Mining High-Speed Data Streams

Pedro Domingos
Dept. of Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350, U.S.A.
pedrod@cs.washington.edu

Geoff Hulten
Dept. of Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350, U.S.A.
ghulten@cs.washington.edu

ABSTRACT

Many organizations today have more than very large databases; they have databases that grow without limit at a rate of several million records per day. Mining these continuous data streams brings unique opportunities, but also new challenges. This paper describes and evaluates VFDT, an anytime system that builds decision trees using constant memory and constant time per example. VFDT can incorporate tens of thousands of examples per second using off-the-shelf hardware. It uses Hoeffding bounds to guarantee that its output is asymptotically nearly identical to that of a conventional learner. We study VFDT's properties and demonstrate its utility through an extensive set of experiments on synthetic data. We apply VFDT to mining the continuous stream of Web access data from the whole University of Washington main campus.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning*; I.5.2 [**Pattern Recognition**]: Design Methodology—*classifier design and evaluation*

General Terms

Decision trees, Hoeffding bounds, incremental learning, disk-based algorithms, subsampling

1. INTRODUCTION

Knowledge discovery systems are constrained by three main limited resources: time, memory and sample size. In traditional applications of machine learning and statistics, sample size tends to be the dominant limitation: the computational resources for a massive search are available, but carrying out such a search over the small samples available (typically less than 10,000 examples) often leads to overfitting or "data dredging" (e.g., [22, 16]). Thus overfitting avoidance becomes the main concern, and only a fraction of the available computational power is used [3]. In contrast, in many (if not most) present-day data mining applications, the bottleneck is time and memory, not examples. The latter are typically in over-supply, in the sense that it is impossible with current KDD systems to make use of all of them within the available computational resources. As a result, most of the available examples go unused, and underfitting may result: enough data to model very complex phenomena is available, but inappropriately simple models are produced because we are unable to take full advantage of the data. Thus the development of highly efficient algorithms becomes a priority.

Currently, the most efficient algorithms available (e.g., [17]) concentrate on making it possible to mine databases that do not fit in main memory by only requiring sequential scans of the disk. But even these algorithms have only been tested on up to a few million examples. In many applications this is less than a day's worth of data. For example, every day retail chains record millions of transactions, telecommunications companies connect millions of calls, large banks process millions of ATM and credit card operations, and popular Web sites log millions of hits. As the expansion of the Internet continues and ubiquitous computing becomes a reality, we can expect that such data volumes will become the rule rather than the exception. Current data mining systems are not equipped to cope with them. When new examples arrive at a higher rate than they can be mined, the quantity of unused data grows without bounds as time progresses. Even simply preserving the examples for future use can be a problem when they need to be sent to tertiary storage, are easily lost or corrupted, or become unusable when the relevant contextual information is no longer available. When the source of examples is an open-ended data stream, the notion of mining a database of fixed size itself becomes questionable.

Ideally, we would like to have KDD systems that operate continuously and indefinitely, incorporating examples as they arrive, and never losing potentially valuable information. Such desiderata are fulfilled by incremental learning methods (also known as online, successive or sequential methods), on which a substantial literature exists. However, the available algorithms of this type (e.g., [20]) have significant shortcomings from the KDD point of view. Some are reasonably efficient, but do not guarantee that the model learned will be similar to the one obtained by learning on the same data in batch mode. They are highly sensitive to example ordering, potentially never recovering from an unfavorable set of early examples. Others produce the same model as the batch version, but at a high cost in efficiency, often to the point of being slower than the batch algorithm.

This paper proposes Hoeffding trees, a decision-tree learning method that overcomes this trade-off. Hoeffding trees can be learned in constant time per example (more precisely, in time that is worst-case proportional to the number of attributes), while being nearly identical to the trees a conventional batch learner would produce, given enough examples. The probability that the Hoeffding and conventional tree learners will choose different tests at any given node decreases exponentially with the number of examples. We also describe and evaluate VFDT, a decision-tree learning system based on Hoeffding trees. VFDT is I/O bound in the sense that it mines examples in less time than it takes to input them from disk. It does not store any examples (or parts thereof) in main memory, requiring only space proportional to the size of the tree and associated sufficient statistics. It can learn by seeing each example only once, and therefore does not require examples from an online stream to ever be stored. It is an anytime algorithm in the sense that a ready-to-use model is available at any time after the first few examples are seen, and its quality increases smoothly with time.

The next section introduces Hoeffding trees and studies their properties. We then describe the VFDT system and its empirical evaluation. The paper concludes with a discussion of related and future work.

2. HOEFFDING TREES

The classification problem is generally defined as follows. A set of N training examples of the form $(\mathbf{x}, y)$ is given, where y is a discrete class label and $\mathbf{x}$ is a vector of d attributes, each of which may be symbolic or numeric. The goal is to produce from these examples a model $y = f(\mathbf{x})$ that will predict the classes y of future examples $\mathbf{x}$ with high accuracy. For example, $\mathbf{x}$ could be a description of a client's recent purchases, and y the decision to send that customer a catalog or not; or $\mathbf{x}$ could be a record of a cellular-telephone call, and y the decision whether it is fraudulent or not. One of the most effective and widely-used classification methods is decision tree learning [1, 15]. Learners of this type induce models in the form of decision trees, where each node contains a test on an attribute, each branch from a node corresponds to a possible outcome of the test, and each leaf contains a class prediction. The label $y = DT(\mathbf{x})$ for an example $\mathbf{x}$ is obtained by passing the example down from the root to a leaf, testing the appropriate attribute at each node and following the branch corresponding to the attribute's value in the example. A decision tree is learned by recursively replacing leaves by test nodes, starting at the root. The attribute to test at a node is chosen by comparing all the available attributes and choosing the best one according to some heuristic measure. Classic decision tree learners like ID3, C4.5 and CART assume that all training examples can be stored simultaneously in main memory, and are thus severely limited in the number of examples they can learn from. Disk-based decision tree learners like SLIQ [10] and SPRINT [17] assume the examples are stored on disk, and learn by repeatedly reading them in sequentially (effectively once per level in the tree). While this greatly increases the size of usable training sets, it can become prohibitively expensive when learning complex trees (i.e., trees with many levels), and fails when datasets are too large to fit in the available disk space.

Our goal is to design a decision tree learner for extremely large (potentially infinite) datasets. This learner should require each example to be read at most once, and only a small constant time to process it. This will make it possible to directly mine online data sources (i.e., without ever storing the examples), and to build potentially very complex trees with acceptable computational cost. We achieve this by noting with Catlett [2] and others that, in order to find the best attribute to test at a given node, it may be sufficient to consider only a small subset of the training examples that pass through that node. Thus, given a stream of examples, the first ones will be used to choose the root test; once the root attribute is chosen, the succeeding examples will be passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on recursively.[1] We solve the difficult problem of deciding exactly how many examples are necessary at each node by using a statistical result known as the *Hoeffding bound* (or additive Chernoff bound) [7, 9]. Consider a real-valued random variable r whose range is R (e.g., for a probability the range is one, and for an information gain the range is $\log c$, where c is the number of classes). Suppose we have made n independent observations of this variable, and computed their mean $\overline{r}$. The Hoeffding bound states that, with probability $1 - \delta$, the true mean of the variable is at least $\overline{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{1}$$

The Hoeffding bound has the very attractive property that it is independent of the probability distribution generating the observations. The price of this generality is that the bound is more conservative than distribution-dependent ones (i.e., it will take more observations to reach the same δ and ϵ). Let $G(X_i)$ be the heuristic measure used to choose test attributes (e.g., the measure could be information gain as in C4.5, or the Gini index as in CART). Our goal is to ensure that, with high probability, the attribute chosen using n examples (where n is as small as possible) is the same that would be chosen using infinite examples. Assume G is to be maximized, and let X_a be the attribute with highest observed $\overline{G}$ after seeing n examples, and X_b be the second-best attribute. Let $\Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b) \geq 0$ be the difference between their observed heuristic values. Then, given a desired δ, the Hoeffding bound guarantees that X_a is the correct choice with probability $1 - \delta$ if n examples have been seen at this node and $\Delta\overline{G} > \epsilon$.[2] In other words, if the ob-

[1] We assume the examples are generated by a stationary stochastic process (i.e., their distribution does not change over time). If the examples are being read from disk, we assume that they are in random order. If this is not the case, they should be randomized, for example by creating a random index and sorting on it.

[2] In this paper we assume that the third-best and lower attributes have sufficiently smaller gains that their probability of being the true best choice is negligible. We plan to lift this assumption in future work. If the attributes at a given node are (pessimistically) assumed independent, it simply involves a Bonferroni correction to δ [11].

served $\Delta\overline{G} > \epsilon$ then the Hoeffding bound guarantees that the true $\Delta G \geq \Delta\overline{G} - \epsilon > 0$ with probability $1 - \delta$, and therefore that X_a is indeed the best attribute with probability $1 - \delta$. This is valid as long as the $\overline{G}$ value for a node can be viewed as an average of G values for the examples at that node, as is the case for the measures typically used. Thus a node needs to accumulate examples from the stream until ϵ becomes smaller than $\Delta\overline{G}$. (Notice that ϵ is a monotonically decreasing function of n.) At this point the node can be split using the current best attribute, and succeeding examples will be passed to the new leaves. This leads to the *Hoeffding tree algorithm*, shown in pseudo-code in Table 1.

The counts n_{ijk} are the sufficient statistics needed to compute most heuristic measures; if other quantities are required, they can be similarly maintained. Pre-pruning is carried out by considering at each node a "null" attribute $X_\emptyset$ that consists of not splitting the node. Thus a split will only be made if, with confidence $1-\delta$, the best split found is better according to G than not splitting. The pseudo-code shown is only for discrete attributes, but its extension to numeric ones is immediate, following the usual method of allowing tests of the form "$(X_i < x_{ij})$?," and computing $\overline{G}$ for each allowed threshold x_{ij}. The sequence of examples S may be infinite, in which case the procedure never terminates, and at any point in time a parallel procedure can use the current tree HT to make class predictions. If d is the number of attributes, v is the maximum number of values per attribute, and c is the number of classes, the Hoeffding tree algorithm requires $O(dvc)$ memory to store the necessary counts at each leaf. If l is the number of leaves in the tree, the total memory required is $O(ldvc)$. This is independent of the number of examples seen, if the size of the tree depends only on the "true" concept and is independent of the size of the training set. (Although this is a common assumption in the analysis of decision-tree and related algorithms, it often fails in practice. Section 3 describes a refinement to the algorithm to cope with this.)

A key property of the Hoeffding tree algorithm is that it is possible to guarantee under realistic assumptions that the trees it produces are asymptotically arbitrarily close to the ones produced by a batch learner (i.e., a learner that uses all the examples to choose a test at each node). In other words, the incremental nature of the Hoeffding tree algorithm does not significantly affect the quality of the trees it produces. In order to make this statement precise, we need to define the notion of *disagreement* between two decision trees. Let $P(\mathbf{x})$ be the probability that the attribute vector (loosely, example) $\mathbf{x}$ will be observed, and let $I(.)$ be the indicator function, which returns 1 if its argument is true and 0 otherwise.

DEFINITION 1. *The* extensional disagreement Δ_e *between two decision trees* DT_1 *and* DT_2 *is the probability that they will produce different class predictions for an example:*

$$\Delta_e(DT_1, DT_2) = \sum_{\mathbf{x}} P(\mathbf{x}) I[DT_1(\mathbf{x}) \neq DT_2(\mathbf{x})]$$

Consider that two internal nodes are different if they contain different tests, two leaves are different if they contain different class predictions, and an internal node is different from a leaf. Consider also that two paths through trees are different if they differ in length or in at least one node.

Table 1: The Hoeffding tree algorithm.

Inputs:	S	is a sequence of examples,
	$\mathbf{X}$	is a set of discrete attributes,
	$G(.)$	is a split evaluation function,
	δ	is one minus the desired probability of choosing the correct attribute at any given node.
Output:	HT	is a decision tree.

Procedure HoeffdingTree $(S, \mathbf{X}, G, \delta)$
Let HT be a tree with a single leaf l_1 (the root).
Let $\mathbf{X_1} = \mathbf{X} \cup \{X_\emptyset\}$.
Let $\overline{G}_1(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S.
For each class y_k
  For each value x_{ij} of each attribute $X_i \in \mathbf{X}$
    Let $n_{ijk}(l_1) = 0$.
For each example $(\mathbf{x}, y_k)$ in S
  Sort $(\mathbf{x}, y)$ into a leaf l using HT.
  For each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
    Increment $n_{ijk}(l)$.
  Label l with the majority class among the examples seen so far at l.
  If the examples seen so far at l are not all of the same class, then
    Compute $\overline{G}_l(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$.
    Let X_a be the attribute with highest $\overline{G}_l$.
    Let X_b be the attribute with second-highest $\overline{G}_l$.
    Compute ϵ using Equation 1.
    If $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \epsilon$ and $X_a \neq X_\emptyset$, then
      Replace l by an internal node that splits on X_a.
      For each branch of the split
        Add a new leaf l_m, and let $\mathbf{X_m} = \mathbf{X} - \{X_a\}$.
        Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m.
        For each class y_k and each value x_{ij} of each attribute $X_i \in \mathbf{X_m} - \{X_\emptyset\}$
          Let $n_{ijk}(l_m) = 0$.
Return HT.

DEFINITION 2. *The* intensional disagreement Δ_i *between two decision trees* DT_1 *and* DT_2 *is the probability that the path of an example through* DT_1 *will differ from its path through* DT_2*:*

$$\Delta_i(DT_1, DT_2) = \sum_{\mathbf{x}} P(\mathbf{x}) I[Path_1(\mathbf{x}) \neq Path_2(\mathbf{x})]$$

where $Path_i(\mathbf{x})$ *is the path of example* $\mathbf{x}$ *through tree* DT_i*.*

Two decision trees agree intensionally on an example iff they are indistinguishable for that example: the example is passed down exactly the same sequence of nodes, and receives an identical class prediction. Intensional disagreement is a stronger notion than extensional disagreement, in the sense that $\forall_{DT_1,DT_2}\ \Delta_i(DT_1, DT_2) \geq \Delta_e(DT_1, DT_2)$.

Let p_l be the probability that an example that reaches level l in a decision tree falls into a leaf at that level. To simplify, we will assume that this probability is constant, i.e., $\forall_l \; p_l = p$, where p will be termed the *leaf probability.* This is a realistic assumption, in the sense that it is typically approximately true for the decision trees that are generated in practice. Let HT_δ be the tree produced by the Hoeffding tree algorithm with desired probability δ given an infinite sequence of examples S, and DT_* be the asymptotic batch decision tree induced by choosing at each node the attribute with true greatest G (i.e., by using infinite examples at each node). Let $E[\Delta_i(HT_\delta, DT_*)]$ be the expected value of $\Delta_i(HT_\delta, DT_*)$, taken over all possible infinite training sequences. We can then state the following result.

THEOREM 1. *If HT_δ is the tree produced by the Hoeffding tree algorithm with desired probability δ given infinite examples (Table 1), DT_* is the asymptotic batch tree, and p is the leaf probability, then $E[\Delta_i(HT_\delta, DT_*)] \leq \delta/p$.*

Proof. For brevity, we will refer to intensional disagreement simply as disagreement. Consider an example $\mathbf{x}$ that falls into a leaf at level l_h in HT_δ, and into a leaf at level l_d in DT_*. Let $l = \min\{l_h, l_d\}$. Let $\text{Path}_H(\mathbf{x}) = (N_1^H(\mathbf{x}), N_2^H(\mathbf{x}), \ldots, N_l^H(\mathbf{x}))$ be $\mathbf{x}$'s path through HT_δ up to level l, where $N_i^H(\mathbf{x})$ is the node that $\mathbf{x}$ goes through at level i in HT_δ, and similarly for $\text{Path}_D(\mathbf{x})$, $\mathbf{x}$'s path through DT_*. If $l = l_h$ then $N_l^H(\mathbf{x})$ is a leaf with a class prediction, and similarly for $N_l^D(\mathbf{x})$ if $l = l_d$. Let I_i represent the proposition "$\text{Path}_H(\mathbf{x}) = \text{Path}_D(\mathbf{x})$ up to and including level i," with $I_0 = \text{True}$. Notice that $P(l_h \neq l_d)$ is included in $P(N_l^H(\mathbf{x}) \neq N_l^D(\mathbf{x})|I_{l-1})$, because if the two paths have different lengths then one tree must have a leaf where the other has an internal node. Then, omitting the dependency of the nodes on $\mathbf{x}$ for brevity,

$$\begin{aligned} & P(\text{Path}_H(\mathbf{x}) \neq \text{Path}_D(\mathbf{x})) \\ & \quad = \; P(N_1^H \neq N_1^D \vee N_2^H \neq N_2^D \vee \ldots \vee N_l^H \neq N_l^D) \\ & \quad = \; P(N_1^H \neq N_1^D|I_0) + P(N_2^H \neq N_2^D|I_1) + \ldots \\ & \qquad\; + P(N_l^H \neq N_l^D|I_{l-1}) \\ & \quad = \; \sum_{i=1}^{l} P(N_i^H \neq N_i^D|I_{i-1}) \;\leq\; \sum_{i=1}^{l} \delta \;=\; \delta l \qquad (2) \end{aligned}$$

Let $HT_\delta(S)$ be the Hoeffding tree generated from training sequence S. Then $E[\Delta_i(HT_\delta, DT_*)]$ is the average over all infinite training sequences S of the probability that an example's path through $HT_\delta(S)$ will differ from its path through DT_*:

$$\begin{aligned} & E[\Delta_i(HT_\delta, DT_*)] \\ & \quad = \; \sum_S P(S) \sum_{\mathbf{x}} P(\mathbf{x}) \, I[\text{Path}_H(\mathbf{x}) \neq \text{Path}_D(\mathbf{x})] \\ & \quad = \; \sum_{\mathbf{x}} P(\mathbf{x}) \, P(\text{Path}_H(\mathbf{x}) \neq \text{Path}_D(\mathbf{x})) \\ & \quad = \; \sum_{i=1}^{\infty} \sum_{\mathbf{x} \in L_i} P(\mathbf{x}) \, P(\text{Path}_H(\mathbf{x}) \neq \text{Path}_D(\mathbf{x})) \qquad (3) \end{aligned}$$

where L_i is the set of examples that fall into a leaf of DT_* at level i. According to Equation 2, the probability that an example's path through $HT_\delta(S)$ will differ from its path through DT_*, given that the latter is of length i, is at most δi (since $i \geq l$). Thus

$$\begin{aligned} E[\Delta_i(HT_\delta, DT_*)] & \leq \sum_{i=1}^{\infty} \sum_{\mathbf{x} \in L_i} P(\mathbf{x})(\delta i) \\ & = \sum_{i=1}^{\infty} (\delta i) \sum_{\mathbf{x} \in L_i} P(\mathbf{x}) \qquad (4) \end{aligned}$$

The sum $\sum_{\mathbf{x} \in L_i} P(\mathbf{x})$ is the probability that an example $\mathbf{x}$ will fall into a leaf of DT_* at level i, and is equal to $(1-p)^{i-1}p$, where p is the leaf probability. Therefore

$$\begin{aligned} & E[\Delta_i(HT_\delta, DT_*)] \\ & \leq \; \sum_{i=1}^{\infty} (\delta i)(1-p)^{i-1} p \;=\; \delta p \sum_{i=1}^{\infty} i(1-p)^{i-1} \\ & = \; \delta p \left[\sum_{i=1}^{\infty} (1-p)^{i-1} + \sum_{i=2}^{\infty} (1-p)^{i-1} + \cdots \right. \\ & \qquad \left. + \sum_{i=k}^{\infty} (1-p)^{i-1} + \cdots \right] \\ & = \; \delta p \left[\frac{1}{p} + \frac{1-p}{p} + \cdots + \frac{(1-p)^{k-1}}{p} + \cdots \right] \\ & = \; \delta \left[1 + (1-p) + \cdots + (1-p)^{k-1} + \cdots \right] \\ & = \; \delta \sum_{i=0}^{\infty} (1-p)^i \;=\; \frac{\delta}{p} \qquad (5) \end{aligned}$$

This completes the demonstration of Theorem 1. □

An immediate corollary of Theorem 1 is that the expected extensional disagreement between HT_δ and DT_* is also asymptotically at most δ/p (although in this case the bound is much looser). Another corollary (whose proof we omit here in the interests of space) is that there exists a subtree of the asymptotic batch tree such that the expected disagreement between it and the Hoeffding tree learned on finite data is at most δ/p. In other words, if δ/p is small then the Hoeffding tree learned on finite data is very similar to a subtree of the asymptotic batch tree. A useful application of Theorem 1 is that, instead of δ, users can now specify as input to the Hoeffding tree algorithm the maximum expected disagreement they are willing to accept, given enough examples for the tree to settle. The latter is much more meaningful, and can be intuitively specified without understanding the workings of the algorithm or the Hoeffding bound. The algorithm will also need an estimate of p, which can easily be obtained (for example) by running a conventional decision tree learner on a manageable subset of the data. How practical are these bounds? Suppose that the best and second-best attribute differ by 10% (i.e., $\epsilon/R = 0.1$). Then, according to Equation 1, ensuring $\delta = 0.1\%$ requires 380 examples, and ensuring $\delta = 0.0001\%$ requires only 345 additional examples. An exponential improvement in δ, and therefore in expected disagreement, can be obtained with a linear increase in the number of examples. Thus, even with very small leaf probabilities (i.e., very large trees), very good agreements can be obtained with a relatively small number of examples per node. For example, if $p = 0.01\%$, an expected disagreement of at most 1% can be guaranteed with 725 examples per node. If $p = 1\%$, the same number of examples guarantees a disagreement of at most 0.01%.

3. THE VFDT SYSTEM

We have implemented a decision-tree learning system based on the Hoeffding tree algorithm, which we call VFDT (Very Fast Decision Tree learner). VFDT allows the use of either information gain or the Gini index as the attribute evaluation measure. It includes a number of refinements to the algorithm in Table 1:

Ties. When two or more attributes have very similar G's, potentially many examples will be required to decide between them with high confidence. This is presumably wasteful, because in this case it makes little difference which attribute is chosen. Thus VFDT can optionally decide that there is effectively a tie and split on the current best attribute if $\Delta\overline{G} < \epsilon < \tau$, where τ is a user-specified threshold.

G computation. The most significant part of the time cost per example is recomputing G. It is inefficient to recompute G for every new example, because it is unlikely that the decision to split will be made at that specific point. Thus VFDT allows the user to specify a minimum number of new examples n_{min} that must be accumulated at a leaf before G is recomputed. This effectively reduces the global time spent on G computations by a factor of n_{min}, and can make learning with VFDT nearly as fast as simply classifying the training examples. Notice, however, that it will have the effect of implementing a smaller δ than the one specified by the user, because examples will be accumulated beyond the strict minimum required to choose the correct attribute with confidence $1 - \delta$. (This increases the time required to build a node, but our experiments show that the net effect is still a large speedup.) Because δ shrinks exponentially fast with the number of examples, the difference could be large, and the δ input to VFDT should be correspondingly larger than the target.

Memory. As long as VFDT processes examples faster than they arrive, which will be the case in all but the most demanding applications, the sole obstacle to learning arbitrarily complex models will be the finite RAM available. VFDT's memory use is dominated by the memory required to keep counts for all growing leaves. If the maximum available memory is ever reached, VFDT deactivates the least promising leaves in order to make room for new ones. If p_l is the probability that an arbitrary example will fall into leaf l, and e_l is the observed error rate at that leaf, then $p_l e_l$ is an upper bound on the error reduction achievable by refining the leaf. $p_l e_l$ for a new leaf is estimated using the counts at the parent for the corresponding attribute value. The least promising leaves are considered to be the ones with the lowest values of $p_l e_l$. When a leaf is deactivated, its memory is freed, except for a single number required to keep track of $p_l e_l$. A leaf can then be reactivated if it becomes more promising than currently active leaves. This is accomplished by, at regular intervals, scanning through all the active and inactive leaves, and replacing the least promising active leaves with the inactive ones that dominate them.

Poor attributes. Memory usage is also minimized by dropping early on attributes that do not look promising. As soon as the difference between an attribute's G and the best one's becomes greater than ϵ, the attribute can be dropped from consideration, and the memory used to store the corresponding counts can be freed.

Initialization. VFDT can be initialized with the tree produced by a conventional RAM-based learner on a small subset of the data. This tree can either be input as is, or over-pruned to contain only those nodes that VFDT would have accepted given the number of examples at them. This can give VFDT a "head start" that will allow it to reach the same accuracies at smaller numbers of examples throughout the learning curve.

Rescans. VFDT can rescan previously-seen examples. This option can be activated if either the data arrives slowly enough that there is time for it, or if the dataset is finite and small enough that it is feasible to scan it multiple times. This means that VFDT need never grow a smaller (and potentially less accurate) tree than other algorithms because of using each example only once.

The next section describes an empirical study of VFDT, where the utility of these refinements is evaluated.

4. EMPIRICAL STUDY

4.1 Synthetic data

A system like VFDT is only useful if it is able to learn more accurate trees than a conventional system, given similar computational resources. In particular, it should be able to use to advantage the examples that are beyond a conventional system's ability to process. In this section we test this empirically by comparing VFDT with C4.5 release 8 [15] on a series of synthetic datasets. Using these allows us to freely vary the relevant parameters of the learning process. In order to ensure a fair comparison, we restricted the two systems to using the same amount of RAM. This was done by setting VFDT's "available memory" parameter to 40MB, and giving C4.5 the maximum number of examples that would fit in the same memory (100k examples).[3] VFDT used information gain as the G function. Fourteen concepts were used for comparison, all with two classes and 100 binary attributes. The concepts were created by randomly generating decision trees as follows. At each level after the first three, a fraction f of the nodes was replaced by leaves; the rest became splits on a random attribute (that had not been used yet on a path from the root to the node being considered). When the decision tree reached a depth of 18, all the remaining growing nodes were replaced with leaves. Each leaf was randomly assigned a class. The size of the resulting concepts ranged from 2.2k leaves to 61k leaves with a median of 12.6k. A stream of training examples was then

[3]VFDT occasionally grew slightly beyond 40MB because the limit was only enforced on heap-allocated memory. C4.5 always exceeded 40MB by the size of the unpruned tree.

generated by sampling uniformly from the instance space, and assigning classes according to the target tree. We added various levels of class and attribute noise to the training examples, from 0 to 30%.[4] (A noise level of n% means that each class/attribute value has a probability of n% of being reassigned at random, with equal probability for all values, including the original one.) In each run, 50k separate examples were used for testing. C4.5 was run with all default settings. We ran our experiments on two Pentium 6/200 MHz, one Pentium II/400 MHz, and one Pentium III/500 MHz machine, all running Linux.

Figure 1 shows the accuracy of the learners averaged over all the runs. VFDT was run with $\delta = 10^{-7}$, $\tau = 5\%$, $n_{min} = 200$, no leaf reactivation, and no rescans. VFDT-boot is VFDT bootstrapped with an over-pruned version of the tree produced by C4.5. C4.5 is more accurate than VFDT up to 25k examples, and the accuracies of the two systems are similar in the range from 25k to 100k examples (at which point C4.5 is unable to consider further examples). Most significantly, VFDT is able to take advantage of the examples after 100k to greatly improve accuracy (88.7% for VFDT and 88.8% for VFDT-boot, vs. 76.5% for C4.5). C4.5's early advantage comes from the fact it reuses examples to make decisions on multiple levels of the tree it is inducing, while VFDT uses each example only once. As expected, VFDT-boot's initialization lets it achieve high accuracy more quickly than without it. However, VFDT-boot's performance is surprising in that its accuracy is much higher than C4.5's at 100k examples, when VFDT-boot has not seen any examples that C4.5 did not. An explanation for this is that many of the experiments reported in Figure 1 contained noise, and, as Catlett [2] showed, over-pruning can be very effective at reducing overfitting in noisy domains.

Figure 2 shows the average number of nodes in the trees induced by each of the learners. Notice that VFDT and VFDT-boot induce trees with similar numbers of nodes, and that both achieve greater accuracy with far fewer nodes than C4.5. This suggests that using VFDT can substantially increase the comprehensibility of the trees induced relative to C4.5. It also suggests that VFDT is less prone than C4.5 to overfitting noisy data.

Figure 3 shows how the algorithms respond to noise. It compares four runs on the same concept (with 12.6k leaves), but with increasing levels of noise added to the training examples. C4.5's accuracy reports are for training sets with 100k examples, and VFDT and VFDT-boot's are for training sets of 20 million examples. VFDT's advantage compared to C4.5 increases with the noise level. This is further evidence that use of the Hoeffding bound is an effective pruning method.

[4]The exact concepts used were, in the form *(f, noise level, #nodes, #leaves)*: (0.15, 0.10, 74449, 37225), (0.15, 0.10, 13389, 6695), (0.17, 0.10, 78891, 39446), (0.17, 0.10, 93391, 46696), (0.25, 0.00, 25209, 12605), (0.25, 0.20, 25209, 12605), (0.25, 0.30, 25209, 12605), (0.25, 0.00, 15917, 7959), (0.25, 0.10, 31223, 15612), (0.25, 0.15, 16781, 8391), (0.25, 0.20, 4483, 2242), (0.28, 0.10, 122391, 61196), (0.28, 0.10, 6611, 3306), (0.25, 0.10, 25209, 12605). The last set of parameters was also used as the basis for the lesion studies reported below.

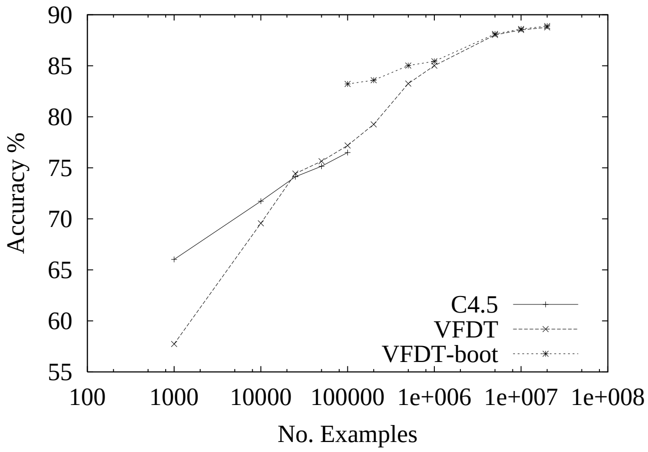

Figure 1: Accuracy as a function of the number of training examples.

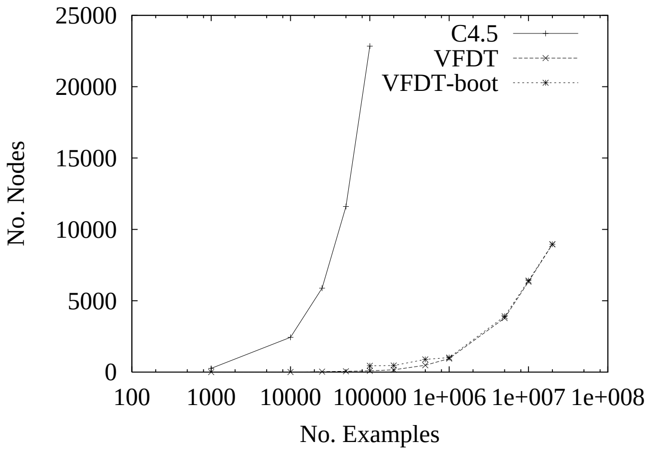

Figure 2: Tree size as a function of the number of training examples.

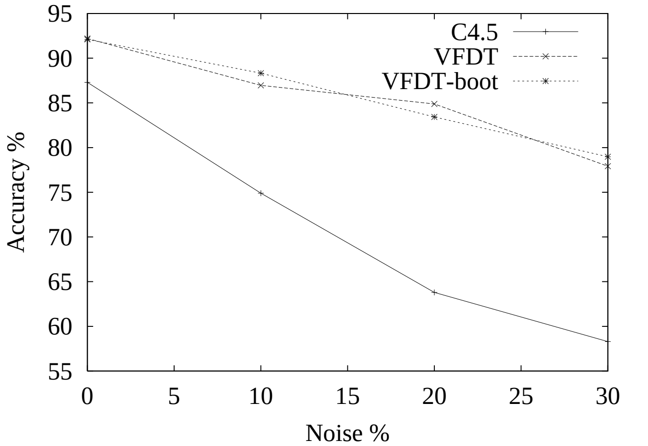

Figure 3: Accuracy as a function of the noise level.

Figure 4 shows how the algorithms compare on six concepts of varying size.[5] All the training sets had 10% noise. As before, C4.5's results are for learning on 100k examples, while VFDT and VFDT-boot's are for 20 million. Both versions of VFDT do better than C4.5 on every concept size considered. However, contrary to what we would expect, as concept size increases the relative benefit seems to remain approximately constant for VFDT and VFDT-boot. Looking deeper, we find that with 20 million examples VFDT and VFDT-boot induce trees with approximately 9k nodes regardless of the size of the underlying concept. This suggests that they would take good advantage of even more training examples.

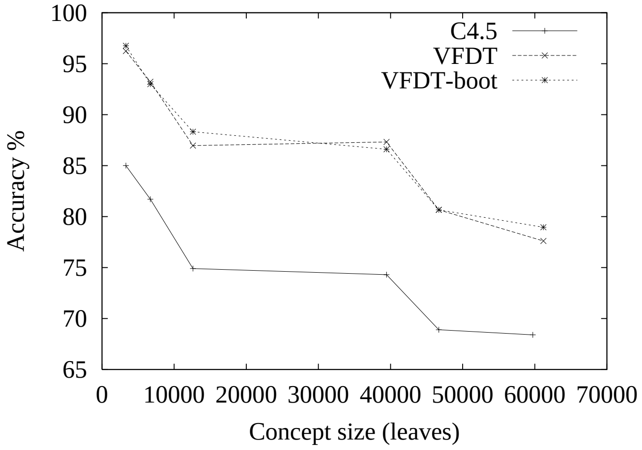

Figure 4: Accuracy as a function of the complexity of the true concept.

We carried out all runs without ever writing VFDT's training examples to disk (i.e., generating them on the fly and passing them directly to VFDT). For time comparison purposes, however, we measured the time it takes VFDT to read examples from the (0.25, 0.10, 25209, 12605) data set from disk on the Pentium III/500 MHz machine. VFDT takes 5752 seconds to read the 20 million examples, and 625 seconds to process them. In other words, learning time is about an order of magnitude less than input time. On the same runs, C4.5 takes 36 seconds to read and process 100k examples, and VFDT takes 47 seconds.

Finally, we generated 160 million examples from the (0.25, 0.10, 25209, 12605) concept. Figure 5 compares VFDT and C4.5 on this data set. VFDT makes progress over the entire data set, but begins to asymptote after 10 million examples; the final 150 million examples contribute 0.58% to accuracy. VFDT took 9501 seconds to process the examples (excluding I/O) and induced 21.9k leaves. In the near future we plan to carry out similar runs with more complex concepts and billions of examples.

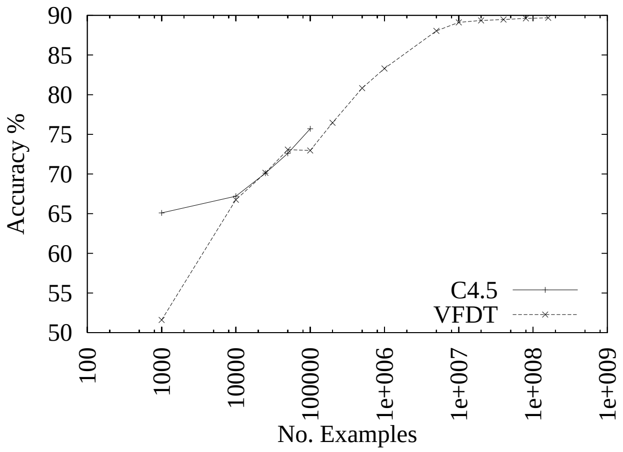

Figure 5: VFDT trained on 160 million examples.

4.2 Lesion studies

We conducted a series of lesion studies to evaluate the effectiveness of some of the components and parameters of the VFDT system. Figure 6 shows the accuracy of the learners on the (0.25, 0.00, 25209, 12605) data set. It also shows a slight modification to the VFDT-boot algorithm, where the tree produced by C4.5 is used without first over-pruning it. All versions of VFDT were run with $\delta = 10^{-7}$, $\tau = 5\%$, $n_{min} = 200$, no leaf reactivation, and no rescans. C4.5 does better without noise than with it, but VFDT is still able to use additional data to significantly improve accuracy. VFDT-boot with the "no over-prune" setting is initially better than the over-pruning version, but does not make much progress and is eventually overtaken. We hypothesize that this is because it has difficulty overcoming the poor low-confidence decisions C4.5 made near its leaves.

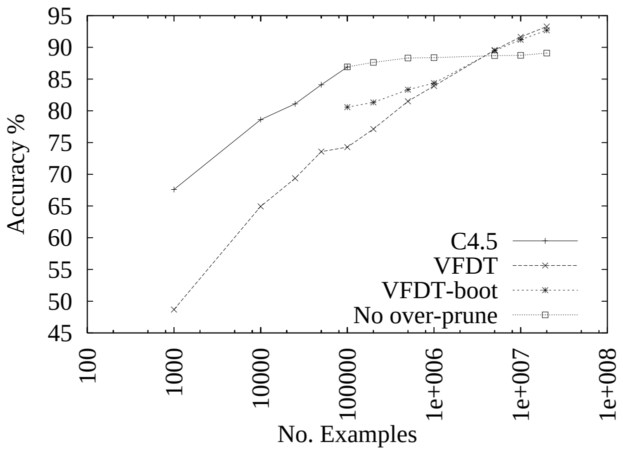

Figure 6: Effect of initializing VFDT with C4.5 with and without over-pruning.

In the remainder of the lesion studies VFDT was run on the (0.25, 0.10, 25209, 12605) data set with $\delta = 10^{-7}$, $\tau = 5\%$, $n_{min} = 200$, no leaf reactivation, and no rescans. We evaluated the effect of disabling ties, so that VFDT does not make any splits until it is able to identify a clear winner.

[5]The concept (0.15, 0.10, 74449, 37225) turned out to be atypically easy, and is not included in the graph to avoid obscuring the trend. The observed accuracies for this concept were: C4.5 – 83.1%; VFDT – 89.0%; VFDT-boot – 89.7%.

We conducted two runs, holding all parameters constant except that the second run never split with a tie. Without ties VFDT induced a tree with only 65 nodes and 72.9% accuracy, compared to 8k nodes and 86.9% accuracy with ties. VFDT-boot without ties produced 805 nodes and 83.3% accuracy, compared to 8k nodes and 88.5% accuracy with ties. We also carried out two runs holding all parameters constant except n_{min}, the number of new examples that must be seen at a node before G's are recomputed. The first run recomputed G every 200 examples ($n_{min} = 200$), and the second did it for every example ($n_{min} = 1$). Doing the G computations for every example, VFDT gained 1.1% accuracy and took 3.8 times longer to run. VFDT-boot lost 0.9% accuracy and took 3.7 times longer. Both learners induced about 5% more nodes with the more frequent G computations. We then carried out two runs holding all parameters but VFDT's memory limit constant. The first run was allowed 40 MB of memory; the second was allowed 80 MB. VFDT and VFDT-boot both induced 7.8k more nodes with the additional memory, which improved VFDT's accuracy by 3.0% and VFDT-boot's by 3.2%. Finally, we carried out two runs holding all parameters but δ constant. The first run had a delta of 10^{-2}, and the second had a delta of 10^{-7}. With the lower δ, VFDT and VFDT-boot both induced about 30% fewer nodes than with the higher one. VFDT's accuracy was 2.3% higher and VFDT-boot's accuracy was 1.0% higher with the lower δ.

4.3 Web data

We are currently applying VFDT to mining the stream of Web page requests emanating from the whole University of Washington main campus. The nature of the data is described in detail in [23]. In our experiments so far we have used a one-week anonymized trace of all the external web accesses made from the university campus. There were 23,000 active clients during this one-week trace period, and the entire university population is estimated at 50,000 people (students, faculty and staff). The trace contains 82.8 million requests, which arrive at a peak rate of 17,400 per minute. The size of the compressed trace file is about 20 GB.[6] Each request is tagged with an anonymized organization ID that associates the request with one of the 170 organizations (colleges, departments, etc.) within the university. One purpose this data can be used for is to improve Web caching. The key to this is predicting as accurately as possible which hosts and pages will be requested in the near future, given recent requests. We applied decision-tree learning to this problem in the following manner. We split the campus-wide request log into a series of equal time slices $T_0, T_1, \ldots, T_t, \ldots$; in the experiments we report, each time slice is an hour. For each organization $O_1, O_2, \ldots, O_i, \ldots, O_{170}$ and each of the 244k hosts appearing in the logs $H_1, \ldots, H_j, \ldots, H_{244k}$, we maintain a count of how many times the organization accessed the host in the time slice, C_{ijt}. We discretize these counts into four buckets, representing "no requests," "1 – 12 requests," "13 – 25 requests" and "26 or more requests." Then for each time slice and host accessed in that time slice (T_t, H_j) we generate an example with attributes $t \bmod 24, C_{1,jt}, \ldots, C_{ijt}, \ldots C_{170,jt}$ and class 1 if H_j is requested in time slice T_{t+1} and 0 if it is not. This can be carried out in real time using modest resources by keeping statistics on the last and current time slices C_{t-1} and C_t in memory, only keeping counts for hosts that actually appear in a time slice (we never needed more than 30k counts), and outputting the examples for C_{t-1} as soon as C_t is complete. Using this procedure we obtained a dataset containing 1.89 million examples, 61.1% of which were labeled with the most common class (that the host did not appear again in the next time slice).

[6]This log is from May 1999. Traffic in May 2000 was double this size; a one-week log was approximately 50 GB compressed.

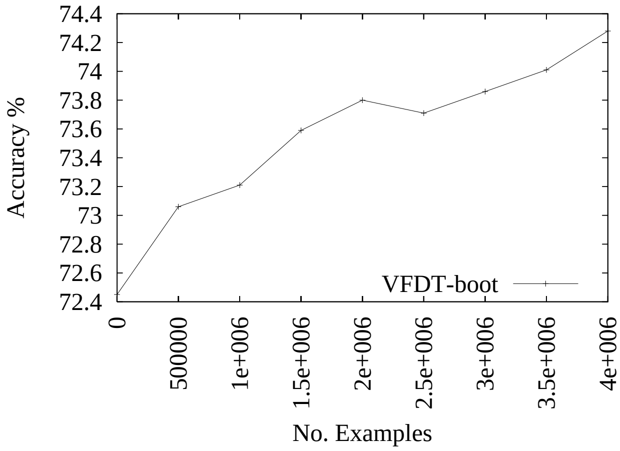

Figure 7: Performance on Web data.

Testing was carried out on the examples from the last day (276,230 examples). VFDT was run with $\delta = 10^{-7}$, $\tau = 5\%$, and $n_{min} = 200$. All runs were carried out on a 400 MHz Pentium machine. A decision stump (a decision tree with only one node) obtains 64.2% accuracy on this data. The decision stump took 1277 seconds to learn, and VFDT took 1450 seconds to do one pass over the training data (after being initialized with C4.5's over-pruned tree). The majority of this time (983 seconds) was spent reading data from disk. The bootstrap run of C4.5 took 2975 seconds to learn on a subsample of 74.5k examples (as many as would fit in 40 MB of RAM) and achieved 73.3% accuracy. Thus VFDT learned faster on 1.61 million examples than C4.5 did on 75k. We also used a machine with 1 GB of RAM to run C4.5 on the entire 1.61 million training examples; the run took 24 hours and the resulting tree was 75% accurate. Figure 7 shows VFDT-boot's performance on this dataset, using 1 GB of RAM. We extended VFDT's run out to 4 million examples by rescanning. The x axis shows the number of examples presented to VFDT after the C4.5 bootstrap phase was complete. Accuracy improves steadily as more examples are seen. VFDT is able to achieve accuracy similar to C4.5's in a small fraction of the time. Further, C4.5's memory requirements and batch nature will not allow it to scale to traces much larger than a week, while VFDT can easily incorporate data indefinitely. The next step is to apply VFDT to predicting page requests from a given host. We also plan to address issues related to time-changing behavior and then set VFDT running permanently, learning and relearning as dictated by the data stream.

5. RELATED WORK

Previous work on mining large databases using subsampling methods includes the following. Catlett [2] proposed several heuristic methods for extending RAM-based batch decision-tree learners to datasets with up to hundreds of thousands of examples. Musick, Catlett and Russell [13] proposed and tested (but did not implement in a learner) a theoretical model for choosing the size of subsamples to use in comparing attributes. Maron and Moore [9] used Hoeffding bounds to speed selection of instance-based regression models via cross-validation (see also [12]). Gratch's Sequential ID3 [6] used a statistical method to minimize the number of examples needed to choose each split in a decision tree. (Sequential ID3's guarantees of similarity to the batch tree were much looser than those derived here for Hoeffding trees, and it was only tested on repeatedly sampled small datasets.) Gehrke et al.'s BOAT [5] learned an approximate tree using a fixed-size subsample, and then refined it by scanning the full database. Provost et al. [14] studied different strategies for mining larger and larger subsamples until accuracy (apparently) asymptotes. In contrast to systems that learn in main memory by subsampling, systems like SLIQ [10] and SPRINT [17] use all the data, and concentrate on optimizing access to disk by always reading examples (more precisely, attribute lists) sequentially. VFDT combines the best of both worlds, accessing data sequentially and using subsampling to potentially require much less than one scan, as opposed to many. This allows it to scale to larger databases than either method alone. VFDT has the additional advantages of being incremental and anytime: new examples can be quickly incorporated as they arrive, and a usable model is available after the first few examples and then progressively refined.

As mentioned previously, there is a large literature on incremental learning, which space limitations preclude reviewing here. The system most closely related to ours is Utgoff's [20] ID5R (extended in [21]). ID5R learns the same tree as ID3 (a batch method), by restructuring subtrees as needed. While its learning time is linear in the number of examples, it is worst-case exponential in the number of attributes. On the simple, noise-free problems it was tested on, it was much slower than ID3; noise would presumably aggravate this. Thus ID5R does not appear viable for learning from high-speed data streams.

A number of efficient incremental or single-pass algorithms for KDD tasks other than supervised learning have appeared in recent years (e.g., clustering [4] and association rule mining [19]). A substantial theoretical literature on online algorithms exists (e.g., [8]), but it focuses on weak learners (e.g., linear separators), because little can be proved about strong ones like decision trees.

6. FUTURE WORK

We plan to shortly compare VFDT with SPRINT/SLIQ. VFDT may outperform these even in fully disk-resident datasets, because it can learn in less than one scan while the latter require multiple scans, and the dominant component of their cost is often the time required to read examples from disk multiple times. VFDT's speed and anytime character make it ideal for interactive data mining; we plan to also study its application in this context (see [18]). Other directions for future work include: further developing the application of VFDT to Web log data; studying other applications of VFDT (e.g., intrusion detection); using non-discretized numeric attributes in VFDT; studying the use of post-pruning in VFDT; further optimizing VFDT's computations (e.g., by recomputing G's exactly when we can tell that the current example may cause the Hoeffding bound to be reached); using adaptive δ's; studying the use of an example cache in main memory to speed induction by reusing examples at multiple levels; comparing VFDT to ID5R and other incremental algorithms; adapting VFDT to learn evolving concepts in time-changing domains; adapting VFDT to learning with imbalanced classes and asymmetric misclassification costs; adapting VFDT to the extreme case where even the final decision tree (without any stored sufficient statistics) does not fit in main memory; parallelizing VFDT; applying the ideas described here to other types of learning (e.g., rule induction, clustering); etc.

7. CONCLUSION

This paper introduced Hoeffding trees, a method for learning online from the high-volume data streams that are increasingly common. Hoeffding trees allow learning in very small constant time per example, and have strong guarantees of high asymptotic similarity to the corresponding batch trees. VFDT is a high-performance data mining system based on Hoeffding trees. Empirical studies show its effectiveness in taking advantage of massive numbers of examples. VFDT's application to a high-speed stream of Web log data is under way.

Acknowledgments

This research was partly funded by an NSF CAREER award to the first author.

8. REFERENCES

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[2] J. Catlett. *Megainduction: Machine Learning on Very Large Databases*. PhD thesis, Basser Department of Computer Science, University of Sydney, Sydney, Australia, 1991.

[3] T. G. Dietterich. Overfitting and undercomputing in machine learning. *Computing Surveys*, 27:326–327, 1995.

[4] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the Twenty-Fourth International Conference on Very Large Data Bases*, pages 323–333, New York, NY, 1998. Morgan Kaufmann.

[5] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-L. Loh. BOAT: optimistic decision tree construction. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 169–180, Philadelphia, PA, 1999. ACM Press.

[6] J. Gratch. Sequential inductive learning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 779–786, Portland, OR, 1996. AAAI Press.

[7] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[8] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1997.

[9] O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, San Mateo, CA, 1994.

[10] M. Mehta, A. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proceedings of the Fifth International Conference on Extending Database Technology*, pages 18–32, Avignon, France, 1996. Springer.

[11] R. G. Miller, Jr. *Simultaneous Statistical Inference*. Springer, New York, NY, 2nd edition, 1981.

[12] A. W. Moore and M. S. Lee. Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 190–198, New Brunswick, NJ, 1994. Morgan Kaufmann.

[13] R. Musick, J. Catlett, and S. Russell. Decision theoretic subsampling for induction on large databases. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 212–219, Amherst, MA, 1993. Morgan Kaufmann.

[14] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 23–32, San Diego, CA, 1999. ACM Press.

[15] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[16] J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1019–1024, Montréal, Canada, 1995. Morgan Kaufmann.

[17] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the Twenty-Second International Conference on Very Large Databases*, pages 544–555, Mumbai, India, 1996. Morgan Kaufmann.

[18] P. Smyth and D. Wolpert. Anytime exploratory data analysis for massive data sets. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 54–60, Newport Beach, CA, 1997. AAAI Press.

[19] H. Toivonen. Sampling large databases for association rules. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pages 134–145, Mumbai, India, 1996. Morgan Kaufmann.

[20] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[21] P. E. Utgoff. An improved algorithm for incremental induction of decision trees. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 318–325, New Brunswick, NJ, 1994. Morgan Kaufmann.

[22] G. I. Webb. OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3:431–465, 1995.

[23] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-based analysis of Web-object sharing and caching. In *Proceedings of the Second USENIX Conference on Internet Technologies and Systems*, pages 25–36, Boulder, CO, 1999.