Tie Breaking in Hoeffding Trees

Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{geoff, rkirkby, bernhard}@cs.waikato.ac.nz

Abstract. A thorough examination of the performance of Hoeffding trees, state-of-the-art in classification for data streams, on a range of datasets reveals that tie breaking, an essential but supposedly rare procedure, is employed much more than expected. Testing with a lightweight method for handling continuous attributes, we find that the excessive invocation of tie breaking causes performance to degrade significantly on complex and noisy data. Investigating ways to reduce the number of tie breaks, we propose an adaptive method that overcomes the problem while not significantly affecting performance on simpler datasets.

1 Introduction

The Hoeffding tree induction algorithm [2] has proven to be one of the best methods for data stream classification. The algorithm is refined and realised in a system known as VFDT (Very Fast Decision Tree learner) which encompasses a number of practical considerations. One of these is connected with ties. Ties occur when two or more attributes have close split evaluation values. Instead of waiting to see which attribute is superior, a potentially wasteful exercise, VFDT forces a split to be made on one of the attributes as long as the difference between the split evaluation values is within user specified bounds. Tie breaking has not been studied in any great detail, but it is to be supposed that its use is rare in practice.

Ties do need to be broken otherwise tree growth is stunted and performance degraded. They are likely to occur for both nominal and numeric attributes. In Section 5 for example, we see reasonable gains on a dataset with no numeric attributes (*led*), however, numeric attributes are arguably more problematic as they are difficult to manage in a stream setting.

Sophisticated methods for handling numeric attributes are likely to be impractical (see Section 3.1). Simpler methods may be more prone to inadvertently generating the conditions under which excessive tie breaking occurs. This phenomenon was observed when we introduced a lightweight method of handling continuous attributes in pursuit of a robust and practical Hoeffding tree implementation [6]. Experiments found the method to work reasonably well, although when noise was added and the concept complexity increased we uncovered cases where it performed poorly. We attributed this failure to premature tie breaking, prompting us to explore methods of improving tie breaking in Hoeffding trees.

The paper is arranged as follows. Section 2 outlines our motivation for looking at tie breaking. Section 3 discusses the handling of numeric attributes in Hoeffding trees and introduces methods aimed at improving tie breaking. Section 4 describes our experimental methodology and Section 5 looks at the results. Finally, Section 6 concludes the paper.

2 Motivation

Several variations of Hoeffding tree induction are present in the literature, all based on the original VFDT system [2]. The variations attempt to improve VFDT but are fundamentally the same [3, 4, 8].

Hoeffding trees work by collecting sufficient statistics in the leaves of the tree of the training instances that reach them. Periodically, these leaves are checked to compare the relative merits of each candidate attribute for splitting. The Hoeffding bound or similar metric is used to decide when to be confident that the best candidate is better than the others. At this point the leaf is split on the best attribute, allowing the tree to grow.

Typically, information gain is used to rank the merits of the split candidates, although other metrics could be substituted. In the case of discrete attributes, it is sufficient to collect counts of attribute labels relative to class labels to compute the information gain afforded by a split.

The implementation discussed in this paper is very close to VFDT, it uses information gain as the split criterion, the original VFDT Hoeffding bound formulation to determine when to split (using parameters $\delta = 10^{-6}$, $\tau = 5\%$, and $n_{min} = 300$), and handles numeric attributes by Gaussian approximation (see Section 3.1).

While stress-testing this implementation ([6], which does not address tie-breaking) an anomaly was found on a particularly complex dataset. On this data (*rrbfc*—described in Section 4.2) we found that the tree induced was uncommonly deep. Further inspection revealed that the depth of the tree was equal to the number of nodes, effectively producing a list. Clearly this is not an intended outcome of the algorithm, and does not lead to high predictive accuracy.

Studying the split decisions faced by the tree shows the complexity of the problem. Figure 1 shows the estimated merit of the various attributes at the root node as an increasing number of instances are observed. The 50 numeric attributes are difficult to separate, the estimates are erratic early on and things do not appear to stabilize until some time after 10,000 instances. Thus the problem appears to center on how ties are handled.

VFDT uses a parameter τ to control ties. Users set the parameter so that when the Hoeffding bound falls below this threshold a tie is broken in favour of the current best attribute. The default setting is a tie error of 0.05 ($\tau = 5\%$) and with the settings discussed earlier ($\delta = 10^{-6}$, $n_{min} = 300$), a tie will always be broken regardless of the observed merit after 3000 instances have made it to

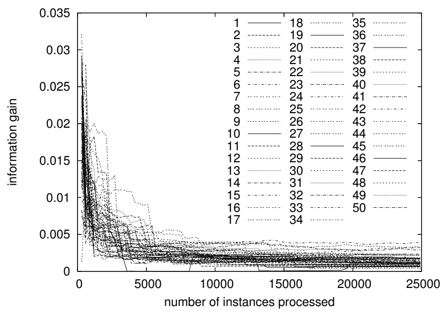

Fig. 1. Infogain of the 50 attributes for the root node on the *rrbfc* dataset

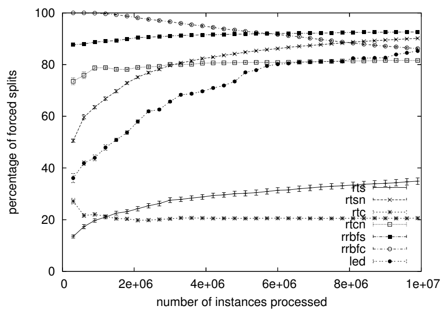

Fig. 2. Percentage of splits that are forced by tie breaking on various datasets

a leaf (2764 is the precise point, but the 300 instance grace period means the tree waits slightly longer).

Effectively the τ parameter controls a minimum growth speed, and it is the ability of the attributes to be separated prior to that point that determines how much faster the tree can grow.

Three thousand instances seems a very early point to make a reasonable decision on the *rrbfc* data, and low in general. Figure 2 shows that of all the split decisions made by the algorithm on several datasets (introduced in Section 4.2), the majority of ties are broken before the Hoeffding bound is confident enough to decide. Two of the datasets end with less than 40% tie breaking (these are simple noise-free concepts), but the rest complete with over 80%.

The percentage of ties being broken is related to the complexity of the problem. The fact that even the simplest concept (*rts*) still requires about 20% tie breaking is revealing. In general these percentages are much higher than expected.

On *rrbfc*, we found that the tree becomes balanced and accuracy improves dramatically if the τ parameter is reduced, deferring split decisions longer before being forced. This implies that the problem lies with premature forcing of split decisions.

The problem cannot be solved, however, by eliminating tie breaking. Comparing Figure 3 with Figure 4, without tie breaking the trees grow much slower or in the extreme case (*led*) not at all, resulting in poor accuracy.

Adjusting the fixed τ parameter to fit a particular dataset is problematic as our experiments indicate that there is no single default value that works well in all situations for this problem.

There is a fine balance to maintain, as tree size often has a close relationship to the overall accuracy of the tree. Larger trees are capable of dividing the problem space into smaller regions, more closely approximating the target concept. It is for this reason that solutions slowing tree growth can suffer in predictive accuracy.

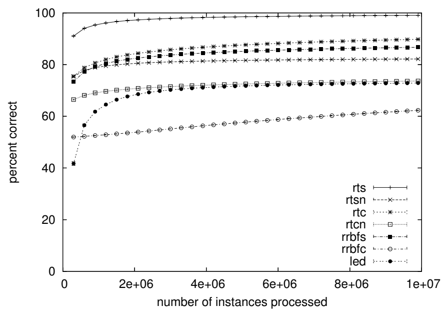

Fig. 3. Default tie breaking

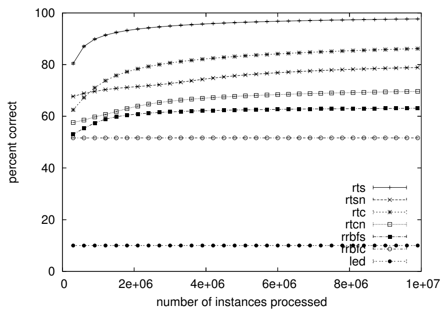

Fig. 4. No tie breaking

Another way to solve the degenerate tree problem is to improve the method of choosing numeric split points. It appears that the simple Gaussian approximation breaks down when too few examples have been witnessed. This results in poor split choices, in the *rrbfc* case so poor that no decent tree structure can be formed.

Section 3.1 describes some alternative methods of choosing numeric split points. Unfortunately they are impractical for large-scale experiments. The fixed width binning and VFDT-style methods are too slow and often exceed enforced memory limits. Thus methods such as Gaussian approximation need more attention as they have the potential to satisfy data stream constraints. This method works well apart from the obvious problem on *rrbfc*.

3 Numeric Splits and Tie Breaking

Continuous attributes pose a difficult problem in resource-bounded settings. In batch tree learning it is possible to evaluate all possible split points to determine the optimal choice. The data stream setting prohibits such analysis as it requires memorization of every observed value.

3.1 Methods of Choosing Continuous Splits

Gaussian Approximation The method considered in this paper approximates the distribution of values by a single Gaussian. The implementation most similar to this is given in [3]. We differ in that a heuristic capable of handling multiple classes is used. The range between the minimum and maximum observed values is divided equally into n parts (in our case n=100). For each of these split possibilities we estimate the distribution of classes to the left and right based on the area under the normal distributions. While this approach is relatively crude compared to other techniques it is also highly efficient, requiring the update of a few counts per class, and performs well in practice.

Incremental Fixed Width Binning We experimented with a method that works by tracking the minimum and maximum values observed at a node, dividing the range into a fixed number of bins. A separate range is maintained for each class label. The weight of the bins is incremented as values fall into their respective ranges. Whenever a new minimum or maximum is seen the old bins are redistributed into the new range by interpolating the fraction covered by each new bin. This method is not very adaptive and easily thrown by outliers. It will not focus attention on the more interesting areas of the range and may waste storage on uninteresting regions.

VFDT implementation The papers describing VFDT do not describe a practical method for handling numeric values. The publicly released implementation of VFDT in the VFML [7] software can handle continuous attributes, although no literature has been published on the method. A binning approach is used, where for every new unique value observed a new bin is inserted into the range. A single set of bins is shared by all of the classes. Once a hard-coded limit of 1000 bins are present the bins remain stationary but their counts are incremented. This method is more adaptive than the fixed width approach, but is still limited in that the bins are only influenced by the first 1000 unique values. It is also the most expensive method, requiring the most processing time and memory usage. This is especially true on synthetic data where it is very unlikely for the exact same value to reappear in the data.

Discussion Other methods have been proposed, such as those described in [4] and [8]. It is an open problem as to how well these perform under highly resource bounded situations. Work related to single-pass quantile estimation ([5]) could well provide useful contributions to this problem.

We found while experimenting with our simple fixed-width approach and the VFDT method that they would add significant space and time costs over the Gaussian approach. With the Gaussian approach it was possible to complete the experiments within a reasonable time frame (100 million instances within 12 hours means at least 2000 instances processed per second), and have the trees occupy a reasonable amount of space (less than 256MB). Experiments with the other two approaches could not keep within these limits.

[3] is nearly identical to our Gaussian approach, but restricted to two classes. As mentioned in that paper, requiring only the mean and variance per class is a major advantage over other approaches.

3.2 Improving Tie Breaking

We consider two alternative attempts to improve tie breaking.

Genuine Tie Detection (*htdt*) The original motivation for tie breaking in Hoeffding trees is that it allows competition between competing split candidates to be resolved, preventing tree growth from being stalled. Without it, two attributes that are equally favoured by the split criterion will prevent a decision from being made, when either of them would be perfectly suitable. The tie breaking problem observed on *rrbfc* can be viewed as ties being broken before sensible decisions are possible, rather than when a genuine tie situation emerges. This observation inspired the method of trying to distinguish between a genuine tie and a lack of suitable candidates.

This approach works by comparing the difference between the best and second-best candidates with the difference between the second-best and worst candidate. In a genuine tie situation one would expect the difference between the best candidates to be much smaller. A parameter α determines how many times smaller the gap should be before it is considered a tie. If $\overline{G}(X_a)$ is the best gain, $\overline{G}(X_b)$ is the second-best and $\overline{G}(X_c)$ is the worst, a split will be allowed on X_a only if $\alpha \times (\overline{G}(X_a) - \overline{G}(X_b)) < (\overline{G}(X_b) - \overline{G}(X_c))$.

Increasing Wait Time After Ties are Broken (*htat*) Another way to overcome the problem is to detect when ties are being broken often, and take steps to reduce the number of ties that occur. The method works by starting with the default tie breaking wait period, but once a tie is broken, the children below that node have their wait period increased by a fixed amount. The effect is cumulative—multiple tie breaking down a particular path will result in an increasingly longer wait period. As soon as a regular split is found the wait period is reset to the default amount.

This method is intended to allow the tree to slowly adapt, taking longer to consider those regions of the space for which decisions are harder to make, while not penalizing tree growth too harshly in easier sections. The parameters controlling this method are the default tie breaking period (which is fixed at 3000) and the increment that is added to the children of broken ties.

For example, if the increment is set to 100, then the first tie will be broken after 3000 instances, but the children of the split will have ties broken after 3100 instances. Children of ties broken after 3100 will then break ties after 3200 instances and so on, whereas children of regular splits will wait the default 3000 instances. Paths where multiple ties are broken will keep incrementing the wait period until sufficient information is available to determine a winner without calling a tie. As soon as a split can be decided before the wait period expires, the wait period in the subtree below reverts back to the default.

4 Experimental setup

4.1 Testing Methodology

To date, most data stream evaluations have used train/test splits. Typically an independent test set is kept aside (easy to do when data is so abundant), and periodically (or perhaps only at the end of learning), the learned model is queried to measure the accuracy achieved on the test set. Even though data is abundant, this particular methodology still requires a parameter to determine the relative sizes of these splits.

Data streams present unique opportunities for evaluation, due to the volume of data available and the any-time property of the algorithms under examination. We consider a method of evaluation that exploits this property whilst maximizing use of the data. Each instance is used exactly once for both training and testing, producing a smooth plot of performance.

A snapshot can be taken at any point during the evaluation process. These snapshots can be plotted graphically to produce learning curves. Typically the statistic of most interest is related to the percentage of instances that were correctly classified. The resulting curve is smooth between snapshots because as the number of processed instances increases, the influence of a single classification/misclassification on the overall percentage becomes smaller.

As data order can be a source of concern in data stream classification, the evaluation method presented here can be extended to average over many different orderings. The evaluation algorithm is repeated multiple times, each time the data is presented in a different order (or in the case of data generators, an independent set of examples). The snapshots at each data point are collected together and averaged between runs. This extension allows analysis of the variance of the performance over several data orders.

4.2 Data Sources

The data sets in this paper consist of three synthetically generated and one real dataset.

Random tree The simple (*rts*) and complex (*rtc*) random tree generators are described in [6]. *rtsn* and *rtcn* refer to versions with 10% added noise.

Random RBF Random RBF data is generated by first creating a random set of centers for each class. Each center is randomly assigned a weight, a central point per attribute, and a standard deviation. To generate new instances, a center is chosen at random taking the weights of each center into consideration. Attribute values are randomly generated and offset from the center, where the overall vector has been scaled so that its length equals a value sampled randomly from the Gaussian distribution of the center. The particular center chosen determines the class of the instance. Random RBF data contains only numeric attributes as it is non-trivial to include nominal values. *rrbfs* refers to a simple random RBF dataset—100 centers, 10 attributes and 2 classes. *rrbfc* is more complex—1000 centers, 50 attributes and 2 classes.

LED This synthetic generator comes from the UCI [1] repository—the LED dataset—allowing us to generate the desired 10 million instances. The particular configuration used of the LED generator produced 24 binary attributes, 10 classes and 10% noise (*led*).

Covertype One of the largest datasets containing real-world data in the UCI repository is the Forest Covertype dataset. This consists of 581,012 instances, 10 numeric attributes, 44 binary attributes and 7 classes (*ct*).

5 Results and Discussion

Table 1 lists the final accuracies, along with the standard error over 10 runs. Figure 5 displays the results of testing the tie breaking modifications described in Section 3.2, over the datasets described in Section 4.2. Note that the order that the algorithms are displayed in each legend corresponds to the order of the final accuracies achieved. Line spaces have also been added to the legend to show which algorithms are grouped together in the figure. On all of the synthetic datasets 10 million instances were processed in 10 runs. The exception is *ct* on which the entire set of 581,012 instances were used. To do 10 runs over this data the instances were randomly shuffled 10 different ways.

For the *htdt* method, α was set to 5 after smaller experiments suggested it to be a practical choice. This means that the second-worst gap needed to be 5 times greater than the best-second gap before a split was allowed. Intuitively this is a conservative number but in practice it turned out that expecting a larger gap ratio would frequently stall tree growth enough to harm accuracy.

As can be seen in both Table 1 and Figure 5 the *htdt* modification performed worse than default *ht* tie breaking on all datasets besides *led* on which it was equal and *rrbfc* on which it hardly improved. This method also varied the most, having higher standard error on several datasets. Clearly this modification does not improve the original scheme.

Three different parameter settings were tested for the *htat* method—a small increment of 100, a mild increment of 500 and the full default tie period of 3000. The trend on the majority of datasets is the the larger the increment the higher the performance drop. The gap between 100 and 3000 is most noticeable on *rrbfs*. The gap between 100 and 500 is smaller and not very noticeable on most datasets.

On the random tree data and *rrbfs* the *htat* methods are on a par or slightly worse than the standard *ht*. On the remaining datasets the *htat* methods are superior. *rrbfc* is the most curious case—in this instance *htat100* does poorly, and *htat500* outperforms *ht3000* after starting out worse.

Based on these results we find *htat500* to be the best compromise. It does almost as well as *htat100* on those datasets where *htat100* does best, and overcomes the problem on *rrbfc* to achieve the highest accuracy.

6 Conclusions

This paper introduced two methods intended to improve tie breaking in Hoeffding trees. The modifications were motivated by the observation that ties are broken more often than expected over a range of different datasets, and that on one particular dataset this caused performance to suffer badly.

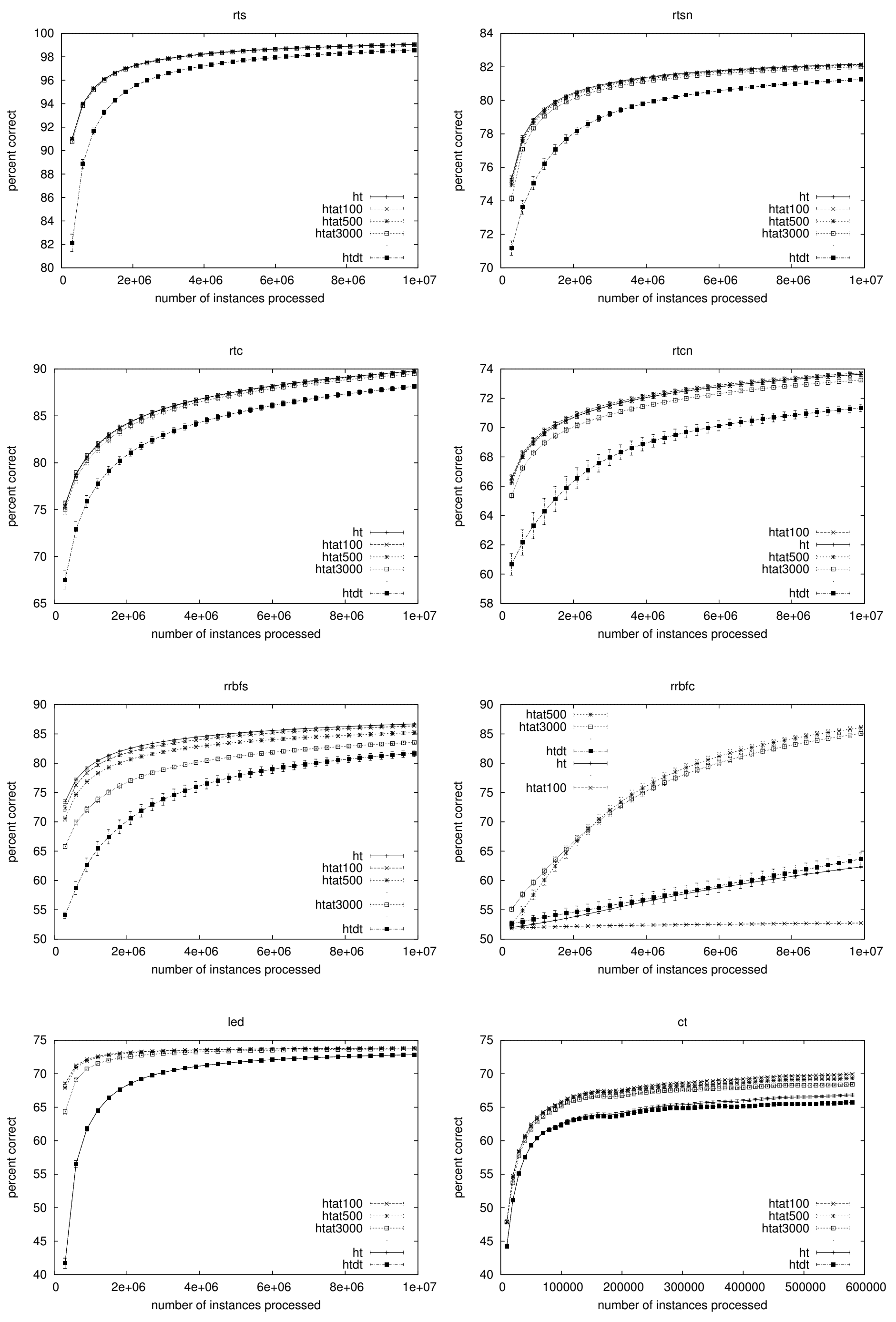

Fig. 5. Accuracy curves of the various tie breaking methods

	ht	htdt	htat100	htat500	htat3000
rts	99.06 ± 0.03	98.56 ± 0.03	99.06 ± 0.03	99.05 ± 0.03	99.04 ± 0.03
rtsn	82.17 ± 0.03	81.24 ± 0.04	82.14 ± 0.03	82.10 ± 0.03	82.01 ± 0.03
rtc	89.79 ± 0.17	88.15 ± 0.24	89.77 ± 0.17	89.71 ± 0.18	89.49 ± 0.22
rtcn	73.64 ± 0.15	71.34 ± 0.25	73.74 ± 0.12	73.63 ± 0.10	73.24 ± 0.12
rrbfs	86.71 ± 0.05	81.69 ± 0.54	86.37 ± 0.10	85.23 ± 0.31	83.56 ± 0.24
rrbfc	62.31 ± 0.10	63.67 ± 1.03	52.74 ± 0.03	86.06 ± 0.42	85.08 ± 0.35
led	72.85 ± 0.03	72.84 ± 0.03	73.83 ± 0.01	73.82 ± 0.01	73.70 ± 0.01
ct	66.83 ± 0.16	65.71 ± 0.17	69.94 ± 0.09	69.33 ± 0.10	68.41 ± 0.11

Table 1. Final accuracies achieved

The *htdt* method did not overcome the problem, but the adaptive *htat* method shows more promise. *htat* enhanced the performance of Hoeffding trees employing a Gaussian approximation for numeric split choices in several complex cases, while not harming performance greatly on simpler data. It also enhanced performance on a dataset comprised entirely of binary attributes (*led*) and a real-world dataset, suggesting that the default VFDT method of tie breaking can lower classification accuracy when the data is sufficiently complex.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
2. Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
3. Joao Gama, Pedro Medas, and Ricardo Rocha. Forest trees for on-line data. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 632–636, New York, NY, USA, 2004. ACM Press.
4. Joao Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528, New York, NY, USA, 2003. ACM Press.
5. Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.
6. Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. Stress-testing hoeffding trees. In *Proceedings of the Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD05)*, 2005.
7. Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams. http://www.cs.washington.edu/dm/vfml/, 2003.
8. Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.