Mining Time-Changing Data Streams

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

Laurie Spencer
Innovation Next
1107 NE 45th St. #427
Seattle, WA 98105, U.S.A
lauries@innovation-next.com

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

ABSTRACT

Most statistical and machine-learning algorithms assume that the data is a random sample drawn from a stationary distribution. Unfortunately, most of the large databases available for mining today violate this assumption. They were gathered over months or years, and the underlying processes generating them changed during this time, sometimes radically. Although a number of algorithms have been proposed for learning time-changing concepts, they generally do not scale well to very large databases. In this paper we propose an efficient algorithm for mining decision trees from continuously-changing data streams, based on the ultra-fast VFDT decision tree learner. This algorithm, called CVFDT, stays current while making the most of old data by growing an alternative subtree whenever an old one becomes questionable, and replacing the old with the new when the new becomes more accurate. CVFDT learns a model which is similar in accuracy to the one that would be learned by reapplying VFDT to a moving window of examples every time a new example arrives, but with $O(1)$ complexity per example, as opposed to $O(w)$, where w is the size of the window. Experiments on a set of large time-changing data streams demonstrate the utility of this approach.

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, data streams, subsampling, concept drift

1. INTRODUCTION

Modern organizations produce data at unprecedented rates; among large retailers, e-commerce sites, telecommunications providers, and scientific projects, rates of gigabytes per day are common. While this data can contain valuable knowledge, its volume increasingly outpaces practitioners' ability to mine it. As a result, it is now common practice either to mine a subsample of the available data or to mine for models drastically simpler than the data could support. In some cases, the volume and time span of accumulated data is such that just storing it consistently and reliably for future use is a challenge. Further, even when storage is not problematic, it is often difficult to gather the data in one place, at one time, in a format appropriate for mining. For all these reasons, in many areas the notion of mining a fixed-sized database is giving way to the notion of mining an open-ended data stream as it arrives. The goal of our research is to help make this possible with a minimum of effort for the data mining practitioner. In a previous paper [9] we presented VFDT, a decision tree induction system capable of learning from high-speed data streams in an incremental, anytime fashion, while producing models that are asymptotically arbitrarily close to those that would be learned by traditional decision tree induction systems.

Most statistical and machine-learning algorithms, including VFDT, make the assumption that training data is a random sample drawn from a stationary distribution. Unfortunately, most of the large databases and data streams available for mining today violate this assumption. They exist over months or years, and the underlying processes generating them changes during this time, sometimes radically. For example, a new product or promotion, a hacker's attack, a holiday, changing weather conditions, changing economic conditions, or a poorly calibrated sensor could all lead to violations of this assumption. For classification systems, which attempt to learn a discrete function given examples of its inputs and outputs, this problem takes the form of changes in the target function over time, and is known as concept drift. Traditional systems assume that all data was generated by a single concept. In many cases, however, it is more accurate to assume that data was generated by a series of concepts, or by a concept function with time-varying parameters. Traditional systems learn incorrect models when they erroneously assume that the underlying concept is stationary if in fact it is drifting.

One common approach to learning from time-changing data is to repeatedly apply a traditional learner to a sliding window of w examples; as new examples arrive they are inserted into the beginning of the window, a corresponding number of examples is removed from the end of the window, and the learner is reapplied [27]. As long as w is small relative to the rate of concept drift, this procedure assures availability of a model reflecting the current concept generating the data. If the window is too small, however, this may result in insufficient examples to satisfactorily learn the concept. Further, the computational cost of reapplying a learner may be prohibitively high, especially if examples arrive at a rapid rate and the concept changes quickly.

To meet these challenges we propose the CVFDT system, which is capable of learning decision trees from high-speed, time changing data streams. CVFDT works by efficiently keeping a decision tree up-to-date with a window of examples. In particular, it is able to keep its model consistent with a window using only a constant amount of time for each new example (more precisely, time proportional to the number of attributes in the data and the depth of the induced tree). CVFDT grows an alternate subtree whenever an old one seems to be out-of-date, and replaces the old one when the new one becomes more accurate. This allows it to make smooth, fine-grained adjustments when concept drift occurs. In effect, CVFDT is able to learn a nearly equivalent model to the one VFDT would learn if repeatedly reapplied to a window of examples, but in $O(1)$ time instead of $O(w)$ time per new example.

In the next section we discuss the basics of the VFDT system, and in the following section we introduce the CVFDT system. We then present a series of experiments on synthetic data which demonstrate how CVFDT can outperform traditional systems on high-speed, time-changing data streams. Next, we apply CVFDT to mining the stream of web page requests for the entire University of Washington campus. We conclude with a discussion of related and future work.

2. THE VFDT SYSTEM

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 $y = f(\mathbf{x})$ which 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 [4, 20]. 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 C4.5 [20], CART, SLIQ [17], and SPRINT [24]

Table 1: The VFDT Algorithm.

Inputs:	S	is a stream of examples,
	$\mathbf{X}$	is a set of symbolic attributes,
	$G(.)$	is a split evaluation function,
	δ	is one minus the desired probability of choosing the correct attribute at any given node,
	τ	is a user-supplied tie threshold,
	n_{min}	is the # examples between checks for growth.
Output:	HT	is a decision tree.

Procedure VFDT $(S, \mathbf{X}, G, \delta, \tau)$
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 $(\mathbf{x}, y)$ 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 $n_{ijy}(l)$.
  Label l with the majority class among the examples seen so far at l.
  Let n_l be the number of examples seen at l.
  If the examples seen so far at l are not all of the same class and n_l mod n_{min} is 0, 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.
    Let $\Delta\overline{G}_l = \overline{G}_l(X_a) - \overline{G}_l(X_b)$.
    If $((\Delta\overline{G}_l > \epsilon)$ or $(\Delta\overline{G}_l <= \epsilon < \tau))$ 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.

use every available training example to select the best attribute for each split. This policy is necessary when data is scarce, but it has two problems when training examples are abundant: it requires all examples be available for consideration throughout their entire runs, which is problematic when data does not fit in RAM or on disk, and it assumes that the process generating examples remains the same during the entire period over which the examples are collected and mined.

In previous work [9] we presented the VFDT (Very Fast Decision Tree learner) system, which is able to learn from abundant data within practical time and memory constraints. It accomplishes this by noting, with Catlett [5] and others [12, 19], that it may be sufficient to use a small sample of the available examples when choosing the split attribute at any given node. Thus, only the first examples to arrive on the data stream need to be used to choose the split attribute at the root; subsequent ones are passed through the induced portion of the tree until they reach a leaf, are used to choose a split attribute there, and so on recursively. To determine the number of examples needed for each decision, VFDT uses a statistical result known as *Hoeffding bounds* or *additive Chernoff bounds* [13]. After n independent observations of a real-valued random variable r with range R, the Hoeffding bound ensures that, with confidence $1-\delta$, the true mean of r is at least $\overline{r}-\epsilon$, where $\overline{r}$ is the observed mean of the samples and

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

This is true irrespective of the probability distribution that generated the observations. Let $G(X_i)$ be the heuristic measure used to choose test attributes (we use information gain). After seeing n samples at a leaf, let X_a be the attribute with the best heuristic measure and X_b be the attribute with the second best. Let $\Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b)$ be a new random variable, the difference between the observed heuristic values. Applying the Hoeffding bound to $\Delta\overline{G}$, we see that if $\Delta\overline{G} > \epsilon$ (as calculated by Equation 1 with a user-supplied δ), we can confidently say that the difference between $\overline{G}(X_a)$ and $\overline{G}(X_b)$ is larger than zero, and select X_a as the split attribute.[1,2] Table 1 contains pseudo-code for VFDT's core algorithm. The counts n_{ijk} are the sufficient statistics needed to compute most heuristic measures; if other quantities are required, they can be similarly maintained. When the sufficient statistics fill the available memory, VFDT reduces its memory requirements by temporarily deactivating learning in the least promising nodes; these nodes can be reactivated later if they begin to look more promising than currently active nodes. VFDT employs a tie mechanism which precludes it from spending inordinate time deciding between attributes whose practical difference is negligible. That is, VFDT declares a tie and selects X_a as the split attribute any time $\Delta\overline{G} < \epsilon < \tau$ (where τ is a user-supplied tie threshold). 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. Notice that the tests for splits and ties are only executed once for every n_{min} (a user supplied value) examples that arrive at a leaf. This is justified by the observation that VFDT is unlikely to make a decision after any given example, so it is wasteful to carry out these calculations for each one of them. The pseudo-code shown is only for symbolic attributes; we are currently developing its extension to numeric ones. 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.

Using off-the-shelf hardware, VFDT is able to learn as fast as data can be read from disk. The time to incorporate an example is $O(ldvc)$ where l is the maximum depth of HT, d is the number of attributes, v is the maximum number of values per attribute, and c is the number of classes. This time is independent of the total number of examples already seen (assuming the size of the tree depends only on the "true" concept, and not on the dataset). Because of the use of Hoeffding bounds, these speed gains do not necessarily lead to a loss of accuracy. It can be shown that, with high confidence, the core VFDT system (without ties or deactivations due to memory constraints) will asymptotically induce a tree arbitrarily close to the tree induced by a traditional batch learner. Let DT_∞ be the tree induced by a version of VFDT using infinite data to choose each node's split attribute, HT_δ be the tree learned by the core VFDT system given an infinite data stream, and p be the probability that an example passed through DT_∞ to level i will fall into a leaf at that point. Then the probability that an arbitrary example will take a different path through DT_∞ and HT_δ is bounded by δ/p [9]. A corollary of this result states that the tree learned by the core VFDT system on a finite sequence of examples will correspond to a subtree of DT_∞ with the same bound of δ/p. See Domingos and Hulten [9] for more details on VFDT and this δ/p bound.

3. THE CVFDT SYSTEM

CVFDT (Concept-adapting Very Fast Decision Tree learner) is an extension to VFDT which maintains VFDT's speed and accuracy advantages but adds the ability to detect and respond to changes in the example-generating process. Like other systems with this capability, CVFDT works by keeping its model consistent with a sliding window of examples. However, it does not need to learn a new model from scratch every time a new example arrives; instead, it updates the sufficient statistics at its nodes by incrementing the counts corresponding to the new example, and decrementing the counts corresponding to the oldest example in the window (which now needs to be forgotten). This will statistically have no effect if the underlying concept is stationary. If the concept is changing, however, some splits that previously passed the Hoeffding test will no longer do so, because an alternative attribute now has higher gain (or the two are too close to tell). In this case CVFDT begins to grow an alternative subtree with the new best attribute at

[1]This is valid as long as $\overline{G}$ (and therefore $\Delta\overline{G}$) can be viewed as an average over all examples seen at the leaf, which is the case for most commonly-used heuristics. For example, if information gain is used, the quantity being averaged is the reduction in the uncertainty regarding the class membership of the example.

[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 [18].

Table 2: The CVFDT algorithm.

Inputs:	S	is a sequence of examples,
	$\mathbf{X}$	is a set of symbolic attributes,
	$G(.)$	is a split evaluation function,
	δ	is one minus the desired probability of choosing the correct attribute at any given node,
	τ	is a user-supplied tie threshold,
	w	is the size of the window,
	n_{min}	is the # examples between checks for growth,
	f	is the # examples between checks for drift.
Output:	HT	is a decision tree.

Procedure CVFDT($S, \mathbf{X}, G, \delta, \tau, w, n_{min}$)
/* Initialize */
Let HT be a tree with a single leaf l_1 (the root).
Let $ALT(l_1)$ be an initially empty set of alternate
  trees for l_1.
Let $\overline{G}_1(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most
  frequent class in S.
Let $\mathbf{X_1} = \mathbf{X} \cup \{X_\emptyset\}$.
Let W be the window of examples, initially empty.
For each class y_k
  For each value x_{ij} of each attribute $X_i \in \mathbf{X}$
    Let $n_{ijk}(l_1) = 0$.
/* Process the examples */
For each example $(\mathbf{x}, y)$ in S
  Sort $(\mathbf{x}, y)$ into a set of leaves L using HT and all
    trees in ALT of any node $(\mathbf{x}, y)$ passes through.
  Let ID be the maximum id of the leaves in L.
  Add $((\mathbf{x}, y), ID)$ to the beginning of W.
  If $|W| > w$
    Let $((\mathbf{x}_w, y_w), ID_w)$ be the last element of W
    ForgetExamples($HT, n, (\mathbf{x}_w, y_w), ID_w$)
    Let $W = W$ with $((\mathbf{x}_w, y_w), ID_w)$ removed
  CVFDTGrow($HT, n, G, (\mathbf{x}, y), \delta, n_{min}, \tau$)
  If there have been f examples since the last checking
    of alternate trees
    CheckSplitValidity(HT, n, δ)
Return HT.

Table 3: The CVFDTGrow procedure.

Procedure CVFDTGrow($HT, n, G, (\mathbf{x}, y), \delta, n_{min}, \tau$)
Sort $(\mathbf{x}, y)$ into a leaf l using HT.
Let P be the set of nodes traversed in the sort.
For each node l_{pi} in P
  For each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_{l_p}$
    Increment $n_{ijy}(l_p)$.
  For each tree T_a in $ALT(l_p)$
    CVFDTGrow($T_a, n, G, (\mathbf{x}, y), \delta, n_{min}, \tau$)
Label l with the majority class among the examples seen
  so far at l.
Let n_l be the number of examples seen at l.
If the examples seen so far at l are not all of the same
  class and n_l mod n_{min} is 0, 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 and δ.
  Let $\Delta\overline{G}_l = \overline{G}_l(X_a) - \overline{G}_l(X_b)$
  If $((\Delta\overline{G}_l > \epsilon)$ or $(\Delta\overline{G}_l <= \epsilon < \tau))$ 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 $ALT(l_m) = \{\}$.
      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 $n_{ijk}(l_m) = 0$.

its root. When this alternate subtree becomes more accurate on new data than the old one, the old subtree is replaced by the new one.

Table 2 contains a pseudo-code outline of the CVFDT algorithm. CVFDT does some initializations, and then processes examples from the stream S indefinitely. As each example $(\mathbf{x}, y)$ arrives, it is added to the window[3], an old example is forgotten if needed, and $(\mathbf{x}, y)$ is incorporated into the current model. CVFDT periodically scans HT and all alternate trees looking for internal nodes whose sufficient statistics indicate that some new attribute would make a better test than the chosen split attribute. An alternate subtree is started at each such node.

Table 3 contains pseudo-code for the tree-growing portion of the CVFDT system. It is similar to the Hoeffding Tree algorithm, but CVFDT monitors the validity of its old decisions by maintaining sufficient statistics at every node in HT (instead of only at the leaves like VFDT). Forgetting an old example is slightly complicated by the fact that HT may have grown or changed since the example was initially incorporated. Therefore, nodes are assigned a unique, monotonically increasing ID as they are created. When an example is added to W, the maximum ID of the leaves it reaches in HT and all alternate trees is recorded with it. An example's effects are forgotten by decrementing the counts in the sufficient statistics of every node the example reaches

[3]The window is stored in RAM if resources are available, otherwise it will be kept on disk.

Table 4: The ForgetExample procedure.

Procedure ForgetExample$(HT, n, (\mathbf{x}_w, y_w), ID_w)$
Sort $(\mathbf{x_w}, y_w)$ through HT while it traverses leaves with id $\leq ID_w$,
Let P be the set of nodes traversed in the sort.
For each node l in P
For each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
Decrement $n_{ijk}(l)$.
For each tree T_{alt} in $ALT(l)$
ForgetExample$(T_{alt}, n, (\mathbf{x}_w, y_w), ID_w)$

in HT whose ID is $\leq$ the stored ID. See the pseudo-code in Table 4 for more detail about how CVFDT forgets examples.

CVFDT periodically scans the internal nodes of HT looking for ones where the chosen split attribute would no longer be selected; that is, where $\overline{G}(X_a) - \overline{G}(X_b) \leq \epsilon$ and $\epsilon > \tau$. When it finds such a node, CVFDT knows that it either initially made a mistake splitting on X_a (which should happen less than δ% of the time), or that something about the process generating examples has changed. In either case, CVFDT will need to take action to correct HT. CVFDT grows alternate subtrees to changed subtrees of HT, and only modifies HT when the alternate is more accurate than the original. To see why this is needed, let $l_\triangle$ be a node where change was detected. A simple solution is to replace $l_\triangle$ with a leaf predicting the most common class in $l_\triangle$'s sufficient statistics. This policy assures that HT is always as current as possible with respect to the process generating examples. However, it may be too drastic, because it initially forces a single leaf to do the job previously done by a whole subtree. Even if the subtree is outdated, it may still be better than the best single leaf. This is particularly true when $l_\triangle$ is at or near the root of HT, as it will result in drastic short-term reductions in HT's predictive accuracy – clearly not acceptable when a parallel process is using HT to make critical decisions.

Each internal node in HT has a list of alternate subtrees being considered as replacements for the subtree rooted at the node. Table 5 contains pseudo-code for the *CheckSplitValidity* procedure. *CheckSplitValidity* starts an alternate subtree whenever it finds a new winning attribute at a node; that is, when there is a new best attribute and $\Delta\overline{G} > \epsilon$ or if $\epsilon < \tau$ and $\Delta\overline{G} \geq \tau/2$. This is very similar to the procedure used to choose initial splits, except the tie criteria is tighter to avoid excessive alternate tree creation. CVFDT supports a parameter which limits the total number of alternate trees being grown at any one time. Alternate trees are grown the same way HT is, via recursive calls to the CVFDT procedures. Periodically, each node with a non-empty set of alternate subtrees, l_{test}, enters a testing mode to determine if it should be replaced by one of its alternate subtrees. Once in this mode, l_{test} collects the next m training examples that arrive at it and, instead of using them to grow its children or alternate trees, uses them to compare the accuracy of the subtree it roots with the accuracies of all of its alternate subtrees. If the most accurate alternate subtree is more accurate than the l_{test}, l_{test} is replaced by the alternate. During the test phase, CVFDT also prunes alternate subtrees that are not making progress (i.e., whose accuracy is not increasing over time). For each alternate subtree of l_{test}, l^i_{alt}, CVFDT remembers the smallest accuracy difference ever achieved between the two, $\Delta_{min}(l_{test}, l^i_{alt})$. CVFDT prunes any alternate whose current test phase accuracy difference is at least $\Delta_{min}(l_{test}, l^i_{alt}) + 1\%$.[4]

Table 5: The CheckSplitValidity procedure.

Procedure CheckSplitValidity(HT, n, δ)
For each node l in HT that is not a leaf
For each tree T_{alt} in $ALT(l)$
CheckSplitValidity(T_{alt}, n)
Let X_a be the split attribute at l.
Let X_n be the attribute with the highest $\overline{G}_l$ other than X_a.
Let X_b be the attribute with the highest $\overline{G}_l$ other than X_n.
Let $\Delta\overline{G}_l = \overline{G}_l(X_n) - \overline{G}_l(X_b)$
If $\Delta\overline{G}_l \geq 0$ and no tree in $ALT(l)$ already splits on X_n at its root
Compute ϵ using Equation 1 and δ.
If $(\Delta\overline{G}_l > \epsilon)$ or $(\epsilon < \tau$ and $\Delta\overline{G}_l \geq \tau/2)$, then
Let l_{new} be an internal node that splits on X_n.
Let $ALT(l) = ALT(l) + \{l_{new}\}$
For each branch of the split
Add a new leaf l_m to l_{new}
Let $\mathbf{X_m} = \mathbf{X} - \{X_n\}$.
Let $ALT(l_m) = \{\}$.
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 $n_{ijk}(l_m) = 0$.

One window size w will not be appropriate for every concept and every type of drift; it may be beneficial to dynamically change w during a run. For example, it may make sense to shrink w when many of the nodes in HT become questionable at once, or in response to a rapid change in data rate, as these events could indicate a sudden concept change. Similarly, some applications may benefit from an increase in w when there are few questionable nodes because this may indicate that the concept is stable – a good time to learn a more detailed model. CVFDT is able to dynamically adjust the size of its window in response to user-supplied events. Events are specified in the form of hook functions which monitor S and HT and can call the *SetWindowSize* function when appropriate. CVFDT changes the window size by updating w and immediately forgetting any examples that no longer fit in W.

We now discuss a few of the properties of the CVFDT system and briefly compare it with VFDT-Window, a learner that reapplies VFDT to W for every new example. CVFDT requires memory proportional to $O(ndvc)$ where n is the number of nodes in CVFDT's main tree and all alternate trees, d is the number of attributes, v is the maximum number of values per attribute, and c is the number of classes. The window of examples can be in RAM or can be stored on

[4] When RAM is short, CVFDT is more aggressive about pruning unpromising alternate subtrees.

disk at the cost of a few disk accesses per example. Therefore, CVFDT's memory requirements are dominated by the sufficient statistics and are independent of the total number of examples seen. At any point during a run, CVFDT will have available a model which reflects the current concept generating W. It is able to keep this model up-to-date in time proportional to $O(l_c dvc)$ per example, where l_c is the length of the longest path an example will have to take through HT times the number of alternate trees. VFDT-Window requires $O(l_v dvcw)$ time to keep its model up-to-date for every new example, where l_v is the maximum depth of HT. VFDT is a factor of wl_v/l_c worse than CVFDT; empirically, we observed l_c to be smaller than l_v in all of our experiments. Despite this large time difference, CVFDT's drift mechanisms allow it to produce a model of similar accuracy. The structure of the models induced by the two may, however, be significantly different, for the following reason. VFDT-Window uses the information from each training example at one place in the tree it induces: the leaf where the example falls when it arrives. This means that VFDT-Window uses the first examples from W to make a decision at its root, the next to make a decision at the first level of the tree, and so on. After an initial building phase, CVFDT will have a fully induced tree available. Every new example is passed through this induced tree, and the information it contains is used to update statistics at every node it passes through. This difference can be an advantage for CVFDT, as it allows the induction of larger trees with better probability estimates at the leaves. It can also be a disadvantage and VFDT-Window may be more accurate when there is a large concept shift part-way through W. This is because VFDT-Window's leaf probabilities will be set by examples near the end of W while CVFDT's will reflect all of W. Also notice that, even when the structure of the induced tree does not change, CVFDT and VFDT-Window can outperform VFDT simply because their leaf probabilities (and therefore class predictions) are updated faster, without the "dead weight" of all the examples that fell into leaves before the current window.

4. EMPIRICAL STUDY

We conducted a series of experiments comparing CVFDT to VFDT and VFDT-Window. Our goals were to evaluate CVFDT's ability to scale up, to evaluate CVFDT's ability to deal with varying levels of drift, and to identify and characterize the situations where CVFDT outperforms the other systems.

4.1 Synthetic Data

The experiments with synthetic data used a changing concept based on a rotating hyperplane. A hyperplane in d-dimensional space is the set of points $\mathbf{x}$ that satisfy

$$\sum_{i=1}^{d} w_i x_i = w_0 \qquad (2)$$

where x_i is the ith coordinate of $\mathbf{x}$. Examples for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled positive, and examples for which $\sum_{i=1}^{d} w_i x_i < w_0$ are labeled negative. Hyperplanes are useful for simulating time-changing concepts because we can change the orientation and position of the hyperplane in a smooth manner by changing the relative size of the weights. In particular, sorting the weights by their magnitudes provides a good indication of which dimensions contain the most information; in the limit, when all but one of the weights are zero, the dimension associated with the non-zero weight is the only one that contains any information about the concept. This allows us to control the relative information content of the attributes, and thus change the optimal order of tests in a decision tree representing the hyperplane, by simply changing the relative sizes of the weights. We sought a concept that maintained the advantages of a hyperplane, but where the weights could be randomly modified without potentially causing the decision frontier to move outside the range of the data. To meet these goals we used a series of alternating class bands separated by parallel hyperplanes. We start with a reference hyperplane whose weights are initialized to .2 except for w_0 which is $.25d$. To label an example, we substitute its coordinates into the left hand side of Equation 2 to obtain a sum s. If $|s| \leq .1 * w_0$ the example is labeled positive, otherwise if $|s| \leq .2 * w_0$ the example is labeled negative, and so on. Examples were generated uniformly in a d-dimensional unit hypercube (with the value of each x_i ranging from [0, 1]). They were then labeled using the concept, and their continuous attributes were uniformly discretized into five bins. Noise was added by randomly switching the class labels of $p\%$ of the examples. Unless otherwise stated, each experiment used the following settings: five million training examples; $\delta = 0.0001$; $f = 20,000$; $n_{min} = 300$; $\tau = 0.05$; $w = 100,000$; CVFDT's window on disk; no memory limits; no pre-pruning; a test set of 50,000 examples; and $p = 5\%$. CVFDT put leaves into alternate tree test mode after 9,000 examples and used test samples of 1,000 examples. All runs were done on a 1GHz Pentium III machine with 512 MB of RAM, running Linux.

The first series of experiments compares the ability of CVFDT and VFDT to deal with large concept-drifting datasets. Concept drift was added to the datasets in the following manner. Every 50,000 examples w_1 was modified by adding $0.01d\sigma$ to it, and the test set was relabeled with the updated concept (with $p\%$ noise as before). σ was initially 1 and was multiplied by -1 at 5% of the drift points and also just before w_1 fell below 0 or rose above $.25d$. Figure 1 compares the accuracy of the algorithms as a function of d, the dimensionality of the space. The reported values are obtained by testing the accuracy of the learned models every 10,000 examples throughout the run and averaging these results. Drift level, reported on the minor axis, is the average percentage of the test set that changes label at each point the concept changes. CVFDT is substantially more accurate than VFDT, by approximately 10% on average, and CVFDT's performance improves slightly with increasing d. Figure 2 compares the average size of the models induced during the run shown in Figure 1 (the reported values are generated by averaging after every 10,000 examples, as before). CVFDT's trees are substantially smaller than VFDT's, and the advantage is consistent across all the values of d we tried. This simultaneous accuracy and size advantage derives from the fact that CVFDT's tree is built on the 100,000 most relevant examples, while VFDT's is built on millions of outdated examples.

We next carried out a more detailed evaluation of CVFDT's concept drift mechanism. Figure 3 shows a detailed view of one of the runs from Figures 1 and 2, the one for $d = 50$. The minor axis shows the portion of the test

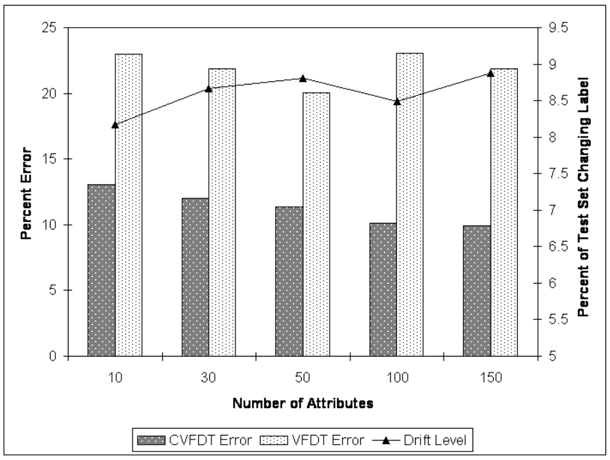

Figure 1: Error rates as a function of the number of attributes.

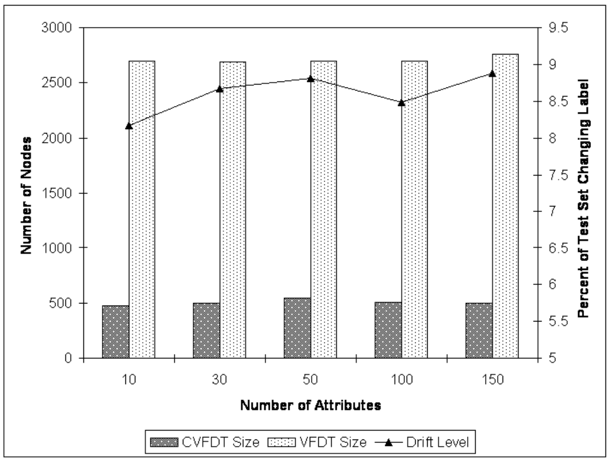

Figure 2: Tree sizes as a function of the number of attributes.

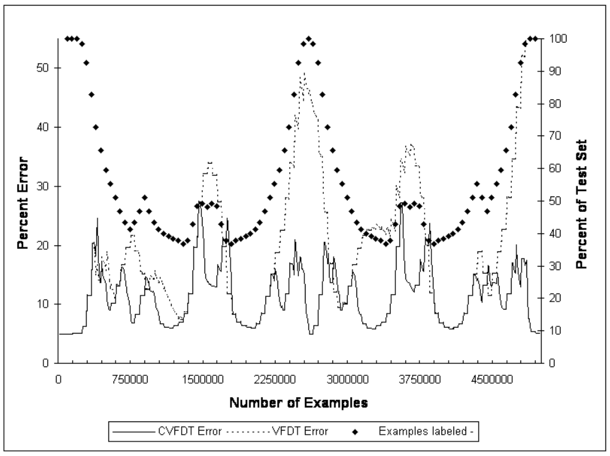

Figure 3: Error rates of learners as a function of the number of examples seen.

set that is labeled negative at each test point (computed before noise is added to the test set) and is included to illustrate the concept drift present in the dataset. CVFDT is able to quickly respond to drift, while VFDT's error rate often rises drastically before reacting to the change. Further, VFDT's error rate seems to peak at worse values as the run goes on, while CVFDT's error peaks seem to have constant height. We believe this happens because VFDT has more trouble responding to drift when it has induced a larger tree and must replicate corrections across more outdated structure. CVFDT does not face this problem because it replaces subtrees when they become outdated. We gathered some detailed statistics about this run. CVFDT took 4.3 times longer than VFDT (5.7 times longer if including time to do the disk I/O needed to keep the window on disk). VFDT's average memory allocation over the course of the run was 23 MB while CVFDT's was 16.5 MB. The average number of nodes in VFDT's tree was 2696 and the average number in CVFDT's tree was 677, of which 132 were in alternate trees and the remainder were in the main tree.

Next we examined how CVFDT responds to changing levels of concept drift on five datasets with $d = 50$. Drift was added using a parameter D. Every 75,000 examples, D of the concept hyperplane's weights were selected at random and updated as before, $w_i = w_i + 0.01 d\sigma_i$ (although σ_i now has a 25% chance of flipping signs, chosen to prevent too many weights from drifting in the same pattern). Figure 4 shows the comparison on these datasets. CVFDT substantially outperformed VFDT at every level of drift. Notice that VFDT's error rate approaches 50% for $D > 2$, and that the variance in VFDT's data points is large. CVFDT's error rate seems to grow smoothly with increasing levels of concept change, suggesting that its drift adaptations are robust and effective.

We wanted to gain some insight into the way CVFDT starts new alternate subtrees, prunes existing ones, and replaces portions of HT with alternates. For this purpose, we instrumented a run of CVFDT on the $D = 2$ dataset from Figure 4 to output a token in response to each of these events. We aggregated the events in chunks of 100,000 training examples, and generated data points for all non-zero values. Figure 5 shows the results of this experiment. There are a large number of events during the run. For example, 109 alternate subtrees were swapped into HT. Most of the swaps seem to occur when the examples in the test set are changing labels quickly.

We also wanted to see how well CVFDT would compare to a system using traditional drift-tracking methods. We thus compared CVFDT, VFDT, and VFDT-Window. We simulated VFDT-Window by running VFDT on W for every 100,000 examples instead of for every example. The dataset for the experiment had $d = 50$ and used the same drift settings used to generate Figure 4 with $D = 1$. Figure 6 shows the results. CVFDT's error rate was the same as VFDT-Window's, except for a brief period during the middle of the run when class labels were changing most rapidly. CVFDT's average error rate for the run was 16.3%, VFDT's was 19.4%, and VFDT-Window's was 15.3%. The difference in runtimes was very large. VFDT took about 10 minutes, CVFDT took about 46 minutes, and we estimate that VFDT-Window would have taken 548 days to do its complete run if applied to every new example. Put another way, VFDT-Window provides a 4% accuracy gain compared

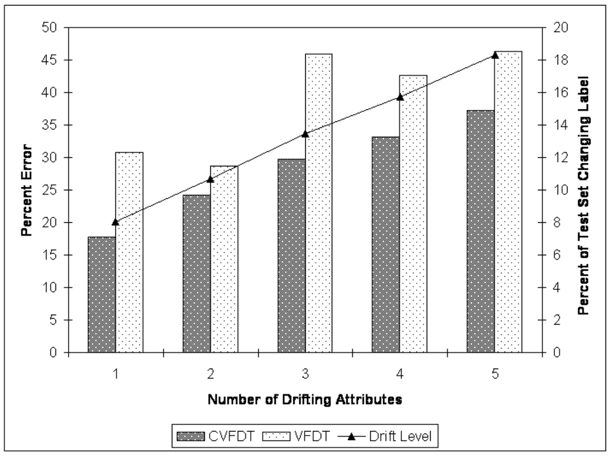

Figure 4: Error rates as a function of the amount of concept drift.

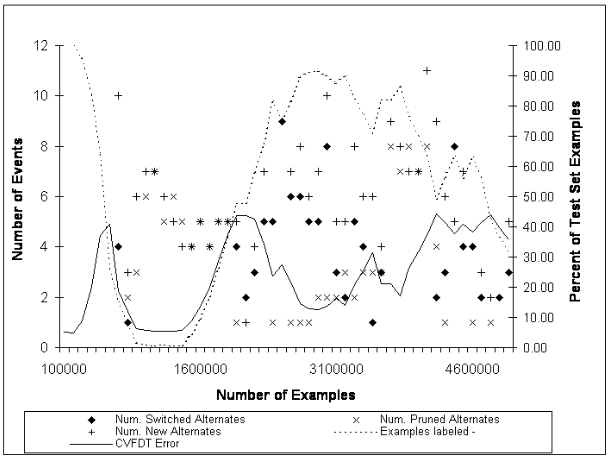

Figure 5: CVFDT's drift characteristics.

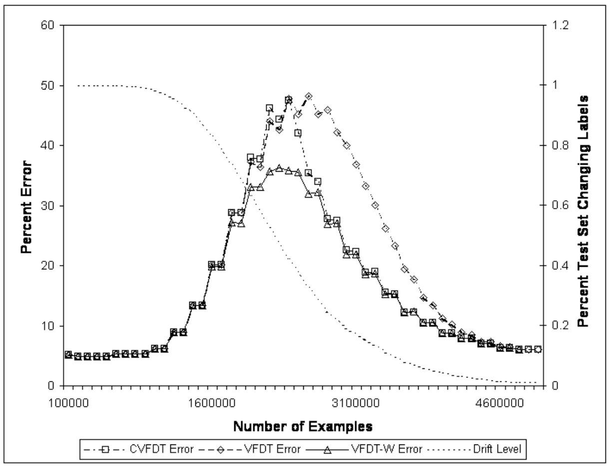

Figure 6: Error rates over time of CVFDT, VFDT, and VFDT-Window.

to VFDT, at a cost of increasing the running time by a factor of 17,000. CVFDT provides 75% of VFDT-Window's accuracy gain, and introduces a time penalty of less than 0.1% of VFDT-Window's.

CVFDT's alternate trees and additional sufficient statistics do not use too much RAM. For example, none of CVFDT's $d = 50$ runs ever grew to more than 70MB. We never observed CVFDT to use more RAM than VFDT; in fact it often used as little as half the RAM of VFDT. The systems' RAM requirements are dominated by the sufficient statistics which are kept at the leaves in VFDT, and at every node in CVFDT. We observed that VFDT often had twice as many leaves as there were nodes in CVFDT's tree and all alternate trees combined. This is what we expected: VFDT considers many more examples and is forced to grow larger trees to make up for the fact that its early decisions become incorrect due to concept drift. CVFDT's alternate tree pruning mechanism seems to be effective at trading memory for smooth transitions between concepts. Further, there is room for more aggressive pruning if CVFDT exhausts available RAM. Exploring this tradeoff is an area for future work.

4.2 Web Data

We are currently applying CVFDT 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 Wolman et al. [29]. 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.[5] 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 maintained a count of how many times the organization accessed the host in the time slice, C_{ijt}. We discretized 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 generated an example with attributes $C_{1,jt}, \ldots, C_{ijt}, \ldots C_{170,jt}$ 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, 60.9% of which were la-

[5]This log is from May 1999. Traffic in May 2000 was more than double this size.

beled with the most common class (that the host did not appear again in the next time slice).

Our exploration was designed to determine if CVFDT's concept drift features would provide any benefit to this application. As each example arrived, we tested the accuracy of the learners' models on it, and then allowed the learners to update their models with the example. We kept statistics about how the aggregated accuracies changed over time. VFDT and CVFDT were both run with $\delta = 0.0001$, $\tau = 5\%$, and $n_{min} = 300$. CVFDT's additional parameters were $w = 100,000$ and $f = 20,000$. VFDT achieved 72.7% accuracy over the whole dataset and CVFDT achieved 72.3%. However, CVFDT's aggregated accuracy was higher for the first 70% of the run, at times by as much as 1.0%. CVFDT's accuracy fell behind only near the end of the run, for (we believe) the following reason. Its drift tracking kept it ahead throughout the first part of the run, but its window was too small for it to learn as detailed a model of the data as VFDT did by the end. This experiment shows that the data does indeed contain concept drift, and that CVFDT's ability to respond to the drift gives it an advantage over VFDT. The next step is to run CVFDT with different, perhaps dynamic, window sizes to further evaluate the nature of the drift. We also plan to evaluate CVFDT over traces longer than a week.

5. RELATED WORK

Schlimmer and Granger's [23] STAGGER system was one of the first to explicitly address the problem of concept drift. Salganicoff [21] studied drift in the context of nearest-neighbor learning. Widmer and Kubat's [27] FLORA system used a window of examples, but also stored old concept descriptions and reactivated them if they seemed to be appropriate again. All of these systems were only applied to small databases (by today's standards). Kelly, Hand, and Adams [14] addressed the issue of drifting parameters in probability distributions. Theoretical work on concept drift includes [16] and [3].

Ganti, Gehrke, and Ramakrishnan's [11] DEMON framework is designed to help adapt incremental learning algorithms to work effectively with time-changing data streams. DEMON differs from CVFDT by assuming data arrives periodically, perhaps daily, in large blocks, while CVFDT deals with each example as it arrives. The framework uses off-line processing time to mine interesting subsets of the available data blocks.

In earlier work [12] Gehrke, Ganti, and Ramakrishnan presented an incremental decision tree induction algorithm, BOAT, which works in the DEMON framework. BOAT is able to incrementally maintain a decision tree equivalent to the one that would be learned by a batch decision tree induction system. When the underlying concept is stable, BOAT can perform this maintenance extremely quickly. When drift is present, BOAT must discard and regrow portions of its induced tree. This can be very expensive when the drift is large or affects nodes near the root of the tree. CVFDT avoids the problem by using alternate trees and removing the restriction that it learn exactly the tree that a batch system would. A comparison between BOAT and CVFDT is an area for future work.

There has been a great deal of work on incrementally maintaining association rules. Cheung, Han, Ng, and Wong [7] and Fazil, Tansel, and Arkun [2] propose algorithms for maintaining sets of association rules when new transactions are added to the database. Sarda and Srinivas [22] have also done some work in the area. DEMON's contribution [11] is particularly relevant, as it addresses association rule maintenance specifically in the high-speed data stream domain where blocks of transactions are added and deleted from the database on a regular basis.

Aspects of the concept drift problem are also addressed in the areas of activity monitoring [10], active data mining [1] and deviation detection [6]. The main goal here is to explicitly detect changes, rather than simply maintain an up-to-date concept, but techniques for the latter can obviously help in the former.

Several pieces of research on concept drift and context-sensitive learning are collected in a special issue of the journal *Machine Learning* [28]. Other relevant research appeared in the ICML-96 Workshop on Learning in Context-Sensitive Domains [15], the AAAI-98 Workshop on AI Approaches to Time-Series Problems [8], and the NIPS-2000 Workshop on Real-Time Modeling for Complex Learning Tasks [26]. Turney [25] maintains an online bibliography on context-sensitive learning.

6. FUTURE WORK

We plan to apply CVFDT to more real-world problems; its ability to adjust to concept changes should allow it to perform very well on a broad range of tasks. CVFDT may be a useful tool for identifying anomalous situations. Currently CVFDT discards subtrees that are out-of-date, but some concepts change periodically and these subtrees may become useful again – identifying these situations and taking advantage of them is another area for further study. Other areas for study include: comparisons with related systems; continuous attributes; weighting examples; partially forgetting examples by allowing their weights to decay; simulating weights by subsampling; and controlling the weight decay function according to external information about drift.

7. CONCLUSION

This paper introduced CVFDT, a decision-tree induction system capable of learning accurate models from the most demanding high-speed, concept-drifting data streams. CVFDT is able to maintain a decision-tree up-to-date with a window of examples by using a small, constant amount of time for each new example that arrives. The resulting accuracy is similar to what would be obtained by reapplying a conventional learner to the entire window every time a new example arrives. Empirical studies show that CVFDT is effectively able to keep its model up-to-date with a massive data stream even in the face of large and frequent concept shifts. A preliminary application of CVFDT to a real world domain shows promising results.

8. ACKNOWLEDGMENTS

This research was partly supported by a gift from the Ford Motor Company, and by NSF CAREER and IBM Faculty awards to the third author.

9. REFERENCES

[1] R. Agrawal and G. Psaila. Active data mining. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 3–8, Montréal, Canada, 1995. AAAI Press.

[2] N. F. Ayan, A. U. Tansel, and M. E. Arkun. An efficient algorithm to update large itemsets with early pruning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 287–291, San Diego, CA, 1999. ACM Press.

[3] P. L. Bartlett, S. Ben-David, and S. R. Kulkarni. Learning changing concepts by exploiting the structure of change. *Machine Learning*, 41:153–174, 2000.

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

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

[6] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. In *Proceedings of the Twenty-Fourth International Conference on Very Large Data Bases*, pages 606–617, New York, NY, 1998. Morgan Kaufmann.

[7] D. W.-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 106–114, New Orleans, Louisiana, 1996. IEEE Computer Society Press.

[8] A. Danyluk, T. Fawcett, and F. Provost, editors. *Proceedings of the AAAI-98 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis.* AAAI Press, Madison, WI, 1998.

[9] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, 2000. ACM Press.

[10] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 53–62, San Diego, CA, 1999. ACM Press.

[11] V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and monitoring evolving data. In *Proceedings of the Sixteenth International Conference on Data Engineering*, pages 439–448, San Diego, CA, 2000.

[12] 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.

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

[14] M. G. Kelly, D. J. Hand, and N. M. Adams. The impact of changing populations on classifier performance. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 367–371, San Diego, CA, 1999. ACM Press.

[15] M. Kubat and G. Widmer, editors. *Proceedings of the ICML-96 Workshop on Learning in Context-Sensitive Domains.* Bari, Italy, 1996.

[16] P. M. Long. The complexity of learning according to two models of a drifting environment. *Machine Learning*, 37:337–354, 1999.

[17] 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.

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

[19] 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.

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

[21] M. Salganicoff. Density-adaptive learning and forgetting. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 276–283, Amherst, MA, 1993. Morgan Kaufmann.

[22] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 240–245, Vienna, Austria, 1998. IEEE.

[23] J. C. Schlimmer and R. H. Granger, Jr. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 502–507, Philadelphia, PA, 1986. Morgan Kaufmann.

[24] 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, Bombay, India, 1996. Morgan Kaufmann.

[25] P. Turney. Context-sensitive learning bibliography. Online bibliography, Institute for Information Technology of the National Research Council of Canada, Ottawa, Canada, 1998. http://ai.iit.nrc.ca/-bibliographies/context-sensitive.html.

[26] S. Vijayakumar and S. Schaal, editors. *Proceedings of the NIPS-2000 Workshop on Real-Time Modeling for Complex Learning Tasks.* NIPS Foundation, Breckenridge, Colorado, 2000.

[27] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

[28] G. Widmer and M. Kubat. Special issue on context sensitivity and concept drift. *Machine Learning*, 32(2), 1998.

[29] 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.