Learning Decision Trees Adaptively from Data Streams with Time Drift

Albert Bifet and Ricard Gavaldà

2007
Learning Decision Trees Adaptively from Data Streams with Time Drift

Albert Bifet        Ricard Gavalda
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
{abifet,gavalda}@lsi.upc.edu

Abstract

We propose a new method for mining concept-drifting data streams using decision trees and adaptive windowing. We present a new algorithm based on Hulten-Spencer-Domingos’s CVFDT that overcomes some of the shortcomings of CVFDT, specifically, dependence on user-entered parameters that determine the guessed speed of change. Our algorithm detects when change occurs and provably adapts to the speed of change without user intervention. It is based on ADWIN, an adaptive algorithm for detecting change and maintaining an updated sample from the input sequence automatically. Our experiments show that the new algorithm does never worse, and in some cases much better, than CVFDT.

1. Introduction

Data streams pose several challenges on data mining algorithm design. Limited use of resources (time and memory) is one. The necessity of dealing with data whose nature or distribution changes over time is another fundamental one. Dealing with time-changing data requires in turn strategies for detecting and quantifying change, forgetting stale examples, and for model revision. Fairly generic strategies exist for detecting change and deciding when examples are no longer relevant. Model revision strategies, on the other hand, are in most cases method-specific.

Decision trees are among the most common and well-studied classifier models. Classical methods such as C4.5 are not apt for data streams, as they assume all training data are available simultaneously in main memory, allowing for an unbounded number of passes, and certainly do not deal with data that changes over time. In the data stream context, reference works on learning decision trees are still those by Domingos’ group in the early 2000’s: the Very Fast Decision Tree method (VFDT) for fast, incremental learning [7], and later the Concept-adapting VFDT method (CVFDT) to deal with time changing data [11]. Later, Gama et al. presented UFFT, which extended these ideas to numerical attributes and provided a different, more adaptive, strategy for dealing with dynamic data [8, 9]. Section 2 will review these approaches in detail.

We present a new algorithm ADWIN-DT, based on Hulten-Spencer-Domingos’s CVFDT that overcomes some of its shortcomings, specifically, the dependence on user-entered parameters telling how often the model should be revised. ADWIN-DT detects the dynamicity of the change as it is occurring in the data and adapts its behavior automatically. In other words, the rate at which it forgets obsolete data and revises the current model depends on how fast data is actually changing, rather than an a priori guess by the user.

ADWIN-DT is based on the use of ADWIN, an algorithm that we proposed recently [4] to detect change and maintain an updated sample from an input sequence adaptively. In fact, it is conceptually simple to combine CVFDT with ADWIN – one the points made in [4] is that ADWIN helps the design of data mining algorithms because it encapsulates, as a black box, all the computation and statistics necessary to detect change and maintain a sample of the currently relevant data. Additionally, ADWIN has rigorous guarantees of performance (a theorem). We show that these guarantees can be transferred to our decision tree learner ADWIN-DT in the following way: in a period of data stability after a change, the classification error of ADWIN-DT will decrease as it reads more data at the same rate as that of VFDT (which assumes no error change), after a transient whose length depends solely on the magnitude of change and not on any user-guessed parameter. In Section 3 we recall the relevant facts about ADWIN from [4], and in Section 4 how to use it to produce ADWIN-DT, an adaptive version of CVFDT.

Because of this self-adaptive property, we can present datasets where our algorithm performs much better than CVFDT, and we never do much worse. We test our method with synthetic datasets, using the SEA concepts, introduced in [14] and a rotating hyperplane as described in [11]. With the hyperplane dataset we perform two types of experi-
mments, with abrupt and gradual concept drift. Also, we test ADWIN-DT with a real dataset, the adult dataset [1] from the UCI repository of machine learning databases. This dataset has no drift, and ADWIN-DT shows its robustness to false positives by not detecting any. We simulate concept drift by ordering the UCI Adult dataset by one of its attributes. These experiments are described in Section 6.

We conclude the paper by comparing the running time and memory usage of our algorithm with that of CVFDT, and making some concluding remarks and describing future work.

2. CVFDT and UFFT

Decision trees are classifier algorithms [5, 13]. Each internal node of a tree \( DT \) 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 \( y = DT(x) \) for an example \( x \) 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. Extended models where the nodes contain more complex tests and leaves contain more complex classification rules are also possible.

A decision tree is learned top-down 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 such as ID3, C4.5 [13], and CART [5] 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. In particular, they are not applicable to data streams, where potentially there is no bound on number of examples and these arrive sequentially.

Domingos and Hulten [7] developed VFDT (Very Fast Decision Tree learner), an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. VFDT trees exploit the idea that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the goodness of an attribute).

The pseudo-code of VFDT is shown in Figure 1. The counts \( n_{ijk} \) are the sufficient statistics needed to compute most heuristic measures, including information gain. The formula on the right-hand side of line 4 is given by the Hoeffding bound (and we do not justify it here). 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 \( DT \) to make class predictions.

As an extension of VFDT to deal with concept change Hulten, Spencer, and Domingos presented Concept-adapting Very Fast Decision Trees CVFDT [11] algorithm. Its pseudocode is shown in Figure 2.

CVFDT works by keeping its model consistent with respect to a sliding window of data from the data stream, and creating and replacing alternate decision subtrees when it detects that the distribution of data is changing at a node. When new data arrives, CVFDT updates the sufficient statistics at its nodes by incrementing the counts \( n_{ijk} \) corresponding to the new examples and decrementing the counts \( n_{ijk} \) corresponding to the oldest example in the window, which is effectively forgotten.

CVFDT uses a number of parameters, with default values that can be changed by the user - but which are fixed for a given execution:

1. \( W \): is the example window size.
2. \( T_0 \): after each \( T_0 \) examples, CVFDT traverses all the decision tree, and checks at each node if the splitting attribute is still the best. If there is a better splitting attribute, it starts growing an alternate tree rooted at this node, and it splits on the currently best attribute according to the statistics in the node.
3. \( T_1 \): after an alternate tree is created, the following \( T_1 \) examples are used to build the alternate tree.
4. \( T_2 \): after the arrival of \( T_1 \) examples, the following \( T_2 \) examples are used to test the accuracy of the alternate tree.

Figure 1. The VFDT algorithm

VFDT\((Stream, \delta)\)
1. \(\triangleright\) Let \( HT \) be a tree with a single leaf (root)
2. \(\triangleright\) Init counts \( n_{ijk} \) at root
3. \(\triangleright\) for each example \((x, y)\) in Stream
4. \(\triangleright\) do VFDTGROW\(((x, y), HT, \delta)\)

VFDTGROW\(((x, y), HT, \delta)\)
1. \(\triangleright\) Sort \((x, y)\) to leaf \(l\) using \( HT \)
2. \(\triangleright\) Update counts \( n_{ijk} \) at leaf \(l\)
3. \(\triangleright\) Compute \( G \) for each attribute
4. if \( G(\text{Best Attr.)} - G(2\text{nd best}) > \sqrt{\frac{R^2 \ln 1/\delta}{2n}} \)
5. then
6. \(\triangleright\) Split leaf \(l\) on best attribute
7. \(\triangleright\) for each branch
8. \(\triangleright\) do \(\triangleright\) Initialize new leaf counts at \(l\)
tree. If the alternate tree is more accurate than the current one, CVDFDT replaces it with this alternate tree (we say that the alternate tree is promoted).

The default values are $W = 50,000$, $T_0 = 10,000$, $T_1 = 9,000$, and $T_2 = 1,000$. One can interpret these figures as the preconception that often the last 50,000 examples are likely to be relevant, and that change is not likely to occur faster than every 10,000 examples. Of course, these preconceptions may be quite wrong for some data sources, and other values could be much more appropriate. Even more, there may be no single “right” set of values for a given data source, because it may experience frequent changes during some periods, and long stable epochs at other times. More in general, an approach based on fixed parameters will be caught in the following tradeoff: the user would like to use large parameters to have more accurate statistics (hence, more precision) during periods of stability, but at the same time use small parameters to be able to quickly react to changes, when they occur.

Partly to overcome this problem, Gama, Medas and Rocha [9] presented the Ultra Fast Forest of Trees (UFFT) algorithm, an algorithm for supervised classification learning, that generates a forest of binary trees.

UFFT is designed for numerical data. It uses analytical techniques to choose the splitting criteria, and the information gain to estimate the merit of each possible splitting-test. For multi-class problems, the algorithm builds a binary tree for each possible pair of classes leading to a forest-of-trees.

The UFFT algorithm maintains, at each node of all decision trees, a Naïve Bayes classifier. Those classifiers were constructed using the sufficient statistics needed to evaluate the splitting criteria when that node was a leaf. After the leaf becomes a node, all examples that traverse the node will be classified by the Naïve Bayes. The basic idea of the drift detection method is to control this error rate. If the distribution of the examples is stationary, the error rate of Naïve Bayes decreases or stabilizes. If there is a change on the distribution of the examples the Naïve Bayes error increases.

The system uses DDM, the drift detection method proposed by Gama et al. [8] that controls the number of errors produced by the learning model during prediction. It compares the statistics of two windows: the first one contains all the data, and the second one contains only the data from the beginning until the number of errors increases. Their method doesn’t store these windows in memory. It keeps only statistics and a window of recent errors. Details on the statistical test used to detect change among these windows can be found in [8].

DDM has a good behaviour detecting abrupt changes and gradual changes when the gradual change is not very slow, but it has difficulties when the change is slowly gradual. In that case, the examples will be stored for long time, the drift

```plaintext
CVFDT(\textit{Stream}, \delta)
1 \triangleright \text{Let } HT \text{ be a tree with a single leaf(root)}
2 \triangleright \text{Init counts } n_{ijk} \text{ at root}
3 \textbf{for} each example \((x, y)\) in Stream
4 \hspace{1em} \textbf{do} Add, Remove and Forget Examples
5 \hspace{1em} CVFDTGROW\((x, y), HT, \delta\)
6 \hspace{1em} \textbf{if} new \(T_0\) examples arrived
7 \hspace{1em} \hspace{1em} \textbf{then}
8 \hspace{1em} \hspace{1em} \textbf{CHECKSPLITVALIDITY}(HT, n, \delta)
```

```plaintext
CVFDTGROW\((x, y), HT, \delta\)
1 \triangleright \text{Sort } (x, y) \text{ to leaf } l \text{ using } HT
2 \triangleright \text{Update counts } n_{ijk} \text{ at leaf } l
3 \hspace{1em} \text{and nodes traversed in the sort}
4 \hspace{1em} \textbf{do} for each tree \(T_{alt}\) in ALT(l)
5 \hspace{1em} \hspace{1em} CVFDTGROW\((x, y), T_{alt}, \delta\)
6 \hspace{1em} \hspace{1em} \textbf{if} \(G(\text{Best Attr}) - G(2\text{nd best}) > \sqrt{\frac{R^2 \ln 1/\delta}{2n}}\)
7 \hspace{1em} \hspace{1em} \hspace{1em} \textbf{then}
8 \hspace{1em} \hspace{1em} \hspace{1em} \textbf{CHECKSPLITVALIDITY}(HT, n, \delta)
9 \hspace{1em} \textbf{if} Mode = \text{Build and new } T_1 \text{ examples arrived}
10 \hspace{1em} \hspace{1em} \textbf{then}
11 \hspace{1em} \hspace{1em} \hspace{1em} \textbf{Mode} \leftarrow \text{Test}
12 \hspace{1em} \textbf{if} Mode = \text{Test and new } T_2 \text{ examples arrived}
13 \hspace{1em} \hspace{1em} \hspace{1em} \textbf{then}
14 \hspace{1em} \hspace{1em} \hspace{1em} \textbf{Switch alternate tree if it is more accurate}
```

```plaintext
CHECKSPLITVALIDITY(HT, n, \delta)
1 \textbf{for} each node \(l\) in \(HT\) that it is not a leaf
2 \hspace{1em} \textbf{do} for each tree \(T_{alt}\) in ALT(l)
3 \hspace{1em} \hspace{1em} \textbf{CHECKSPLITVALIDITY}(T_{alt}, n, \delta)
4 \hspace{1em} \textbf{if} exists a new promising attributes at node \(l\)
5 \hspace{1em} \hspace{1em} \textbf{do} \textbf{Start an alternate subtree}
6 \hspace{1em} \hspace{1em} \textbf{do} \textbf{Mode} \leftarrow \text{Build}
```

Figure 2. The CVFDT algorithm
level can take too much time to trigger and the examples memory can be exceeded.

When UFFT detects an statistically significant increase of the Naive-Bayes error in a given node, an indication of a change in the distribution of the examples, this suggest that the splitting-test that has been installed at this node is no longer appropriate. The subtree rooted at that node is pruned, and the node becomes a leaf. All the sufficient statistics of the leaf are initialized.

To conclude, let us remark that [7] shows theoretical guarantees on the error rate of VFDT: under some technical conditions, one can show that the tree learned by VFDT on a finite sample is very similar to a subtree of the one that would be produced by using an infinite sample. By contrast, no similar guarantees are shown for CVFDT in [11], and they seem hard to obtain. As mentioned, we can give performance guarantees for our algorithm, in the sense that in stable periods our tree will quickly tend to that produced by VFDT. We believe that using our proof techniques it should be possible to prove similar (but worse) guarantees for UFFT, although they are not shown in the paper [9].

3. The ADWIN Algorithm

In this section we review ADWIN, an algorithm for detecting change and dynamically adjusting the length of a data window. For details see [4].

The inputs to ADWIN are a confidence value δ ∈ (0, 1) and a (possibly infinite) sequence of real values x1, x2, x3, . . . . The value of xi is available only at time t. Each xi is generated according to some distribution Di, independently for every t. We denote with µi the expected value of xi when it is drawn according to Di. We assume that xi is always in [0, 1]; by an easy rescaling, we can handle any case in which we know an interval [a, b] such that a ≤ xi ≤ b with probability 1. Nothing else is known about the sequence of distributions Di; in particular, µi is unknown for all t.

Algorithm ADWIN uses a sliding window W with the most recently read xj. Let µW denote the (known) average of the elements in W, and µW the (unknown) average of µt for t ∈ W. We use |W| to denote the length of a (sub)window W.

Algorithm ADWIN is presented in Figure 3. The idea is simple: whenever two “large enough” subwindows of W exhibit “distinct enough” averages, one can conclude that the corresponding expected values are different, and the older portion of the window is dropped. In other words, W is kept as long as possible while the null hypothesis “µt has remained constant in W” is sustainable up to confidence δ.”Large enough” and “distinct enough” above are made precise by choosing an appropriate statistical test for distribution change, which in general involves the value of δ, the

**Algorithm ADWIN:**

1. Initialize Window W
2. for each t > 0
3. do W ← W ∪ {xj} (i.e., add xj to the head of W)
4. repeat Drop elements from the tail of W
5. until |μW0 − μW| ≥ εcut holds
6. for every split of W into W = W0 · W1
7. output μW

**Figure 3. Algorithm ADWIN.**

The meaning of “large enough” and “distinct enough” can be made precise again by using the Hoeffding bound. The test eventually boils down to whether the average of the two subwindows is larger than a value εcut computed as follows

\[ m := \frac{2}{1/|W_0| + 1/|W_1|} \]

\[ ε_{cut} := \sqrt{\frac{1}{2m} \cdot \ln \frac{4|W|}{δ}} \]

where m is the harmonic mean of |W0| and |W1|. In [4] we describe a more sensitive test based on the normal distribution which, although not 100% rigorous, is perfectly valid in practice; we omit its description here.

The main technical result in [4] about the performance of ADWIN is the following theorem, that provides bounds on the rate of false positives and false negatives for ADWIN:

**Theorem** With εcut defined as above, at every time step we have:

1. (False positive rate bound). If µt has remained constant within W, the probability that ADWIN shrinks the window at this step is at most δ.
2. (False negative rate bound). Suppose that for some partition of W in two parts W0W1 (where W1 contains the most recent items) we have |µW0 − µW| > 2εcut. Then with probability 1 − δ ADWIN shrinks W to W1, or shorter.

This theorem justifies us in using ADWIN in two ways:

- as a change detector, since ADWIN shrinks its window if and only if there has been a significant change in recent times (with high probability)
• as an estimator for the current average of the sequence it is reading since, with high probability, older parts of the window with a significantly different average are automatically dropped.

We will see three variants of our decision tree builder based on using ADWIN in the first way, the second way, or both. The algorithm as in Figure 3 is computationally expensive, because it checks exhaustively all “large enough” subwindows of the current window for possible cuts. Furthermore, the contents of the window is kept explicitly, with the corresponding memory cost as the window grows. To reduce these costs we presented in [4] an efficient version, ADWIN2 that uses ideas developed in data stream algorithms [2, 12, 3, 6] to find a good cutpoint quickly. We summarize the behavior of this policy in the following theorem.

**Theorem 2** The ADWIN2 algorithm maintains a data structure with the following properties:

- It maintains the same theoretical bounds on false positives and false negatives as in Theorem 1, except for constant factors that can be made arbitrary small.
- It uses memory $O(\log W)$.
- The arrival of a new element can be processed in $O(1)$ amortized time and $O(\log W)$ worst-case time.

That is, ADWIN2 uses logarithmic memory and time per item instead of linear memory and time per item as ADWIN does. In the sequel, whenever we say ADWIN we really mean its efficient implementation, ADWIN2.

In [4] we perform two types of experiments. In the first type, we test the ability of ADWIN to track some unknown quantity, independent of any learning. We generate a sequence of random bits with some hidden probability $p$ that changes over time. We check the rate of false positives ($p$ of claimed changes when $p$ does not really change) and false negatives ($p$ of changes missed when $p$ does change) and in this case the time until the change is declared. We compare ADWIN with a number of fixed-size windows and show, as expected, that it performs about as well or only slightly worse than the best window for each rate of change, and performs far better than any window of any fixed size $W$ when the change of rate is very different from $W$. We also compare it to DDM [8], method described in Section 2, and show that it performs better, for moderately large quantities of data.

Also, we test ADWIN in conjunction with the Naïve Bayes (NB) learning algorithm. We try both using ADWIN “outside”, monitoring NB’s error rate, and “inside”, providing accurate statistics to NB. We compare them to fixed-size windows and the DDM variable-length window strategy in [8]. The second combination (ADWIN inside NB) performs best, sometimes spectacularly so. The first combination performs about as well as [8] in some cases, and substantially better in others.

### 4. Combining CVFDT and ADWIN

We propose a new method for managing alternate trees. Rather than checking at fixed intervals, whether to build alternate trees, or promote them, we run a number of ADWIN instances that continuously check whether change is occurring.

More precisely, we combine CVFDT and ADWIN in three different ways:

- Using one instance of ADWIN in each node, as a change detector, to monitor the classification error rate at that node. As in the work of Gama et al. [9], a significant increase in that rate indicates that the data is changing w.r.t. the time at which the subtree was created. We call this version ADWIN-DT DET.
• using multiple instances of ADWIN in each node, as estimators of frequent statistics, that is, replacing the \( n_{ijk} \) counters in CVFDT. Additionally, these ADWIN instances will give an alarm when a change in the attribute-class statistics at that node is detected, which indicates a possible concept change. We call this version ADWIN-DT DET+Est.

• Both. We call this version ADWIN-DT DET+Est.

In either case, when any instance of ADWIN at a node detects change, we begin to build a new alternate tree without splitting any attribute. Using two ADWIN instances at every node, we monitor the average error of the decision subtree rooted at this node and the average error of the new alternate subtree. When there is enough evidence (as stated by the two ADWIN) that the new alternate tree is doing better than the original decision subtree, we replace the original decision subtree by the new alternate subtree. Figure 4 shows the pseudo-code of this algorithm, ADWIN-DT.

The main advantages of this new method are:

• All relevant statistics from the examples are kept in the nodes. There is no need for an additional window to store examples not discarded yet. So, we do not need to choose its size, to store and delete examples. For medium window sizes, this factor substantially reduces our memory consumption with respect to CVFDT.

• CVFDT stores only a bounded part of the window in main memory. The rest (most of it, for large window sizes) is stored in disk. Our algorithm keeps all its data in main memory. Additionally, there is no need in our approach to recheck the whole tree at given intervals, because whenever a node undergoes change, the associated ADWIN will call for attention. As we will see, these two effects compensate to a large extent the overhead in running time associated to updating the ADWIN instances.

• The alternates trees are created as soon as change is detected, without having to wait that a fixed number of examples arrives after the change. Furthermore, the more abrupt the change is, the faster a new alternate tree will be created.

• We replace the old trees by the new alternates trees as soon as there is evidence that they are more accurate, rather than having to wait for another fixed number of examples.

The last two effects can be summarized as saying that our algorithm adapts itself to the scale of time change in the data, rather than having to rely on the a priori guesses made by the user.

We compare the memory complexity of ADWIN-DT and CVFDT. We observe that CVFDT needs to store a window of length \( W \) of examples, and for each node of the decision tree a counter for each class, attribute, and value of attribute. Denoting

- \( E \): size of an example
- \( A \): number of attributes
- \( V \): maximum number of values for an attribute
- \( C \): number of classes
- \( T \): number of nodes

then CVFDT complexity is \( O(WE + TAVC) \). As ADWIN-DT doesn’t need to store a window of examples, its complexity is reduced to \( O(TAVC + T \log W) \) using ADWIN only to detect change. ADWIN-DT using ADWIN also as an estimator of node statistics, has a complexity of \( O(TAVC \log W) \). The term \( WE \) can be huge, depending on the size of the window \( W \).

5. Theoretical Performance Guarantee

In this section we show a performance guarantee about the error rate of ADWIN-DT. Informally speaking, it states that after a change followed by a stable period, ADWIN-DT’s error rate will decrease at the same rate as that of VFDT, after a transient period that depends only on the magnitude of the change.

We consider the following scenario: Let \( C \) and \( D \) be arbitrary concepts, that can differ both in example distribution and label assignments. Suppose the input data sequence \( S \) is generated according to concept \( C \) up to time \( t_0 \), that it abruptly changes to concept \( D \) at time \( t_0 + 1 \), and remains stable after that. Let ADWIN-DT be run on sequence \( S \), and \( e_1 \) be error(ADWIN-DT,\( S,t_0 \)), and \( e_2 \) be error(ADWIN-DT,\( S,t_0 + 1 \)), so that \( e_2 - e_1 \) measures how much worse the error of ADWIN-DT has become after the concept change.

Here error(ADWIN-DT,\( S,t \)) denotes the classification error of the tree kept by ADWIN-DT at time \( t \) on \( S \). Similarly, error(VFDT,\( D,t \)) denotes the expected error rate of the tree kept by VFDT after being fed with \( t \) random examples coming from concept \( D \).

**Theorem 3** Let \( S \), \( t_0 \), \( e_1 \), and \( e_2 \) be as described above, and suppose \( t_0 \) is sufficiently large w.r.t. \( e_2 - e_1 \). Then for every time \( t > t_0 \), we have

\[
\text{error}(\text{ADWIN-DT}, S, t) \leq \min\{e_2, e_{\text{VFDT}}\}
\]

with probability at least \( 1 - \delta \), where

\[
\delta = \frac{e_2 - e_1}{\text{error}(\text{VFDT}, S, t_0)}
\]
• \( e_{VFD T} = error(VFD T, D, t - t_0 - g(e_2 - e_1)) + O(\frac{1}{\sqrt{t - t_0}}) \)

• \( g(e_2 - e_1) = 8/(e_2 - e_1)^2 \ln(4t_0/\delta) \)

The following corollary is a direct consequence, since \( O(1/\sqrt{t - t_0}) \) tends to 0 as \( t \) grows.

**Corollary 1** Suppose that \( error(VFD T, D, t) \) tends to some quantity \( \epsilon \leq e_2 \) as \( t \) tends to infinity. Then \( error(ADWIN-DT, S, t) \) tends to \( \epsilon \) as \( t \) tends to infinity too.

Proof. Note: The proof is only sketched in this version.

We know by the ADWIN False negative rate bound that with probability \( 1 - \delta \), the ADWIN instance monitoring the error rate at the root shrinks at time \( t_0 + n \) if

\[
|e_2 - e_1| > 2\epsilon_{cut} = \sqrt{2/m \ln(4(t - t_0)/\delta)}
\]

where \( m \) is the harmonic mean of the lengths of the sub-windows corresponding to data before and after the change. This condition is equivalent to

\[
m > 4/(e_1 - e_2)^2 \ln(4(t - t_0)/\delta)
\]

If \( t_0 \) is sufficiently large w.r.t. the quantity on the right hand side, one can show that \( m \) is, say, less than \( n/2 \) by definition of the harmonic mean. Then some calculations show that for \( n \geq g(e_2 - e_1) \) the condition is fulfilled, and therefore by time \( t_0 + n \) ADWIN will detect change.

After that, ADWIN-DT will start an alternative tree at the root. This tree will from then on grow as in VFD T, because ADWIN-DT behaves as VFD T when there is no concept change. While it does not switch to the alternate tree, the error will remain at \( e_2 \). If at any time \( t_0 + g(e_1 - e_2) + n \) the error of the alternate tree is sufficiently below \( e_2 \), with probability \( 1 - \delta \) the two ADWIN instances at the root will signal this fact, and ADWIN-DT will switch to the alternate tree, and hence the tree will behave as the one built by VFD T with \( t \) examples. It can be shown, again by using the False Negative Bound on ADWIN, that the switch will occur when the VFD T error goes below \( e_2 - O(1/\sqrt{t}) \), and the theorem follows after some calculation.

Note that, unlike in VFD T, the theorem depends only on one user-defined parameter, the confidence \( \delta \). The rest of the quantities \((e_1, e_2, t_0, \epsilon)\) are defined from the input sequence.

6. Experiments

We test ADWIN-DT using synthetic and real datasets. In the experiments with synthetic datasets, we use the SEA Concepts [14] and a changing concept dataset based on a rotating hyperplane explained in [11]. In the experiments with real datasets we use the UCI Adult dataset [1] from the UCI repository of machine learning databases. In all experiments, we use the values \( \delta = 10^{-4}, T_0 = 20,000, T_1 = 9,000, \) and \( T_2 = 1,000 \), following the original CVFDT experiments [11].

First, we experiment using the SEA concepts, a dataset with abrupt concept drift, first introduced in [14]. This artificial dataset is generated using three attributes, where only the two first attributes are relevant. All three attributes have values between 0 and 10. We generate 400,000 random samples. We divide all the points in blocks with different concepts. In each block, we classify using \( f_1 + f_2 \leq \theta \), where \( f_1 \) and \( f_2 \) represent the first two attributes and \( \theta \) is a threshold value. We use threshold values 9, 8, 7, and 9.5 for the data blocks. We inserted about 10% class noise into each block of data.

We test ADWIN-DT using discrete and continuous attributes. The on-line errors results for discrete attributes are shown in Table 1. On-line errors are the errors measured each time an example arrives with the current decision tree, before updating the statistics. Each column reflects a different speed of concept change. We observe that CVFDT best performance is not always with the same example window size, and that there is no optimal window size. The different versions of ADWIN-DT have a very similar performance, essentially identical to that of CVFDT with optimal window size for that speed of change. More graphically, Figure 5 shows its learning curve using continuous attributes for a speed of change of 100,000. Note that at the points where the concept drift appears ADWIN-DT decreases its error faster than CVFDT, due to the fact that it detects change faster.

**Table 1.** SEA on-line errors using discrete attributes with 10% noise

<table>
<thead>
<tr>
<th>Window</th>
<th>Change speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,000</td>
</tr>
<tr>
<td>CVFDT</td>
<td>19.47%</td>
</tr>
<tr>
<td>ADWIN-DT DET</td>
<td>17.01%</td>
</tr>
<tr>
<td>ADWIN-DT EST</td>
<td>17.72%</td>
</tr>
<tr>
<td>ADWIN-DT DET+EST</td>
<td>17.72%</td>
</tr>
</tbody>
</table>

Other common dataset is the rotating hyperplane used as testbed for CVFDT versus VFD T in [11]. A hyperplane in \( d \)-dimensional space is the set of points \( x \) that satisfy

\[
\sum_{i=1}^{d} w_i x_i \geq w_0
\]
where \( x_i \) is the \( i \)th coordinate of \( x \). 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.

We experiment with abrupt and with gradual drift. In the first set of experiments, we apply abrupt change. We use 2 classes, \( d = 5 \) attributes, and 5 discrete values per attribute. We do not insert class noise into the data. After every \( N \) examples arrived, we change the sign of the classification. So, we classify the first \( N \) examples using \( \sum_{i=1}^{d} w_i x_i \geq w_0 \) and the next \( N \) examples using \( \sum_{i=1}^{d} w_i x_i \leq w_0 \).

The on-line errors are shown in Table 2, where each column reflects a different value of \( N \), the period for classification change. We detect that ADWIN-DT methods substantially outperform CVFDT in all speed changes.

Table 2. On-line errors of Hyperplane Experiments with abrupt concept drift

<table>
<thead>
<tr>
<th>Window</th>
<th>Change speed</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVFDT</td>
<td></td>
<td>30.10%</td>
<td>22.07%</td>
<td>19.00%</td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td>32.49%</td>
<td>23.25%</td>
<td>19.56%</td>
</tr>
<tr>
<td>10,000</td>
<td></td>
<td>29.02%</td>
<td>23.74%</td>
<td>19.59%</td>
</tr>
<tr>
<td>100,000</td>
<td></td>
<td>19.94%</td>
<td>8.40%</td>
<td>7.81%</td>
</tr>
<tr>
<td>ADWIN-DT</td>
<td>DET</td>
<td>20.05%</td>
<td>9.47%</td>
<td>7.99%</td>
</tr>
<tr>
<td>ADWIN-DT</td>
<td>EST</td>
<td>11.87%</td>
<td>9.27%</td>
<td>7.60%</td>
</tr>
<tr>
<td>ADWIN-DT</td>
<td>DET+EST</td>
<td>11.87%</td>
<td>9.11%</td>
<td>7.68%</td>
</tr>
</tbody>
</table>

In the second type of experiments, we look at gradual drift. We vary the first attribute over time slowly, from zero to a maximum value of \( 1/2 \), and then from \( 1/2 \) to zero, linearly as a triangular wave. We adjust the rest of weights in order to have the same number of examples for each class.

The on-line errors rates are shown in Table 3. Observe that, in contrast to previous experiments, ADWIN-DT DET and ADWIN-DT DET+EST do much better than ADWIN-DT DET. We believe this will happen often in the case of gradual changes, because gradual changes will be detected earlier in individual attributes than in the overall error rate.

Table 3. On-line errors of Hyperplane Experiments with gradual concept drift

<table>
<thead>
<tr>
<th>Window</th>
<th>Change speed</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVFDT</td>
<td></td>
<td>20.74%</td>
<td>8.40%</td>
<td>8.21%</td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td>17.06%</td>
<td>19.94%</td>
<td>7.81%</td>
</tr>
<tr>
<td>10,000</td>
<td></td>
<td>16.59%</td>
<td>16.57%</td>
<td>17.59%</td>
</tr>
<tr>
<td>100,000</td>
<td></td>
<td>20.05%</td>
<td>8.40%</td>
<td>7.81%</td>
</tr>
<tr>
<td>ADWIN-DT</td>
<td>DET</td>
<td>20.05%</td>
<td>9.47%</td>
<td>7.99%</td>
</tr>
<tr>
<td>ADWIN-DT</td>
<td>EST</td>
<td>11.87%</td>
<td>9.27%</td>
<td>7.60%</td>
</tr>
<tr>
<td>ADWIN-DT</td>
<td>DET+EST</td>
<td>11.87%</td>
<td>9.11%</td>
<td>7.68%</td>
</tr>
</tbody>
</table>

We test ADWIN-DT on a real dataset in two different ways: with and without concept drift. We use the UCI Adult dataset [1], a stable dataset without concept drift. It aims to predict whether a person makes over 50k a year, and it was created based on census data. It consists of 48,842 instances, 14 attributes (6 continuous and 8 nominal) and missing attribute values. In order to test our method robustness to false alarms, we use this original dataset UCI Adult dataset. As expected, ADWIN-DT doesn’t detect drift.

An important problem with most of the real-world benchmark data sets is that there is little concept drift in them [15] or the amount of drift is unknown, so in many research works, concept drift is introduced artificially.

We simulate concept drift by ordering the UCI Adult dataset by one of its attributes, education. Figure 6 shows the results, comparing ADWIN-DT DET to CVFDT using different window sizes. We observe that CVFDT on-line error decreases when the example window size increases, and that ADWIN-DT on-line error is lower for all window sizes.

7. Time and memory

In this section, we discuss briefly the time and memory performance of ADWIN-DT.

All programs were implemented in C modifying and expanding the version of CVFDT available from the
VFML [10] software web page. We have slightly modified the CVFDT implementation to follow strictly the CVFDT algorithm explained in the original paper by Hulten, Spencer and Domingos [11]. The experiments were performed on a 2.0 GHz Intel Core Duo PC machine with 2 Gigabyte main memory, running Cygwin on Microsoft Windows Vista.

Consider the experiments on SEA Concepts, with different speed of changes: 1,000, 10,000 and 100,000. Figure 7 shows the memory used on these experiments. As expected by memory complexity described in section 2, ADWIN-DT DET, is the method that uses less memory. The reason for this fact, is that it doesn’t keep examples in memory as CVFDT, and that it doesn’t store ADWIN data for all attributes, attribute values and classes, as the other versions of ADWIN-DT. One additional parameter of CVFDT is the size of the example window maintained in memory, 10,000 by default. So, the memory that CVFDT uses depends on the size of the example window maintained in memory. In these experiments for $W = 10,000$ and $W = 100,000$, as CVFDT uses a 10,000 example window in memory, CVFDT memory was a factor of 10 greater than ADWIN-DT DET memory.

Figure 8 shows the number of nodes used in the experiments of SEA Concepts. We see that the number of nodes is similar for all methods, confirming that the good results on memory of ADWIN-DT DET is not due to smaller size of trees.

Finally, with respect to time we see that CVFDT is still the fastest method, but ADWIN-DT DET has a very similar performance, a remarkable fact given that it is monitoring all the change that may occur in any node of the main tree.
and all the alternate trees. Other versions of ADWIN-DT increases time by a factor of 4, so they are still usable if time or data speed is not the main concern.

8. Conclusions and Future Work

We have presented three variants of algorithm ADWIN-DT, a decision tree miner for data streams based on CVFDT. Contrary to CVFDT, it has theoretical guarantees of performance, relative to those of VFDT. In our experiments, ADWIN-DT is always as accurate as CVFDT and, in some cases, it has substantially lower error. Its running time is only slightly higher, and its memory consumption is remarkably smaller, often by an order of magnitude.

An obvious future work is experimenting with real-world datasets that are both larger and have some quantifiable drift, so that one can make meaningful comparisons among algorithms for dealing with time drift. As remarked before, these datasets do not abound.

Additionally, we would like to compare our method with Gama’s UFFT [9]; unfortunately, there is no publicly available software, and this has delayed our experimentation. Since our change detection method ADWIN often outperformed the one underlying UFFT in our previous experiments [4], we expect that UFFT would benefit from being combined with ADWIN.

References