the modified algorithm$\}$. Note that $\alpha_{\tilde{t}} > U(h_\star)(\epsilon/2)$ for any $\tilde{t} < t_0$ and that $U(h_\star)(1-\epsilon) > U(h_{\mathrm{bad}})$. From these we have

$$P_1'(\tilde{t}) \le \Pr\{U(h_{\mathrm{bad}}, S_{\tilde{t}}) > U(h_{\mathrm{bad}}) + \alpha_{\tilde{t}}\}.$$

On the other hand, since the sample size is fixed in the modified algorithm, we can now use Lemma 2 bound to the righthand side. Also by our choice of α_t, we have

$$P_1'(\tilde{t}) < \exp\left(-2\frac{\alpha_{\tilde{t}}^2}{(cd)^2}\tilde{t}\right) \le \frac{\delta}{n\tilde{t}(\tilde{t}+1)}$$

In fact, α_t is defined so that the last inequality holds, which point will be important in Section 3.3.

Now by using the above bound, we can bound $\Pr\{$(Case 1) holds$\}$ by $\sum_{1\le\tilde{t}<t_0} \frac{\delta}{\tilde{t}(\tilde{t}+1)} \le \delta(1 - 1/t_0)$.

Next consider (Case 2). Clearly, (Case 2) implies $U(h_\star, S_{\tau_0}) < \alpha_{\tau_0} \cdot (2/\epsilon - 1)$, which implies, as in (Case 1), that $U(h_\star, S_{t_0}) < \alpha_{t_0} \cdot (2/\epsilon - 1)$ in a modified algorithm that always sees t_0 examples. ¿From this observation, an argument as in the (Case 1) shows that the probability of (Case 2) is at most $\exp(-2\frac{\alpha_{t_0}^2}{(cd)^2}t_0) = \frac{\delta}{nt_0(t_0+1)} \le \delta/t_0$.

Therefore, the probability that either (Case 1) or (Case 2) holds is bounded by δ. □

Next we estimate the running time of the algorithm. As above we regard one repeat-loop iteration as a basic step of the algorithm and measure the algorithm's running time in terms of the number of repeat-loop iterations that is exactly the number of required examples. In the above proof, we have already showed that the probability that the algorithm does not terminate within t_0 steps, that is, (Case 2) occurs, is at most δ. Thus, the following theorem is immediate from the above proof.

Theorem 4. *With probability* $1-\delta$, AdaSelect(X, H, ϵ, δ) *halts within* t_0 *steps* (*in other words*, AdaSelect(X, H, ϵ, δ) *needs at most* t_0 *examples*), *where* t_0 *is the smallest integer such that* $\alpha_{t_0} \leq U(h_\star)(\epsilon/2)$.

From the fact that t_0 is the smallest integer satisfying $\alpha_t \leq U(h_\star)(\epsilon/2)$, we may assume that $\alpha_{t_0} \approx U(h_\star)(\epsilon/2)$. Then by using some approximate unfolding of the logarithm factor, we can estimate t_0 as follows.

$$t_0 \approx 2\left(\frac{cd}{\epsilon U(h_\star)}\right)^2 \cdot \left(\ln\frac{4n}{\delta} + 4\ln\frac{cd}{\epsilon U(h_\star)}\right).$$

Let us discuss the meaning of this formula. Since both n and δ are within the log function, their influence to the complexity is small. In other words, we can handle the case with a relatively large number of rules and/or the case requiring a very high confidence without increasing too much the sample size needed. Or we may roughly consider that the sample size is proportional to $(1/\epsilon)^2$ and $((cd)/U(h_\star))^2$. Depending on the choice of U, in some cases we may assume that $(cd)/U(h_\star)$ is not so large; or, in some other cases, we may not need small ϵ, and thus, $1/\epsilon$ is not so large. AdaSelect performs particularly well in the latter case.

It should be noted here that in the case that $y_{h,x}$ is either 0 or 1 and $U(h, S)$ is defined as $U(h, S) = \text{avg}(y_{h,x} : x \in S)$, we had better use the algorithm of Lipton et al. (1993) Of course, we could also use AdaSelect even for this case, but the sample size is roughly $\mathcal{O}(1/(\epsilon U(h_\star))^2)$, whereas their sample size is $\mathcal{O}(1/\epsilon^2 U(h_\star))$.

We should also mention that a similar result could be obtained for the case where $\|H\|$ is infinite using the VC-dimension of H instead of its cardinality. However, in this case, finding the best rule at each iteration cannot be done by enumeration. Some other strategy, specific for each class, is required.

In summary, AdaSelect shows its advantage most when U is chosen so that (1) relatively large ϵ is sufficient, and (2) though $(cd)/U(h_\star)$ is not bounded in general, it is not so large in lucky cases, which happen more often than the bad cases. We will see such an example in Section 4.

3.3. Some improvements

We can still improve the above algorithm while keeping its outline. Here we explain two such improvements.

Although we may roughly consider that the sample size is proportional to $((cd)/(\epsilon U(h_\star)))^2$, it is certainly nice if we can reduce the factor $\ln(4n/\delta) + 4\ln((cd)/(\epsilon U(h_\star)))$. First we try reducing this factor.

Recall the probability analysis in the proof of Theorem 3. The probability of (Case 1) is bounded by the sum of $nP_1(\tilde{t})$ for all $\tilde{t}$, $1 \leq \tilde{t} < t_0$. This is because our stopping condition is not monotonic and we have to bound the error probability for each $\tilde{t}$, $1 \leq \tilde{t} < t_0$. Then this error probability bound could be reduced if we check the stopping condition less often instead of checking it every time we get an example. This is an idea of our first improvement.

This modification is also practically good in some cases where it is not efficient to draw examples one by one or it is not so easy to compute U in an incremental way.

Now let us state our modification precisely and discuss how it works. For a given parameter s, we modify the algorithm so that the stopping condition is checked only at the s^kth step for each $k \geq 0$. In other words, the repeat-condition is replaced with "$\exists k \geq 0\ [t = s^k]$ and $\exists h \in H\ [U(h, S_t) \geq \alpha_t \cdot (2/\epsilon - 1)]$?". (Of course, we do not have to draw one by one nor to compute U or α_t if $t = s^k$ does not hold for any k.) This reduces the number of occasions that some undesired hypothesis is selected during the first t steps from t to $\log_s t$. Due to this reduction, we can use the following α_t for $t = s^k$.

$$\alpha_t = cd\sqrt{\ln(nk(k+1)/\delta)/(2t)}. \tag{1}$$

Let us refer to this modified algorithm as the *geometric* AdaSelect, whereas the original one as the *arithmetic* AdaSelect. For this algorithm, we can also prove the same theorem as Theorem 3. The proof is essentially the same; only the difference is now, instead of t_0, we need to consider t_1 that is defined as $t_1 = s^{k_1}$ with an integer k_1 satisfying

$$\alpha_{s^{k_1-1}} > U(h_\star)\frac{\epsilon}{2} \quad \text{and} \quad \alpha_{s^{k_1}} \leq U(h_\star)\frac{\epsilon}{2}.$$

Similarly, we can show that the algorithm halts within t_1 steps with high probability. Then an upper bound of the sample size of the geometric AdaSelect is estimated as follows.

$$t_1 \approx 2s\left(\frac{cd}{\epsilon U(h_\star)}\right)^2\left(\ln\frac{4n}{\delta} + 2\ln\log_s\frac{cd}{\epsilon U(h_\star)} + 2\ln\log_s 2s\right).$$

Here we have a factor $2s$ instead of 2 as before. The new factor s is what we have to pay for by modifying the algorithm to check the stopping condition less often.

Comparing this estimation with the previous one for t_0, we can see that t_1 is asymptotically better than t_0 for any fixed s. In actual applications, we can often improve the sample size by choosing s appropriately. See the next section for such example.

Next we consider using a bound from the Central Limit Theorem because it is again widely applicable and it gives a quite good approximation.

We first recall the Central Limit Theorem. Let g be the average of t independent and identically distributed random variables $g_1, \ldots, g_t$ that take a value in some closed interval, and let E and V be the expectation and variance of each g_i respectively. (Then we can calculate the expectation and the variance of g as $\mathrm{E}[g] = E$ and $\mathrm{V}[g] = V/t$.) *The Central Limit Theorem* claims that $(g - E[g])/\sqrt{V[g]}$ converges to the normal distribution when t goes infinity. More specifically, if t is large enough, then we may be able to use the following approximation.

$$\Pr\left\{\frac{g - \mathrm{E}[g]}{\sqrt{\mathrm{V}[g]}} \leq a\right\} = \Pr\left\{\frac{g - E}{\sqrt{\mathrm{V}/t}} \leq a\right\} \approx \Phi(a),$$

where $\Phi(a) = (1/\sqrt{2\pi})\int_{-\infty}^{a} e^{-x^2/2}\,\mathrm{d}x$ is the normal distribution function. We refer to this approximation in the following as *the Central Limit Approximation*.

While Φ and Φ^{-1} are not so difficult to compute, it is still convenient if we can state our bound without using Φ, and it is particularly helpful for comparing with the Hoeffding bound. Fortunately, we have reasonably close bound $1 - \Phi(x) \leq \exp(-x^2/2)/(x\sqrt{2\pi})$ (Feller, 1950). Using this, we can easily derive the following bound.

$$\Pr\{g \geq E + \epsilon\} \approx \Pr\{g \leq E - \epsilon\} \lesssim \frac{\sqrt{V}}{\epsilon\sqrt{2\pi t}} \exp\left(-\frac{\epsilon^2}{2V}\, t\right).$$

By using this bound, we can show a lemma corresponding to Lemma 3.2.

Lemma 5. *Let $S \subseteq X$ be a set of size t obtained by independently drawing t elements from X at random. Let b^2 be an upper bound of the variance of $y_{h,x}$. For any $h \in H$ and $\alpha \geq 0$, if t is large enough to use the Central Limit Approximation, then we have*

$$\Pr\{U(h, S) \geq U(h) + \alpha\} \lesssim \frac{bc}{\alpha\sqrt{2\pi t}} \exp\left(-\frac{\alpha^2}{2(bc)^2}\, t\right).$$

The same inequality holds for $\Pr\{U(h, S) \leq U(h) - \alpha\}$.

Now we modify our algorithm based on this lemma. First consider the original, i.e., the arithmetic Adaselect. Recall the remark on α_t in the proof of Theorem 3. We defined α_t so that the probability given by the Hoeffding bound is bounded by $\delta/(nt(t+1))$. Thus, with our new bound, we need to define α_t so that the following inequality holds.

$$\frac{bc}{\alpha_t\sqrt{2\pi t}} \exp\left(-\frac{\alpha_t^2}{2(bc)^2}\, t\right) \leq \frac{\delta}{nt(t+1)},$$

for which it is sufficient to define α_t as follows.

$$\alpha_t = bc\sqrt{(2\ln\tau - \ln\ln\tau + 1)/t}, \quad \text{where } \tau = nt(t+1)/(2\delta\sqrt{\pi}). \tag{2}$$

Then it is clear from our discussion that a theorem corresponding to Theorem 3 holds for this choice of α_t. Similarly, we can define α_t for the geometric AdaSelect; we only need to change t in τ to k. Also we can analyze the efficiency of these modified algorithms in the same way as Theorem 4, but we leave the analysis to the interested reader.

4. An application of AdaSelect: A case study

In this section we describe in detail how our method can be instantiated into a particular application and provide an experimental evaluation of the algorithms proposed in Section 3.

4.1. Some remarks on applications and our example

Before describing our application, let us make some remarks about the kind of problems where our algorithms can be applied. First, recall that our algorithms are designed for the General Rule Selection problem described in Section 3. This problem is simple; it may be

too simple to capture any sort of actual data mining problem. We do not propose to use our sampling algorithms for solving some data mining problems, but instead, we do propose to use them as a tool. As explained below, some subtasks of data mining algorithms can be casted as instances of the General Rule Selection problem and then, those parts can be scaled by using our sampling algorithms instead of all the data. Also as remarked in Section 2, our method is not the one that we should always use. It shows its advantages most when an appropriate utility function U can be used, and there are some cases, though only very particular cases, where the other method performs better.

The example we choose is a boosting based classification algorithm that uses a simple decision stump learner as a base learner, and our sampling algorithms are used to scale-up the base learner. *Boosting* (Freund and Schapire, 1997) is a technique to make a sufficiently "strong" learning algorithm based on a "weak" learning algorithm. Since a weak base learning algorithm (a *base learner*, in short) does not need to obtain a highly accurate classification rule, we can use for a base learner a learning algorithm that produces a not-so-accurate but simple classification rule (which we call a *weak hypothesis*). Then since a set H of such simple weak hypotheses is not so large, we may be able to select the best one just by searching through H exhaustively, which is indeed one instance of our General Rule Selection problem. That is, any rule selection algorithm can be used for a base learner, and we do this by random sampling. Here we use a set H_{DS} of decision stumps for our weak hypothesis class, where a *decision stump* is a single-split decision tree based on a single attribute with only two terminal nodes. We consider the case that attributes used for a split node are discrete attributes and that the number of all possible decision stumps is not so large. On the other hand, a set X of examples is huge and it is just infeasible to see all examples in X. In such a situation, the best possible approach is the selection via random sampling, i.e., selecting a weak hypothesis from H_{DS} that performs the best classification on randomly selected examples from X. We use our sampling algorithms for this selection.

Here we had better explain why this particular example is chosen. Weak hypothesis selection was the initial target of our series of research (Domingo et al., 1998) and we have chosen this example because it is the easiest but quite illustrative application of AdaSelect that reveals why the other similar random sampling methods are not suitable. (See the discussion about the utility function in the next subsection.) It also shows a fundamental difference between our method and other sampling methods described in the knowledge discovery literature like John and Langley (1996) and Provost et al. (1999). These methods are based on learning curves of the induction algorithm used as a black box. Roughly speaking, these methods run the overall learning algorithm (which itself corresponds to executing enough number of boosting iterations) several times by increasing sample size, produce "strong" hypotheses, and determine the point where the accuracy of obtained hypotheses saturates. That is, these methods should be used on top of some well-designed learning algorithm, while ours are used as a part of some learning algorithm. Finally, the choice of boosting has been motivated for its current success. Many recent papers (see, for instance, Bauer and Kohavi (1998) and references therein) report very extensive experiments showing the advantage of boosting against other induction methods, but to the authors' knowledge, there has not been much research reported on how to use it on very large datasets.

We should also clarify the type of boosting technique we use here. For the boosting part, the obvious choice would be the AdaBoost algorithm of Freund and Schapire (1997) since this algorithm has been repeatedly reported to outperform any other voting method. However, AdaBoost has the following problem that makes it not suitable for being combined with our sampling method. AdaBoost is designed for "boosting by subsampling" or "boosting by re-weighting" where its base learner is required to produce a hypothesis that tries to minimize error with respect to a weighted training set. The training set is fixed at the beginning and used throughout all the boosting steps with AdaBoost modifying the weights at each iteration depending on the hypotheses being obtained. However, we are aiming to use our sampling method to speed up the base learner by changing the size of a training set adaptively. Thus, instead of fixing a training set, we have to draw random examples, filter them according to their current weights, and pass some of them to the base learner. This is what is usually called "boosting by filtering" or "boosting by re-sampling". However, the weights produced by AdaBoost can become very small and thus the expected time required by the filter becomes very large. This problem of AdaBoost has been recently addressed by Domingo and Watanabe (1999). For overcoming this problem, they proposed MadaBoost, a simple modification of AdaBoost that is suitable for boosting by re-weighting and re-sampling while keeping the desired boosting property. MadaBoost uses basically the same weighting scheme as AdaBoost but using the initial weight as a saturation bound so the weights cannot grow uncontrolled. The combined function is a weighted majority of the base functions defined in the same way as AdaBoost. (For more details on MadaBoost we refer the reader to Domingo and Watanabe (1999).) In our experiments, we use MadaBoost and do boosting by re-weighting and re-sampling. Thus, we can assume that a base learner can efficiently draw examples under an appropriately modified distribution at each boosting stage.

4.2. The description of the problem and the algorithm

As we have already explained, we use our sampling algorithm for a base learner used by the boosting algorithm MadaBoost. Our task is to select, from a weak hypothesis class, a "good" hypothesis that performs nearly the best classification on randomly selected examples from X. Here note that the boosting algorithm modifies a distribution over X at each boosting step, and the goodness of hypotheses has to be measured based on the current distribution. But on the other hand, we can assume some example generator EX, provided by the boosting algorithm, that generates x from X according to the current distribution.

Now we define each item of the General Rule Selection problem specifically. We assume some set of discrete attributes. Each example is regarded as a vector of the values of each attribute on the instance plus its class. Thus, set X contains all such labeled examples. The class H of rules is H_{DS}, the set of all possible decision stumps over the set of discrete attributes, and a decision stump is used as a classification rule. For each $h \in H_{DS}$ and each $x = (a, b) \in X$ (a being the example and b its class), we define $y_{h,x} = 1$ if x is correctly classified by h (that is, $h(a) = b$) and $y_{h,x} = 0$ otherwise. That is, $y_{h,x}$ indicates "goodness" of h on x. Boosting techniques require a base learner to provide a weak hypothesis that performs better than random guess. Since they were originally designed for binary classification the

worst classifier is assumed to have accuracy equal to 50%. A weak hypothesis with error less than 50% would force the boosting process to stop. Thus, in order to make sure we get a weak hypothesis with prediction error less than $1/2$, we should measure the goodness of a hypothesis by its "advantage" over the random guess, i.e., its correct probability $-1/2$. Here it is quite easy to adjust our goodness measure. What we need is to define the utility function U by $U(h) = \mathrm{E}[y_{h,x}] - 1/2$; that is, we use $F(y) = y - 1/2$ to calculate our total goodness of h from the average goodness of h. Accordingly, $U(h, S)$ is defined by

$$U(h, S) = \mathrm{avg}(t_{h,x} : x \in S) - \frac{1}{2} = \frac{\sum_{x \in S} y_{h,x}}{\|S\|} - \frac{1}{2}.$$

This is the specification of the problem that we want to solve. One important point here is that we do not have to worry so much about the accuracy parameter ϵ, which is due to our choice of the utility function. For example, we can fix $\epsilon = 1/2$. Then our sampling algorithm may choose some h whose utility $U(h)$ is $U(h_\star)/2$, just the half of the best one. But we can still guarantee that the misclassification probability of h is smaller than $1/2$, in fact, it is $1/2 - U(h_\star)/2$, whereas that of the best one is $1/2 - U(h_\star)$. On the other hand, if we measured the goodness of each hypothesis by its correct classification probability, then we would have to choose ϵ small enough (depending on the best performance) to ensure that the selected hypothesis is better than the random guess. By introducing the notion of "utility", we could discuss general enough selection problems that allow us to attack problems like this example easily.

Next we define some other parameters and instantiate our algorithm. Note that the function F used here to define U from $\mathrm{avg}(y_{h,x} : x \in S)$ is just $F(z) = z - 1/2$; hence, it is 1-Lipschitz and the parameter c is set to 1. Also since $y_{h,x}$ is either 0 or 1, the parameter d is set to 1. We use the bound from the Central Limit Approximation because it gives better sample size and the number of examples is large enough to assume that the Central Limit Approximation holds. For using this bound, we need a bound b for $\mathrm{V}[y_{h,x}]$. Note that $\mathrm{V}[y_{h,x}] = p_h(1 - p_h)$, where p_h is the probability that h classifies $x \in X$ correctly under the assumed distribution over X. Hence, $\mathrm{V}[y_{h,x}] \leq 1/4$, and we use $b = 1/4$. With all necessary parameters being fixed, we can now state our algorithm as in figure 2 (the arithmetic version). For the geometric version, we set the parameter $s = 2$ because with this setting, the sample size t_1' is smaller than t_0' for any reasonable situation.

Algorithm Decision Stump Selector % We set $\epsilon = 0.5$ and $\delta = 0.1$.
  $t \leftarrow 0$; $S \leftarrow \emptyset$; $n \leftarrow ||H_{DS}||$
  repeat
    use **EX** to generate one example and add it to S;
    $t \leftarrow t + 1$;
    $\alpha_t = \sqrt{(2 \ln \tau - \ln \ln \tau + 1)/t}$, where $\tau = nt(t+1)/(2\delta\sqrt{\pi})$;
    $U(h, S) \leftarrow \|\{x \in S : h \text{ classifies } x \text{ correctly}\}\|/t - 1/2$;
  until $(\exists h \in H[\, U(h, S) \geq \alpha_t(2/\epsilon - 1)\,])$
  output $h_0 \in H$ with largest $U(h, S)$;
  output $U(h_0, S) + 1/2$ as an estimation of h_0's success prob.;

Figure 2. The arithmetic version of Decision Stump Selector using the CLA bound.

One additional remarks on this algorithm. We modified the algorithm to output not only the hypothesis h_0 but also an estimation of the correct classification probability of h_0, because it is necessary in the boosting algorithm.

4.3. Experimental results

Here we provide some experimental results for investigating the performance of our boosted Decision Stump Selector.

We have conducted our experiments on a collection of datasets from the repository at University of California at Irvine (Keogh et al., 1998). Two points need to be clarified on the way we used these datasets.

Firstly, some attributes in these datasets are continuous, while our learning algorithm is designed for discrete attributes. Hence, we needed to discretize some of the data. For a discretization algorithm, we have used equal-width interval binning discretization with 5 intervals. Although this method has been shown to be inferior to more sophisticated methods like entropy discretization of Fayad and Irani (1993), it is very easy to implement, very fast, and the performance difference is small (Dougherty et al., 1995). Missing attributes are handled by treating "missing" as a legitimate value.

Secondly, we had to inflate the datasets since we need large datasets but most of the UCI datasets are quite small. That is, following John and Langley (1996), we have artificially inflated the training set introducing 100 copies of all records and randomizing their order. Importantly, the test set has been left unchanged to avoid making the problem easier. Inflating the datasets only makes the problem harder in terms of running time, does not change the problem in terms of accuracy if we restrict ourselves to use algorithms that just calculate statistics about the training sample. For instance, in our case, if one particular decision stump has an accuracy of 80% on the original training set, then this stump has the same accuracy in the inflated dataset. Thus, if that is the stump of choice in the small dataset it will also be in the inflated version. Moreover, it also does not affect the results concerning sampling, neither makes the problem easier nor more difficult. The reason is that all the statistical bounds used here to calculate necessary sample sizes provide results that are independent of the size of the probability space from where we are sampling, i.e., the training set size. The necessary sample sizes depend on the probabilities on the training set and these are unchanged when we inflate the dataset. In other words, if in one particular situation the necessary sample size is 10,000, this sample size will be the same independently of whether we are sampling from the original dataset or from the inflated version. We are aware that this is perhaps not the best method to test our algorithms and real large datasets would have been better; but we still believe that the results are informative enough to show the goodness of our method.

We have chosen only datasets with 2 classes since, according to previous experiments on boosting stumps, we are not expecting our base learner to be able to find weak hypothesis with accuracy better than 50% for most problems with a large number of classes. Apart from this, the choice of datasets has been done so it reflects a variety of datasets sizes and combination of discrete and continuous attributes.

For every dataset, we had performed a 10-fold cross validation and for the algorithms using sampling (and thus, randomized) the results are averaged over 10 runs. All the experiments have been done in a computer with a CPU alpha 600 Mhz using 256 Mb of memory and a hard disk of 4.3 Gb running under Linux. Since enough memory was available, all the data has been loaded in a table in main memory and from there the algorithms have been run. Loading the data took few seconds and since this time is the same for all the algorithms it has been omitted from the results. Notice that for the algorithms not using sampling, this is the only way to run them. For the algorithms using sampling, it could have been done from disk using an efficient sampling method from external memory. Doing experiments under these conditions is part of our future work. The time taken to construct a set with all possible decision stumps is included in the total time. As a test bed for comparing our boosting decision stumps algorithms we have chosen two well known learning algorithms, the decision tree inducer C4.5 of Quinlan (1993) and the Naive Bayes classifier (both just used in a single run, not boosted). According to the discussion above about experiments with inflated datasets, for Naive Bayes classifier we had provided both, accuracy and running time results since this learning algorithm satisfies the requirements described before and therefore these results are meaningful. However, in the case of C4.5 we had only provided running time results, not accuracy results since this algorithm is not strictly based on probabilities over the sample. It makes, for instance, decisions about whether to split or not based on the actual number of instances following in one particular node and these numbers are obviously changed when we inflate the dataset. Thus, accuracy results of C4.5 in these sets cannot be used for comparison.

The experiments were done for estimating the performance of the arithmetic and geometric versions of Decision Stump Selector that are used by MadaBoost for boosting by filtering. For comparison, we executed MadaBoost for boosting by re-weighting with the whole dataset; that is, MadaBoost with the base learner that selects the best decision stump by searching trough the whole dataset exhaustively. Below we use Ar., Geo., and Exh. respectively to denote the arithmetic and the geometric versions of Decision Stump Selector and the exhaustive search selector. We also carried out the experiments with Exh. using the original AdaBoost and found the difference with MadaBoost with Exh. almost negligible. We have set the number of boosting rounds to be 10 which usually is enough to converge to a fixed training error (that is, although we keep obtaining hypothesis with accuracy slightly better than 50%, the training error does not get reduced anymore).

Table 1 shows the accuracy obtained on these datasets by three combinations of selectors with MadaBoost. The columns "Size" and "$\|H_{DS}\|$" show, for each dataset, its size and the number of all possible decision stumps respectively. As we can see easily, there is no significant difference between the accuracies obtained by these three methods. These results indicate that our sampling method is accurate enough. Moreover, even though the hypotheses produced are very simple (a weighted majority of ten depth-1 decision trees), the accuracies are comparable to that obtained using Naive Bayes, a learning algorithm that has been reported to be competitive with more sophisticated methods like decision trees. Once we have established that there is no loss in accuracy due to the use of sampling, we should check whether there is any gain in efficiency. Table 2 shows the running times of MadaBoost combined with three selection algorithms, exhaustive one (Exh.), and the arithmetic (Ar.)

Table 1. Accuracies of boosted decision stumps with and without sampling and that of Naive Bayes.

Name	Size	$\|H_{DS}\|$	DS	Exh.	Ar.	Geo.	NB
Agaricus	731100	296	88.68	97.74	97.84	98.03	98.82
Kr-vs-kp	287600	222	68.24	93.19	92.89	92.71	88.05
Hypothy.	284600	192	95.70	95.86	95.84	95.86	95.43
Sick-euthy.	284600	192	90.74	91.02	90.93	90.93	90.26
German	90000	222	69.90	74.10	74.28	74.30	76.00
Ionos	31590	408	76.18	90.26	89.53	89.63	89.14

Table 2. Running times (in seconds) of MadaBoost with and without sampling, and that of Naive Bayes and C4.5.

Name	Exh.	Ar.	Geo.	NB	C4.5
Agaricus	892.31	2.07	1.78	16.34	21.65
Kr-vs-kp	265.63	3.68	2.75	10.07	31.13
Hypothyroid	233.24	5.82	5.67	7.14	67.40
Sick-euthy.	232.05	6.84	6.77	7.08	162.76
German	80.75	16.96	10.47	1.08	20.34
Ionos	56.95	6.29	4.74	0.85	29.47

and the geometric (Geo.) versions. As we mentioned, we have also provided the running time of Naive Bayes and C4.5 for those datasets. The reason for doing that is that boosting is usually a slow process and thus, even though we are reducing its running time by using sampling, we still want to know how fast/slow is this running time compared to the running time of an state of the art learning method like C4.5 and to a very fast algorithm like Naive Bayes whose running time scales exactly linearly in the training set.

Let us discuss about these results. First, one can easily see that using all the dataset is a very slow process, particularly for large and complex datasets that need a large number of decision stumps. The running time of MadaBoost with Exh. is a function of the dataset size, the number of decision stumps considered (which depends on how many attributes the dataset has and their range of values), and the number of boosting rounds.

For the algorithms using sampling, we can see that the running time has been all greatly reduced. For instance, for MadaBoost with the arithmetic version, the running time is, on average, reduced about by $1/42$ of the Exh. case when including Agaricus, which is a particularly good case where the running time is reduced by $1/431$; even if we do not count Agaricus, the average reduction of running time is about $1/22$. With respect with the Geo. case, the running time has been reduced in average by a factor of $1/54$ times.

Surprisingly enough, for the sampling versions the fastest dataset becomes the largest one, Agaricus. It is due to the particular structure of this dataset. First, the decision stump size $\|H\|$ is of reasonable size. Second, during all the 10 boosting iterations one can find hypothesis with accuracy larger than 70% and thus, the sample sizes needed are very small.

This contrasts with datasets like German where a similar number of decision stumps is considered and, even though the dataset is less than $1/8$ of Agaricus, the running time on German is 8 times larger. This is because for this dataset, after the third boosting iteration, even the best stump has accuracy smaller than 60% and this affects the efficiency of the sampling method.

This difference between Agaricus and German becomes more clear by looking at the total number of examples used by a base learner, i.e., Decision Stump Selector, during the boosting iterations. Recall that we are using boosting by filtering; so when the base learner asks for an example, the filtering method might have to actually sample several of them until to get one that passes the filter and is given to the base learner. Our experiment shows that, in total, German needs 253,354 examples while Agaricus needs only 78,113 examples. On the other hand, the number of examples actually used by the base learner is 121,846 for German and 12,625 for Agaricus, around 10 times more.

As for the difference between arithmetic and geometric sampling, we can see that, for this particular problem and our choice of $s = 2$, the geometric sampling is, on average, 1.38 times faster than the arithmetic sampling, which is what we can expect from our theoretical estimation.

With respect to C4.5, we can see that our algorithm is faster in all datasets. More specifically, MadaBoost with the geometric sampling is around 10 times faster than C4.5 in average. Concerning Naive Bayes, the geometric version is faster in the three largest datasets, 3.28 times in average and slower in the three smallest, 2.43 in average although averaging over all datasets is slightly faster, 1.32 times.

5. Other applications

In this section we review some other data mining algorithms where our sampling method is potentially applicable. We remark here how one might try to apply the method and which problems you might encounter. Each of this application requires an individual deep study and experimentation in order to determine whether it is feasible or not. Our purpose in this section is just to encourage the application of adaptive sampling methods to these settings since we believe that might lead to increases in performance.

Association rules. We already studied this problem in Domingo et al. (1999) where our algorithm was used to estimate the support of the rules through sampling. There, the distance of the support of a rule being estimated to the minimum support was used as utility function. Although preliminary experiments reported there were very encouraging there is still further work to integrate our method in an overall algorithm for mining association rules like Apriori. It should be noticed that in this case, the rule set does not necessarily correspond to the set of all possible itemsets since this would make the problem computationally infeasible. For instance, if our algorithm is used in combination with Apriori, it could be used to estimate which candidate itemsets are frequent or not at each step of the algorithm. Notice that in this case, the number of rules considered is not the overall number of possible rules but a very small fraction of it.

Naive Bayes. This induction system seems very suitable for our algorithm. It works by creating a set of discriminant functions, one for each class which are just certain probabilities

obtained from the training set. However, this application is not as straightforward as it seems. One should carefully determine what will be the most appropriate choice for the utility function and also we should decide how to deal with of some probabilities having 0 or very small value. Again, we can see here that our method will provide a sampling procedure not for the overall algorithm as it was done in John and Langley (1996) but for a subprocedure of it, for example, one for every discriminant function.

Decision tree induction. This application is the most challenging due to the well established efficiency of decision trees and its appeal for human interpretation. For this problem our sampling method might be used to decide, every time a new node is being constructed, on which attribute we should split. Intuitively, the splitting criteria should be used as the utility function since it is a function on an average over the dataset. It is also clear that one might not need to go trough all the data to decide which split is the best (or close to the best) and for here is where our sampling method might be useful. However, this application is far from trivial due to the complexity of some commonly used splitting functions, like the gain ratio, and due to the fragmentation problem. Even though the amount of data falling at nodes close to the root might be large enough to use sampling to decide which split to use, the larger we build the tree the smallest becomes the amount of data available at each node. Thus, the effectiveness of sampling might be lost if not cleverly applied.

6. Concluding remarks

We have presented a new methodology for sampling that, while keeping all the theoretical guarantees of previous ones, it is applicable in a wider class of tasks. The key point is to perform sampling sequentially and determine the time to stop sampling by a carefully designed stopping condition. We theoretically give both justification and efficiency analysis of our algorithms, and then provide experimental evidence of the advantage of our method. In order to give some concrete example, we fix one specific problem, i.e., the design of a base learner for a boosting algorithm, and discuss how our method can be used. There are many other settings, discussed in Section 5, where our method might well be useful. We plan to study these in the future.

Acknowledgments

We would like to thank Heikki Mannila for pointing us the work in adaptive sampling for database query estimation, Pedro Domingos for pointing us to several machine learning papers related to this work, and Tadashi Yamazaki for technical help with the experiments.

Note

1. The only assumption is that we can obtain examples independently under the same distribution, a natural assumption that holds for all problems considered here.

References

Bauer, E. and Kohavi, R. 1998. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning, 36:105–139.

Domingo, C., Gavaldà, R., and Watanabe, R. 1998. Practical algorithms for on-line selection. In Proceedings of the First International Conference on Discovery Science (DS'98), Lecture Notes in Artificial Intelligence, vol. 1532. (pp. 150–161).

Domingo, C., Gavaldà, R., and Watanabe, O. 1999. On-line sampling methods for discovering association rules. Dept. of Math and Computing Science, Tokyo Institute of Technology, Tokyo Tech Rep. C-126.

Domingo, C. and Watanabe, O. 1999. MadaBoost: A modification of AdaBoost for the filtering framework. Dept. of Math and Computing Science, Tokyo Institute of Technology, Tokyo, Tech Rep. C-138. Available at www.is.titech.ac.jp/research/research-report/C/index.html. Also submitted for publication.

Domingo, C. and Watanabe, O. 2000a. Scaling-up a boosting based learner via Adaptive Sampling. In Proceedings of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'2000), to appear.

Domingo, C. and Watanabe. O. 2000b. Experimental evaluation of an adaptive boosting by filtering algorithm. Submitted for publication. Dept. of Math and Comp. Science, Tokyo Insitute of Technology, Tokyo, Tech Rep. C-139. Available at www.is.titech.ac.jp/research/research-report/C/index.html.

Dougherty, J., Kohavi, R., and Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. In Proceedings of the Twelfth International Conference on Machine Learning.

Fayad, U.M. and Irani, K.B. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. In Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1022–1027.

Feller, W. 1950. An Introduction to Probability Theory and its Applications. New York: John Willey and Sons.

Freund, Y. and Schapire, R.E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139.

John, G.H. and Langley, P. 1996. Static versus dynamic sampling for data mining. In Proc. of the Second International Conference on Knowledge Discovery and Data Mining. AAAI/MIT Press.

Kearns, M.J. and Vazirani, U. 1994. An Introduction to Computational Learning Theory. Cambridge: Cambridge University Press.

Keogh, E., Blake, C. and Merz, C.J. 1998. UCI repository of machine learning databases. Available at http://www.ics.uci.ed u/ mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

Kivinen, J. and Mannila, H. 1994. The power of sampling in knowledge discovery. In Proceedings of the ACM SIGACT-SIGMOD-SIGACT Symposium on Principles of Database Theory, pp. 77–85.

Lipton, R.J. and Naughton, J.F. 1995. Query size estimation by adaptive sampling. Journal of Computer and System Science, 51:18–25.

Lipton, R.J., Naughton, J.F., Schneider, D.A. and Seshadri, S. 1995. Efficient sampling strategies for relational database operations. Theoretical Computer Science, 116:195–226.

Provost, F., Jensen, D., and Oates, T. 1999. Efficient progressive sampling. In Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining.

Quinlan, J.R. 1993. C4.5: Programs for machine learning. San Mateo, CA: Morgan Kaufmann.

Toivonen, H. 1996. Sampling large databases for association rules. In Proceedings of the 22nd International Conference on Very Large Databases, pp. 134–145.

Wald, A. 1947. Sequential Analysis. Wiley Mathematical, Statistics Series. New York: Wiley.

Wang, M., Iyer, B., and Vitter, J.S., 1998. MIND: A scalable mining for classifier in relational databases. In Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD).

Carlos Domingo holds a bachelor in Computer Science from the Universitat Politecnica de Catalunya (UPC), Spain, a Master in Computer Science from the Tokyo Institute of Technology (Titech), Japan, and a PhD in Computer Science form UPC in 1998. Until 2000 he was an European Union Science and Technology Fellow at the Tokyo Institute of Technology doing research on computational learning theory, machine learning and data mining. Curently he is the head of the 3Gengine division for Internet and mobile Internet applications at Synera Systems S.L. (www.synerasystems.com).

Ricard Gavaldà received his Ph.D. in Computer Science at the Technical University of Catalonia (UPC), Spain in 1992. He is an associate professor at the Department of Software, UPC, since 1993. He worked mostly in computational complexity theory for several years. More recent research interests include also algorithmic aspects of machine learning and data mining.

Osamu Watanabe received in 1982 Master of Science from Tokyo Institute of Technology (Titech) and received Dr. of Engineering in Computer Science from Titech in 1987. He is a professor at the Dept. Mathematical and Computing Science, Titech, since 1997. His research interests are computational complexity theory and design and analysis of algorithms.