# Use of Hoeffding Trees in Concept Based Data Stream Mining

Stefan Hoeglinger
School of Computing and Mathematical Science,
Auckland University of Technology
*stehoe92@aut.ac.nz*

Russel Pears
School of Computing and Mathematical Science,
Auckland University of Technology
*russel.pears@aut.ac.nz*

*Abstract*— **Recent research in data mining has focussed on developing new algorithms for mining high-speed data streams. Most real-world data streams have in common that the underlying data generation mechanism changes over time, introducing so-called concept drift into the data. Many current algorithms incorporate a time-based window to be able to cope with drift in order to keep their model up-to-date with the data stream. A major problem with this approach is the potential loss of valuable information as data slides out of the time window. This is particularly a concern in those environments where patterns recur. In this paper, we present a concept-based window approach, which is integrated with a high-speed decision tree learner. Our approach uses the content of the data stream itself in order to decide which information is to be erased. Several methodologies, all based around minimising the overall information loss when pruning the decision tree, are discussed.**

*Keywords*— **data stream mining, hoeffding bound, decsion trees, information gain**

## I. INTRODUCTION

Knowledge discovery systems are typically constrained by three main limited resources; processing time, memory, and sample size. Ideally, such systems should operate continuously and indefinitely, and should never lose potentially valuable information [1-3]. Such criteria can be fulfilled only partially by incremental learning algorithms as these methods also have significant shortcomings, such as efficiency, instance arrival order, and speed [2, 4, 5].

The speed versus memory and accuracy trade-offs are becoming increasingly acute as data sources used for data mining get larger or even infinite as in the case of online and real-time applications such as sensor reading and monitoring, telecommunications data and credit card transactions. Real-time necessities create an additional level of complexity: the need for high-speed processing.

As the run-time and resource requirements of established algorithms such as C4.5 are excessive in such environments, recent research has focussed on modifying these established methods and developing new algorithms for mining high-speed data streams [6, 7]. New algorithms allow online and incremental learning, while adaptive approaches attempt to deal with changing concepts in the underlying data stream. This activity gave birth to a new field of research in data mining, called high-speed data stream mining, which features algorithms that can handle as many as tens of thousands of instances per second [4, 13]. Most of these algorithms use time-based windows for aging the training model that they build. The major problem with this approach is the potential loss of valuable information as data slides out of the time window. This is particularly a concern in those environments where patterns recur over a period of time.

In this paper, we present a concept-based window approach, which is integrated with a high-speed decision tree learner. Several methodologies, all based around minimising the overall information loss when pruning the decision tree, are discussed.

In the next section, we present related research. Thereafter, in section three, we discuss an overview of the principles governing the proposed high-speed decision tree learner. Section four presents the idea of a concept-based window and contrasts it to the more commonly used time-based window approach. Next, in section five, we propose new methodologies to modify existing algorithms in order to incorporate the concept-based window construct. The paper then concludes with a short summary in section six and suggests future work.

## II. RELATED WORK

We draw a clear distinction between work on incremental and adaptive algorithms: whereas incremental algorithms tend to be faster, they also have higher storage requirements, as they do not forget already learnt knowledge. On the other hand, adaptive algorithms selectively forget instances at the possible expense of precision.

One of the earlier data stream mining algorithms was presented by Domingos and Hulten [3]. The authors describe and evaluate VFDT (Very Fast Decision Tree learner), a decision tree learning system based on *Hoeffding* trees. *Hoeffding* trees can be learned at a constant time per instance and overcome the storage limitations of classic decision tree learners such as ID3, C4.5 and SLIQ by using the *Hoeffding bound.* This bound is used to decide how many instances are required at each internal node of the decision tree in order to make statistically significant decisions on how to split the node. One good feature of the *Hoeffding bound* is that it is independent of the probability distribution of the original dataset, which in turn means that it will take more observations to reach the same conclusions than with distribution-dependent methods. The crucial feature of the *Hoeffding bound* is that it ensures that with high probability, the attribute chosen is the same as that would be chosen using an infinite number of instances. VFDT is capable of processing tens of thousands of instances per second with the use of off-the-shelf hardware

Another early incremental approach was based on generalising the so-called *change-point detection* problem, which means that by analysing a time series, certain points can be identified at which the data stream changes its behaviour [8]. They proposed an iterative algorithm that fits a model to a time segment, and uses a likelihood criterion to determine whether the segment should be partitioned further.

A different approach altogether, based on a recursive version of the k-means clustering algorithm was proposed in [9]. For this incremental algorithm, the authors use a divide-and-conquer approach to first divide the stream into partitions, each of which is then clustered recursively. They proved that a constant-factor approximation can be obtained when reclustering a constant number of times, although it cannot achieve a better run time than *O(nk)* for any k-means algorithms.

Kasabov [10] presented detailed work on Evolving Connectionist Systems (ECOS) and showed on-line learning systems that are able to trace the movement of clusters. In these systems clusters are concepts that are internally learned in the structure of a model and created from incoming windows of data. One of the described models is the On-Line Evolving Clustering Method (ECM), which is a fast single-pass incremental algorithm, which dynamically clusters the input stream without the requirement for a predefined number of clusters.

Incremental algorithms also seem to be predecessors to their adaptive counterparts: Hulten, Spencer and Domingos [4] presented an improved version of VFDT, called Concept-adapting Very Fast Decision Tree learner (CVFDT). CVFDT is able to adapt to concept-drift in the data stream, growing alternate sub-trees at each node of the decision tree, replacing the current sub-tree with the alternate whenever the latter becomes more accurate. This is achieved by maintaining sufficient statistics on a time-window moving over the data stream.

Another example of adaptive algorithms can be found in the work of [1], in which the authors propose a new forecasting framework for time-series (AWSOM) that uses limited CPU, memory, network bandwidth and power, and only requires a single pass over the data. It can adapt and handle arbitrary periodicity and can deal with various types of noise, while discovering interesting patterns and trends without the need for user intervention or expert tuning. Micro-clusters are employed in [5] to adapt the training model quickly to changes in the evolving data stream. Each of these clusters corresponds to a separate class. A certain number of micro-cluster snapshots are stored at geometrically increasing time intervals (depending on the expected total length of the data stream) in order to provide sufficient information about different time horizons.

Although much of the work in mining data streams uses adaptive approaches, most of those algorithms still implement time-based windows in order to detect changes in the concept of the underlying data stream [1, 2, 4, 5, 8, 11-16].

In [2], the authors present a different approach: They state that the most fundamental problem in learning drifting concepts is how to identify data in the training set that are no longer consistent with the current concept and propose that the expiration of such outdated data must rely on the data's distribution rather than the arrival time of the data. This is managed by training a weighted ensemble of classifiers, giving each classifier a weight based on its expected prediction accuracy on the current set of test instances.

Our method uses a concept-based window approach to guide a decision tree learner, based on the sufficient statistics used by VFDT, CVFDT and the above-mentioned ensemble approach in order to overcome shortcomings such as time-based windowing and resource requirements. The proposed algorithm allows for real-time reorganization of decision trees based on capturing changing patterns in the distribution of instances.

## III. ROLE OF HOEFFDING BOUND

Essentially, the Hoeffding bound provides a confidence interval in which the mean of a set of n independent observations lies with a given probability. Given n independent observations of variable r, and their mean being computed as $\bar{r}$, the above *Hoeffding bound* states that with a probability of 1 – δ, the true mean of the variable is at least $\bar{r}$ – ε. In addition, the Hoeffding bound has the property of being independent of the probability distribution of the process that generates r. This makes the bound more conservative than distribution-dependant ones (i.e. it takes more observations to reach the same δ and ε).

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

| | | |
|---|---|---|
| $R$ | … | Range of the real-value random variable $r$ |
| $n$ | … | Number of independent observations |
| 1-$\delta$ | … | Error probability |
| $\varepsilon$ | … | Desired error tolerance |

In a decision tree learner, the Hoeffding bound can be exploited by associating $\bar{r}$ with the mean information gain of a given attribute. This provides a statistically rigorous way of choosing the right attribute to split at decision nodes of the tree. This approach is utilized in the VFDT algorithm which first identifies the two attributes with the highest individual information gains. With a sufficient number, n instances as given by the Hoeffding bound, the difference in information gain $\Delta\overline{G}$ between the two attributes (A1, A2) with the two highest individual information gains can now be used to identify the correct attribute to split on (A1 having the higher information gain). Whenever $\Delta\overline{G} > \varepsilon$, the *Hoeffding bound* can now guarantee that A1 is the correct choice with a probability of 1 – δ.

CVFDT is an extension of the basic principles employed in VFDT. It maintains VFDT's speed and accuracy, but adds the ability to detect and respond to drift in the data stream. It achieves this by using a sliding time-based window of instances with which it tries to keep its model current.

Unlike many other algorithms, it does not need to learn the adapted model from scratch each time a change happens. Instead, each time a new instance arrives it updates statistics at the decision nodes of the tree corresponding to the new and oldest instances in the window.

Given a stationary underlying concept, this would have no statistical effect on the tree. However, when the concept is changing, some splits that previously passed the *Hoeffding test* will now fail, and the algorithm starts to grow alternate sub-trees at such nodes, because a different attribute will now have a higher information gain at this node. Over time, an alternative sub-tree can become more accurate on new instances than the original one, which causes the algorithm to replace the old sub-tree with the new one.

In order to suppress unchecked growth of alternate sub-trees, the algorithm periodically enters a testing mode, in which the performance of the current sub-tree and all alternate sub-trees is verified over the next *m* instances. If the algorithm finds alternate sub-trees that are not increasing their accuracy over time (compared to the current sub-tree), they are pruned aggressively in this test phase.

The authors of CVFDT acknowledge the main weakness of a time-based window approach, and allow for an external event in their algorithms to adapt the window size. This in turn, allows external triggers to recognise a rapidly changing or a very stable concept, meaning that if many nodes become questionable at once, a smaller window can be chosen, or if there are only a few questionable nodes, a larger window could be chosen in order to learn a more detailed model.

The undesirable need for an external algorithm to enable CVFDT to react to concept drift and the vulnerability of time-based windows in modelling recurring concepts correctly inspired us to introduce the notion of a *concept-based window*. Here content, rather than time is the driving force behind the decision to reorganize the decision tree that has already been built.

## IV. CONCEPT-BASED VS TIME-BASED DRIFT

Most real-world data streams have in common that the underlying data generation mechanism changes over time, introducing so-called concept drift into the data. Algorithms that assume stationary distributions can generate accurate models initially, but as the fundamental processes evolve over time, those models become less accurate. Therefore, many current algorithms incorporate a time-based window to be able to cope with drift of the underlying data stream in order to keep their model up-to-date with the data stream [1, 2, 4, 5, 8, 11-16].

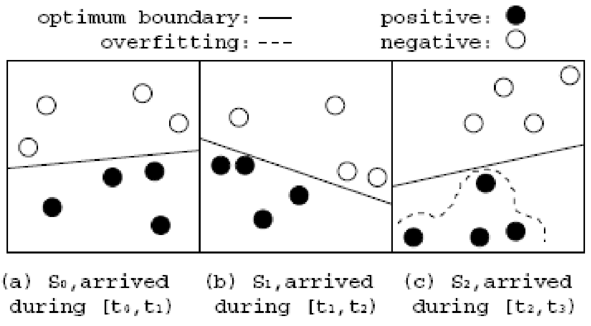
optimum boundary: ——
overfitting: ---
positive: ●
negative: ○
(a) S₀,arrived during [t₀,t₁)
(b) S₁,arrived during [t₁,t₂)
(c) S₂,arrived during [t₂,t₃)

Figure 1. Data Distribution and optimal boundaries

Figure 1 shows the general dilemma of the time-based window approach: Older concepts ($S_0$) – already out-of-scope – would fit the current concepts ($S_2$) quite well. Unfortunately, the knowledge about $S_0$ was removed as data slipped out of the time window when learning about $S_1$.

The general idea of the concept-window algorithm is that it is not *time* that decides which already learnt information should slip out-of-scope, but rather the *content* of the data stream itself: the more information is being gathered about certain features of the data stream, the more details can be learnt about this feature.

Other research developed an ensemble-approach to solve the problem of data expiration [2]. The use of a weighted ensemble of classifiers allows adding characteristics learnt from various different time-periods of the data stream, therefore combining the desired features for the current concept (see Figure 1). However, our strategy, although retaining the spirit of the ensemble approach, utilises growth and pruning strategies to capture concept drift over time without being restricted by arbitrary window-sizes.

Similar to the time-based window algorithms, one of the most important features of the algorithm is to decide which information should be forgotten, since the concept of a window inherently limits the available space for the storage of newly acquired information. In the time-based window approach, the decision about which information is to be erased is based purely on the age of the instances used to learn the respective feature. The proposed concept-based window approach, on the other hand, utilises usage-statistics to determine possible candidates for deletion. Ultimately, the sub-tree with the least *overall* loss of information can be removed in order to create room for a new node.

The main difference between concept-based and time-based algorithms is that the removed information is not determined by a function of time, *f(time)*, but rather by determining $f_{min}$*(information-loss)* of the all sub-trees considered for removal.

This also allows atypical behaviour in the data stream (e.g. bursts or just general noise) to be learnt more accurately in a concept-based approach than in a time-based approach.

Given an unlimited window-size for the time-based approach and unlimited storage space for the concept-based approach, both resulting trees should be identical.

One of the biggest strengths of the time-window approach is its simplicity and the relative ease with which already learnt instances can be removed from the current model. Here, the concept-based approach shows its weakness: Correct tree maintenance can introduce a manifold of potential complexities to the algorithm. Nevertheless, increased algorithmic complexity comes with an improved precision for re-occurring concepts, whereas the time-window approach only analyses snapshots, and forgets short and long-term concepts as they drift outside the time-window.

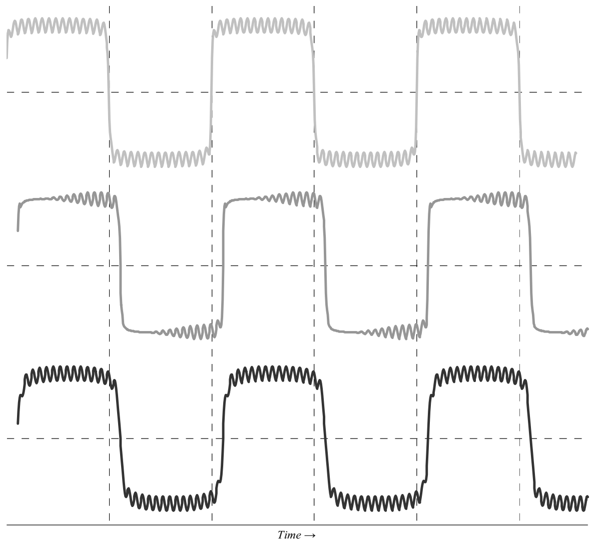

Figure 2. Concept-drift recognition: time-based vs. concept-based window approach

Figure 2 shows concept drifts in a data stream (top): A large drift with a long frequency and a super-imposed drift with a smaller frequency. A time-based window approach (middle) will have to re-learn the super-imposed drift each time the base concept changes. Using a concept-based approach (bottom), the algorithm still remembers the super-imposed concept even if the main concept changes drastically.

## V. Proposed Methodology

The introduction of the basic idea of a concept-based window allowed us track concept drift utilising $f_{min}$*(information-loss)*. In following sections we will define $f_{min}$*(information-loss)* in more detail and use it to restructure a decision tree by removing complete or partial leaf nodes before creating new leaves.

### *A. Node removal*

Our approach is based on the VFDT algorithm [3], adopted to deal with concept drift that recurs over a period of time. In contrast to CVFDT [4], we deliberately choose to grow sub-trees at the leaves only and not at each intermediate node. Furthermore, we do not evaluate potential alternative sub-trees for each node of the tree.

Both changes are necessary in order to gracefully adapt the tree's structure to new requirements. Consider the case where a newly arriving instance causes a leaf node to be split. When the available storage for new nodes is exhausted, our algorithm determines which of the current leaf nodes of the tree should be *recombined* (the algorithm needs to reverse the effects of a previously performed split) in order to minimise the overall loss of information for the tree. In order to perform this recombination correctly, and to decide which sub-tree to collapse, the algorithm utilises certain usage-statistics, which are maintained for all nodes and leaves in the tree.

***Definition 1: Decision path***
A decision path is defined as an ordered sequence of decision nodes (inclusive of the root node) an instance passes through until it reaches a leaf of the tree. A combination of root, decision nodes and leaf nodes comprise the decision path for an instance.

***Definition 2: Least-used decision path***
The least-used decision path (LDP) is that decision path that is least traversed.

The LDP of a tree can be shortened by reversing the split of the youngest decision node, therefore creating room for new leaves in a different part of the tree. This allows the algorithm to keep the tree within certain size boundaries, but still adjust to conceptual changes in the data stream by learning new information and discarding information with the least impact on the tree's error rate.

In order to determine the LDP, we need to introduce additional counters C (over and above those used by VFDT and CVFDT) at the root node, decision nodes and leaves. The counters at the root node and the affected leaf node are incremented each time a new instance is presented. In addition, every decision node needs to carry a copy of the value of CRoot at the time when the decision node was created. The LDP of the tree can now be calculated using:

$$\frac{C_{Leaf}}{C_{Root} - C_{DecisionNode} + 1} \cdot \overline{G}$$

Where $\overline{G}$ is the anticipated average information gain, achieved by the leaf in question after n instances have been presented

The first part of the above formula allows for calculating the *relative* usage of each leaf of the decision tree. Now, we can calculate the potential relative information loss for all leaves of each sub-tree, which in turn allows choosing the sub-tree with the least potential *average* information loss for removal. Since the average information loss of all leaves of a sub-tree is calculated, we can postulate that the quality of the removal decision will be highest with binary trees.

In order to be able to construct the *recombined* leaf from the siblings to be removed, we need to keep classification error statistics in each leaf, thus enabling the determination of the correct class assignment of the new leaf and reinstating required leaf statistics.

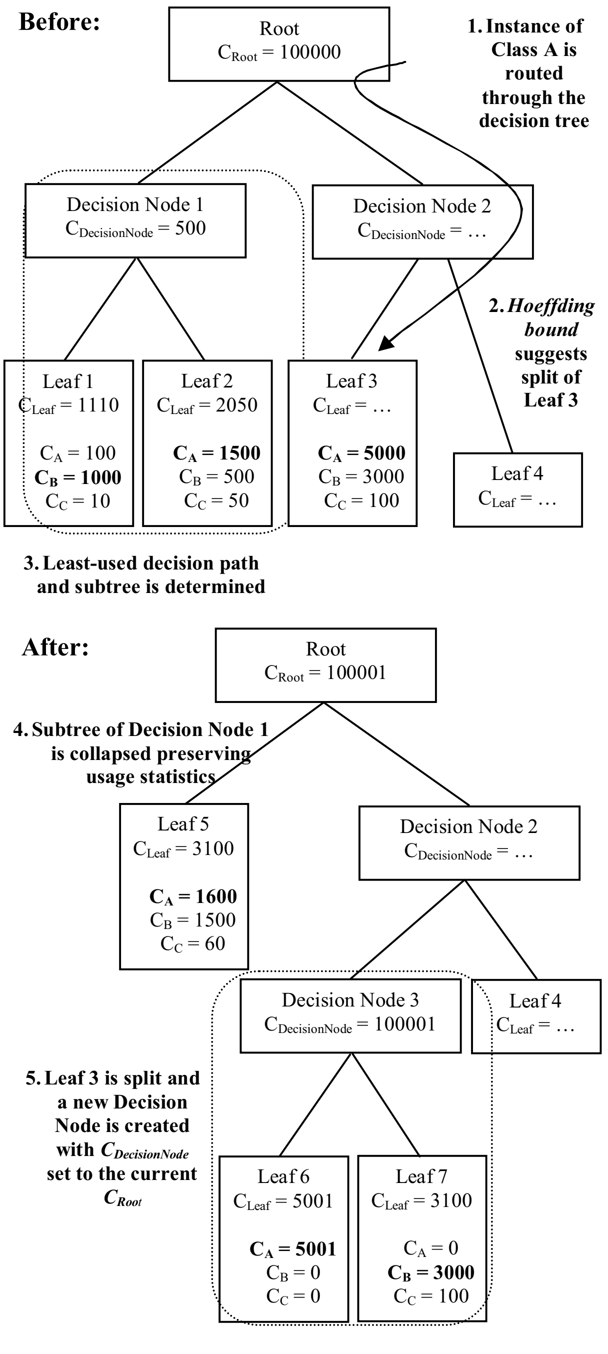

Figure 3. Before/after snapshots when a leaf node needs to be split

The snapshots in Figure 3 show the simplified behaviour of the proposed algorithm in case a leaf has to be split when processing an instance. The upper snapshot details the binary decision tree before the split; the lower snapshot shows the tree after the event.

## B. *Partial node removal*

This algorithm is based on the one outlined in section V A, but instead of immediately removing the relatively least-used siblings, all leaves are assigned a *removal-probability* – determined by the number of leaves, which are labelled with the same class value divided by the total number of leaves in the tree. The *removal-probability* at each leaf is updated according the expression below:

$$RP_{new} = RP_{old} + \frac{n_{L_E}}{\sum_{C'=C-E} n_{L_{C'}}}$$

| | | |
|---|---|---|
| $RP_{old/new}$ | ... | *Removal-probability* before/after the creation event |
| $E$ | ... | Instance causing the creation event is assigned to class value $E$ |
| $C$ | ... | Set of all class values currently occurring in the tree |
| $n_{L_X}$ | ... | Number of leaf nodes assigned to class value $X$ |

Over a period of time, several leaf creation requests accrue in each leaf (each time only in leaves of a different class than the instance that caused the creation of the decision node), and for each elimination round the algorithm can investigate which of the nodes above a certain elimination threshold should be removed by utilising $f_{min}$*(information-loss)*.

Since the above scenario means that leaf removal does not necessarily follow a leaf creation request, it implies that a buffer is built for growing the tree without removing leaves. In other words, we have a procedure that automatically adapts the elimination threshold to the expected memory requirements of the tree.

For the elimination round algorithm, several options can be envisioned:

An elimination round can automatically be triggered once the growth-buffer is full (or reached a certain size limit); it can be triggered each time after a certain number of creation requests, or it can be started by an idle process utilising spare capacities.

The algorithm can remove all leaf-siblings with a *removal-probability* above a certain threshold, or it can remove only the top *n* leaf-siblings (sorted by *removal-probability*) or use some other metrics to determine appropriate candidates for removal (e.g. adjust the *removal-probability* with the relative-usage formula in section V A).

After the actual elimination, the algorithm can reset the *removal-probability* of each leaf to allow equal start conditions or let the leaves carry the value of the *removal-probability* into the next round (or only part of the value adjusted by some factor).

## C. *Changes to ε*

Another idea to adapt VFDT/CVFDT to recognise concept drift sans time-windows is to allow ε to vary according to the *relative* number of instances passing through the tree for each class.

This, in turn, would allow the tree to grow more profusely and with more precision (by lowering ε in such cases) for classes with a high number of instances and to restrict growth (higher ε) for classes with only a few instances – maybe even restricting the differentiation between low-volume classes.

## VI. Conclusions

In this paper, we introduced the idea of the concept-based window and proposed several methodologies that can be used to implement an incremental high-speed decision tree learner. These methodologies allow the tree to be restructured by refining leaf attributes and by pruning leaves that are deemed unnecessary after concept change in the underlying data stream.

We believe that such concept-based approaches are better suited to capture certain types of conceptual changes, i.e. slowly evolving concepts, erratically re-occurring patterns, super-imposed harmonics, etc.

Future work includes the implementation of this algorithm and testing in order to verify and refine our concepts. Another issue that needs to be resolved is the conservative approach that our proposed methodology suggests: Changes to the tree's structure are only performed in a subtle, conservative manner. This can prevent the algorithm from reacting to sudden changes in the data stream's concept. An autonomous mechanism that can adapt the level of pruning to the requirements of the data stream or that can undertake localised structural changes to the tree needs to be studied.

## References

[1] S. Papadimitriou, A. Brockwell, and C. Faloutsos, "Adaptive, unsupervised stream mining," *The VLDB Journal,* vol. 13, pp. 222-239, 2004.

[2] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 226-235, 2003.

[3] P. Domingos and G. Hulten, "Mining high-speed data streams," *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 71-80, 2000.

[4] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 97-106, 2001.

[5] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "On demand classification of data streams," *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 503-508, 2004.

[6] G. Mohamed Medhat, Z. Arkady, and K. Shonali, "Mining data streams: a review," *SIGMOD Rec.,* vol. 34, pp. 18-26, 2005.

[7] G. Dong, J. Han, L. V. S. Lakshmanan, J. Pei, H. Wang, and P. S. Yu, "Online Mining of Changes from Data Streams: Research Problems and Preliminary Results," *Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams,* 2003.

[8] V. Guralnik and J. Srivastava, "Event detection from time series data," *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 33-42, 1999.

[9] S. Guah, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams: Theory and Practice," *TKDE Splecial Issue on Clustering,* vol. 15, 2003.

[10] N. Kasabov, *Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines*. London: Springer-Verlag, 2003.

[11] E. Keogh and J. Lin, "Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research," *Proceedings of the 3rd IEEE International Conference on Data Mining,* pp. 19-22, 2003.

[12] Y. D. Cai, D. Cliutter, G. Pape, J. Han, and M. Welge, "MAIDS: Mining Alarming Incidents from Data Streams," *Proceedings of the 23rd ACM SIGMOD International conference on Management of Data,* 2004.

[13] G. Hulten and P. Domingos, "Mining complex models from arbitrarily large databases in constant time," *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 525-531, 2002.

[14] V. Ganti, J. Gehrke, and R. Ramakrishnan, "Mining data streams under block evolution," *SIGKDD Explorations Newsletter,* vol. 3, pp. 1-10, 2002.

[15] M. Last, "Online Classification of Nonstationary Data Streams," *Intelligent Data Analysis,* vol. 6, pp. 129-147, 2002.

[16] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," *Proceedings of the 28th VLDB Conference,* pp. 358-369, 2002.