An Incremental Fuzzy Decision Tree Classification Method for Mining Data Streams*

Tao Wang[1], Zhoujun Li[2], Yuejin Yan[1], and Huowang Chen[1]

[1] Computer School, National University of Defense Technology, Changsha, 410073, China
[2] School of Computer Science & Engineering, Beihang University, Beijing, 100083, China
InsistStar@nudt.edu.cn

Abstract. One of most important algorithms for mining data streams is VFDT. It uses Hoeffding inequality to achieve a probabilistic bound on the accuracy of the tree constructed. Gama et al. have extended VFDT in two directions. Their system VFDTc can deal with continuous data and use more powerful classification techniques at tree leaves. In this paper, we revisit this problem and implemented a system fVFDT on top of VFDT and VFDTc. We make the following four contributions: 1) we present a threaded binary search trees (TBST) approach for efficiently handling continuous attributes. It builds a threaded binary search tree, and its processing time for values inserting is *O(nlogn)*, while VFDT`s processing time is $O(n^2)$. When a new example arrives, VFDTc need update *O(logn)* attribute tree nodes, but fVFDT just need update one necessary node.2) we improve the method of getting the best split-test point of a given continuous attribute. Comparing to the method used in VFDTc, it improves from *O(nlogn)* to *O (n)* in processing time. 3) Comparing to VFDTc, fVFDT`s candidate split-test number decrease from *O(n)* to *O(logn)*.4)Improve the soft discretization method to be used in data streams mining, it overcomes the problem of noise data and improve the classification accuracy.

Keywords: Data Streams, Incremental, Fuzzy, Continuous Attribute, Threaded Binary Search Tree.

1 Introduction

Decision trees are one of the most used classification techniques for data mining. Tree models have high degree of interpretability. Global and complex decisions can be approximated by a series of simpler and local decisions. Algorithms that construct decision trees from data usually use a divide and conquer strategy. A complex problem is divided into simpler problems and recursively the same strategy is applied to the sub-problems. The solutions of sub-problems are combined in the form of a tree to yield the solution of the complex problem [3, 20, 22].

* This work was supported by the National Science Foundation of China under Grants No. 60573057, 60473057 and 90604007.

P. Perner (Ed.): MLDM 2007, LNAI 4571, pp. 91–103, 2007.

More recently, the data mining community has focused on a new model of data processing, in which data arrives in the form of continuous streams [1, 3, 9, 11, 12, 16, 28, 29]. The key issue in mining on data streams is that only one pass is allowed over the entire data. Moreover, there is a real-time constraint, i.e. the processing time is limited by the rate of arrival of instances in the data stream, and the memory and disk available to store any summary information may be bounded. For most data mining problems, a one-pass algorithm cannot be very accurate. The existing algorithms typically achieve either a deterministic bound on the accuracy or a probabilistic bound [21, 23].

Domingos and Hulten [2, 6] have addressed the problem of decision tree construction on data streams. Their algorithm guarantees a probabilistic bound on the accuracy of the decision tree that is constructed. Gama et al. [5] have extended VFDT in two directions: the ability to deal with continuous data and the use of more powerful classification techniques at tree leaves.

Peng et al.[30]propose the soft discretization method in traditional data mining field,it solve the problem of noise data and improve the classification accuracy.

The rest of the paper is organized as follows. Section 2 describes the related works that is the basis for this paper. Section 3 presents the technical details of fVFDT. The system has been implemented and evaluated, and experimental evaluation is done in Section 4. Last section concludes the paper, resuming the main contributions of this work.

2 Related Work

In this section we analyze the related works that our fVFDT bases on.

Decision trees support continuous attributes by allowing internal nodes to contain tests of the form $A_i \leq T$ (the value of attribute i is less than threshold T). Traditional induction algorithms learn decision trees with such tests in the following manner. For each continuous attribute, they construct a set of candidate tests by sorting the values of that attribute in the training set and using a threshold midway between each adjacent pair of values that come from training examples with different class labels to get the best split-test point.

There are several reasons why this standard method is not appropriate when learning from data streams. The most serious of these is that it requires that the entire training set be available ahead of time so that split thresholds can be determined.

2.1 VFDT

VFDT(Very Fast Decision Tree) system[2], which is able to learn from abundant data within practical time and memory constraints. In VFDT a decision tree is learned by recursively replacing leaves with decision nodes. Each leaf stores the sufficient statistics about attribute-values. The sufficient statistics are those needed by a heuristic evaluation function that evaluates the merit of split-tests based on attribute-values. When an example is available, it traverses the tree from the root to a leaf, evaluating the

appropriate attribute at each node, and following the branch corresponding to the attribute's value in the example. When the example reaches a leaf, the sufficient statistics are updated. Then, each possible condition based on attribute-values is evaluated. If there is enough statistical support in favor of one test over the others, the leaf is changed to a decision node. The new decision node will have as many descendant leaves as the number of possible values for the chosen attribute (therefore this tree is not necessarily binary). The decision nodes only maintain the information about the split-test installed in this node. The initial state of the tree consists of a single leaf: the root of the tree. The heuristic evaluation function is the Information Gain (denoted by G(.). The sufficient statistics for estimating the merit of a discrete attribute are the counts nijk, representing the number of examples of class k that reach the leaf, where the attribute j takes the value i. The Information Gain measures the amount of information that is necessary to classify an example that reach the node: G(Aj)=info(examples)-info(Aj). The information of the attribute j is given by:

$$\inf o(A_j) = \sum_i P_i(\sum_k -P_{ik} \log(P_{ik}))$$

where $P_{ik} = n_{ijk} \Big/ \sum_a n_{ajk}$, is the probability of observing the value of the attribute *i* given class *k* and $P_i = \sum_a n_{ija} \Big/ \sum_a \sum_b n_{ajb}$ is the probabilities of observing the value of attribute *i*.

As mentioned in Catlett and others [23], that it may be sufficient to use a small sample of the available examples when choosing the split attribute at any given node. To determine the number of examples needed for each decision, VFDT uses a statistical result known as Hoeffding bounds or additive Chernoff bounds. After *n* independent observations of a real-valued random variable *r* with range *R,* the Hoeffding_bound ensures that, with confidence *1-δ*, the true mean of *r* is at least $r - \varepsilon$, where r is the observed mean of samples and $\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$. This is true irrespective of the probability distribution that generated the observations.

Let G(.) be the evaluation function of an attribute. For the information gain, the range *R*, of G(.) is $log_2 \#classes$. Let x_a be the attribute with the highest G(.), x_b the attribute with second-highest G(.) and $\Delta\overline{G} = \overline{G}(x_a) - \overline{G}(x_b)$, the difference between the two better attributes. Then if $\Delta G > \varepsilon$ with *n* examples observed in the leaf, the Hoeffding bound states with probability *1-δ* that x_a is really the attribute with highest value in the evaluation function. In this case the leaf must be transformed into a decision node that splits on x_a.

For continuous attribute, whenever VFDT starts a new leaf, it collects up to *M* distinct values for each continuous attribute from the first examples that arrive at it. These are maintained in sorted order as they arrive, and a candidate test threshold is maintained midway between adjacent values with different classes, as in the traditional method. Once VFDT has *M* values for an attribute, it stops adding new candidate

thresholds and uses additional data only to evaluate the existing ones. Every leaf uses a different value of *M*, based on its level in the tree and the amount of RAM available when it is started. For example, *M* can be very large when choosing the split for the root of the tree, but must be very small once there is a large partially induced tree, and many leaves are competing for limited memory resources. Notice that even when *M* is very large (and especially when it is small) VFDT may miss the locally optimal split point. This is not a serious problem here for two reasons. First, if data is an independent, identically distributed sample, VFDT should end up with a value near (or an empirical gain close to) the correct one simply by chance. And second, VFDT will be learning very large trees from massive data streams and can correct early mistakes later in the learning process by adding additional splits to the tree.

Thinking of each continuous attribute, we will find that the processing time for the insertion of new examples is $O(n^2)$, where n represents the number of distinct values for the attribute seen so far.

2.2 VFDTc

VFDTc is implemented on top of VFDT, and it extends VFDT in two directions: the ability to deal with continuous attributes and the use of more powerful classification techniques at tree leaves. Here, we just focus on the handling of continuous attributes.

In VFDTc a decision node that contains a split-test based on a continuous attribute has two descendant branches. The split-test is a condition of the form $attrib_j \leq T$. The two descendant branches correspond to the values *TRUE* and *FALSE* for the split-test. The cut point is chosen from all the possible observed values for that attribute. In order to evaluate the goodness of a split, it needs to compute the class distribution of the examples at which the attribute-value is less than or greater than the cut point. The counts n_{ijk} are fundamental for computing all necessary statistics. They are kept with the use of the following data structure: In each leaf of the decision tree it maintains a vector of the classes' distribution of the examples that reach this leaf. For each continuous attribute *j*, the system maintains a binary attribute tree structure. A node in the binary tree is identified with a value *i*(that is the value of the attribute *j* seen in an example), and two vectors (of dimension *k*) used to count the values that go through that node. Two vectors, *VE* and *VH* contain the counts of values respectively $\leq i$ and $> i$ for the examples labeled with class *k*. When an example reaches leaf, all the binary trees are updated. In [5], an algorithm of inserting a value in the binary tree is presented. Insertion of a new value in this structure is *O(nlogn)* where *n* represents the number of distinct values for the attribute seen so far.

To obtain the Information Gain of a given attribute, VFDTc uses an exhaustive method to evaluate the merit of all possible cut points. Here, any value observed in the examples seen so far can be used as cut point. For each possible cut point, the information of the two partitions is computed using equation 1.

$$\inf o(A_j(i)) = P(A_j \leq i) \ * \ iLow(A_j(i)) + P(A_j > i) \ * i \ High(A_j(i)) \quad (1)$$

Where i is the cut point, $iLow(A_j(i))$ the information of $A_j \leqslant i$ (equation 2) and $iHigh(A_j(i))$ the information of $A_j > i$ (equation 3).

$$iLow(A_j(i)) = -\sum_K P(K = k \mid A_j \leq i) * \log(P(K=k|A_j \leq i)) \quad (2)$$

$$iHigh(A_j(i)) = -\sum_K P(K = k \mid A_j > i) * \log(P(K=k|A_j > i)) \quad (3)$$

VFDTc only considers a possible *cut_point* if and only if the number of examples in each of subsets is higher than Pmin (a user defined constant) percentage of the total number of examples seen in the node. [5] Presents the algorithm to compute #(Aj≤ i) for a given attribute j and class k. The algorithm's processing time is O(logn), so the best split-test point calculating time is O(nlogn). Here, n represents the number of distinct values for the attribute seen so far at that leaf.

2.3 Soft Discretization

Soft discretization could be viewed as an extension of hard discretization, and the classical information measures defined in the probability domain have been extended to new definitions in the possibility domain based on fuzzy set theory [13]. A crisp set A_c is expressed with a sharp characterization function $A_c(a): \Omega \to \{0,1\}: a \in \Omega$, alternatively a fuzzy set A is characterized with a membership function $A(a): \Omega \to [0,1]: a \in \Omega$. The membership $A(a)$ is called the possibility of A to take a value $a \in \Omega$ [14]. The probability of fuzzy set A is defined, according to Zadeh [15], by $P_F(A) = \int_\Omega A(a)dP$, where dP is a probability measure on Ω, and the subscript F is used to denote the associated fuzzy terms. Specially, if A is defined on discrete domain $\Omega = \{a_1, ..., a_i, ..., a_m\}$, and the probability of $P(a_i) = p_i$ then its probability is $P_F(A) = \sum_{i=1}^{m} A(a_i) p_i$.

Let $Q = \{A_1, ..., A_k\}$ be a family of fuzzy sets on Ω. Q is called a fuzzy partition of Ω [16] when $\sum_{r=1}^{k} A_r(a) = 1, \forall a \in \Omega$.

A hard discretization is defined with a threshold *T*, which generates the boundary between two crisp sets. Alternatively, a soft discretization is defined by a fuzzy set pair which forms a fuzzy partition. In contrast to the classical method of non-overlapping partitioning, the soft discretization is overlapped. The soft discretization is defined with three parameters/functions, one is the cross point *T*, the other two are the membership functions of the fuzzy set pair A_1 and A_2: $A_1(a)+A_2(a)=1$. The cross point *T*, i.e. the localization of soft discretization, is determined based on whether it can maximize the

information gain in classification, and the membership functions of the fuzzy set pair are determined according to the characteristics of attribute data, such as the uncertainty of the associated attribute.

3 Technique Details

Improving soft discretizaiont method, we implement a system named fVFDT on top of VFDT and VFDTc. It handles continuous attributes based on threaded binary search trees, and uses a more efficient best split-test point calculating method.

For discrete attributes, they are processed using the algorithm mentioned in VFDT [2]. Our fVFDT specially focus on continuous attribute handling.

3.1 Threaded Binary Search Tree Structure for Continuous Attributes

fVFDT maintains a threaded binary search tree for each continuous attribute. The threaded binary search tree data structure will benefit the procedure of inserting new example and calculating best split-test point.

For each continuous attribute *i*, the system maintains a threaded binary search tree structure. A node in the threaded binary search tree is identified with a value *keyValue* (that is the value of the attribute *i* seen in the example)， and a vector(of dimension *k*) used to count the values that go through that node. This vector *classTotals[k]* contains the counts of examples which value is *keyValue* and class labeled with *k*. A node manages *left* and *right* pointers for its left and right child, where its left child corresponds to $\leqslant keyValue$, while its right child corresponds to >*keyValue*. For the goodness of calculating the best split-test point, a node contains *prev* and *next* pointers for the previous and next node. At most, three nodes` *prev* and *next* pointers will be updated while new example arrives.

fVFDT maintains a *head* pointer for each continuous attribute to traverse all the threaded binary trees.

3.2 Updates the Threaded Search Binary Tree While New Examples Arrives

One of the key problems in decision tree construction on streaming data is that the memory and computational cost of storing and processing the information required to obtain the best split-test point can be very high. For discrete attributes, the number of distinct values is typically small, and therefore, the class histogram does not require much memory. Similarly, searching for the best split predicate is not expensive if number of candidate split conditions is relatively small.

However, for continuous attributes with a large number of distinct values, both memory and computational costs can be very high. Many of the existing approaches are scalable, but they are multi-pass. Decision tree construction requires a preprocessing phase in which attribute value lists for continuous attributes are sorted [20]. Preprocessing of data, in comparison, is not an option with streaming data, and sorting during execution can be very expensive. Domingos and Hulten have described and

evaluated their one-pass algorithm focusing only on discrete attributes [2], and in later version they uses sorted array to handle continuous attribute. This implies a very high memory and computational overhead for inserting new examples and determining the best split point for a continuous attribute.

In fVFDT a Hoeffding tree node manages a threaded binary search tree for each continuous attribute before it becomes a decision node.

```
Procedure InsertValueTBSTree(x, k, TBSTree)
  Begin
   while (TBSTree ->right != NULL || TBSTree ->left != NULL )
        if (TBSTree ->keyValue = = x )      then   break;
        Elseif (TBSTree ->keyValue > x )   then TBSTree = TBSTree ->left;
        else TBSTree = TBSTree ->right;
   Creates a new node curr based on x and k;
   If ( TBSTree.keyValue = = x )      then     TBSTree.classTotals[k]++;
   Elesif (TBSTree.keyValue > x)     then     TBSTree.left = curr;
   else                TBSTree.right = curr;
   Threads the tree ;( The details of threading is in figure2)
  End
```

Fig. 1. Algorithm to insert value x of an example labeled with class k into a threaded binary search tree corresponding to the continuous attribute i

In the induction of decision trees from continuous-valued data, a suitable threshold T, which discretizes the continuous attribute i into two intervals: $atrr_i \leq T$ and $atrr_i > T$, is determined based on the classification information gain generated by the corresponding discretization. Given a threshold, the test $atrr_i \leq T$ is assigned to the left branch of the decision node while $atrr_i > T$ is assigned to the right branch. As a new example (x,k) arrives, the threaded binary search tree corresponding to the continuous attribute i is update as Figure 1.

In [5], when a new example arrives, $O(logn)$ binary search tree nodes need be updated, but fVFDT just need update a necessary node here. VFDT will cost $O(n^2)$, and our system fVFDT will just cost $O(nlogn)$ (as presented in Figure 1) in execution time for values inserting, where n represents the number of distinct values for the given attribute seen so far.

3.3 Threads the Binary Tree While New Example Arrives

fVFDT need thread the binary search trees while new example arrives. If the new example's value is equal to an existing node`s value, the threaded binary tree doesn't need be threaded. Otherwise, the threaded binary tree need be threaded as Figure 2.

At most, three relevant nodes need be updated here. This threading procedure mentioned in Figure 2 can be embedded in the procedure presented in Figure 1, and the inserting procedure's processing time is still $O(nlogn)$.

```
Procedure TBSTthreads()
Begin
if (new node curr is left child of ptr)
        curr->next = ptr;
        curr->nextValue = ptr->keyValue;
        curr->prev = ptr->prev;
        ptr->prev->next = curr;
        prevPtr->nextValue = value;
        ptr->prev = curr;
        if (new node curr is right child of ptr)
            curr->next = ptr->next;
            curr->nextValue = ptr->nextValue;
            curr->prev = ptr;
            ptr->next->prev = curr;
            ptr->nextValue = value;
            ptr->next = curr;
 End
```

Fig. 2. Algorithm to thread the binary search tree while new example arrives

3.4 Soft Discretization of Continuous Attributes

Taking advantage of threaded binary search tree, we use a more efficient method to obtain the fuzzy information gain of a given attribute.

Assuming we are to select an attribute for a node having a set S of N examples arrived, these examples are managed by a threaded binary tree according to the values of the continuous attribute i; and an ordered sequence of distinct values a_1, a_2 … a_n is formed. Every pair of adjacent data points suggests a potential threshold $T= (a_i+a_{i+1})/2$ to create a cut point and generate a corresponding partition of attribute *i*. In order to calculate the goodness of a split, we need to compute the class distribution of the examples at which the attribute value is less than or greater than threshold *T*. The counts *TBSTree.classTotals[k]* are fundamental for computing all necessary statistics.

To take the advantage of threaded binary search tree, we record the *head* pointer of each attribute's threaded binary search tree. As presented in Figure 3, traversing from the *head* pointer to the tail pointer, we can easily compute the fuzzy information of all the potential thresholds. fVFDT implies soft discretization by managing Max/Min value and example numbers.

```
Procedure BSTInorderAttributeSplit(TBSTtreePtr ptr,int *belowPrev[])
   Begin
      if ( ptr->next == NULL) then    break;
      for ( k = 0 ; k < count ; k++)
            *belowPrev[k] += ptr->classTotals[k];
    Calculates the information gain using *belowPrev[];
      BSTInorderAttributeSplit( ptr->next,int *belowPrev[]);
   End
```

Fig. 3. Algorithm to compute the information gain of a continuous attribute

Here, VFDTc will cost *O(nlogn)* , and our system fVFDT will just cost *O(n)* in processing time, where *n* represents the number of distinct values for the given continuous attribute seen so far.

3.5 Classify a New Example

The classification for a given unknown object is obtained from the matching degrees of the object to each node from root to leaf. The possibility of an object belonging to class C_i is calculated by a fuzzy product operation $\otimes$. In the same way, the possibility of the object belonging to each class can be calculated, $\{\prod_i\}_{i=1...k}$. If more than one leaf are associated with a same class C_i, say, the value of $\prod_i = \oplus(\prod_j)$ will be considered as the possibility of the corresponding class, where the maximum operation is used as the fuzzy sum operation $\oplus$ In the end, if one possibility value, such as $\prod_k$, is much higher than others, that is $\prod_k >> \prod_{i...k}$, then the class will be assigned as the class of the object, otherwise the decision tree predicts a distribution over all the classes.

4 Evaluation

In this section we empirically evaluate fVFDT. The main goal of this section is to provide evidence that the use of threaded binary search tree decreases the processing time of VFDT, while keeps the same error rate and tree size. The algorithms` processing time is listed in Table 1.

Table 1. Algorithm's processing time

Algorithm Name	Inserting time	Best split-test point calculating time
VFDT	$O(n^2)$	$O(n)$
VFDTc	$O(n\log n)$	$O(n\log n)$
fVFDT	$O(n\log n)$	$O(n)$

We first describe the data streams used for our experiments. We use a tool named *treeData* mentioned in [2] to create synthetic data .It creates a synthetic data set by sampling from a randomly generated decision tree. They were created by randomly generating decision trees and then using these trees to assign classes to randomly generated examples. It produced the random decision trees as follows. Starting from a tree with a single leaf node (the root) it repeatedly replaced leaf nodes with nodes that tested a randomly selected attribute which had not yet been tested on the path from the root of the tree to that selected leaf. After the first three levels of the tree each selected leaf had a probability of f of being pre-pruned instead of replaced by a split (and thus of remaining a leaf in the final tree). Additionally, any branch that reached a depth of 18 was pruned at that depth. Whenever a leaf was pruned it was randomly (with uniform probability) assigned a class label. A tree was completed as soon as all of its leaves were pruned.

VFDTc`s goal is to show that using stronger classification strategies at tree leaves will improve classifier's performance. With respect to the processing time, the use of naïve Bayes classifier will introduce an overhead [5], VFDTc is slower than VFDT. In order to compare the VFDTc and fVFDT , we implement the continuous attributes solving part of VFDTc ourselves.

We ran our experiments on a Pentium IV/2GH machine with 512MB of RAM, which running Linux RedHat 9.0.

Table 2 shows the processing (excluding I/O) time of learners as a function of the number of training examples averaged over nine runs. VFDT and fVFDT run with parameters $\delta = 10^{-7}, \tau = 5\%, n_{\min} = 300, example\ number = 100000K$, no leaf reactivation, and no rescan. Averagely, comparing to VFDT, fVFDT`s average reduction of processing time is 16.66%, and comparing to VFDTc, fVFDT`s average reduction is 6.25%.

Table 2. The comparing result of processing time

time(seconds) / example numbers	VFDT	VFDTc	fVFDT
10000	4.66	4.21	3.75
20736	9.96	8.83	8.12
42996	22.88	20.59	18.57
89156	48.51	43.57	40.87
184872	103.61	93.25	87.12
383349	215.83	187.77	175.23
794911	522.69	475.65	441.61
1648326	1123.51	1022.39	939.35
3417968	2090.31	1839.45	1758.89
7087498	3392.94	3053.65	2882.23
14696636	5209.47	4688.53	4389.35
30474845	8203.05	7382.75	6850.12
43883922	13431.02	11953.61	11068.23
90997707	17593.46	15834.12	15020.46
100000000	18902.06	16822.86	15986.23

In this work, we measure the size of tree models as the number of decision nodes plus the number of leaves. As for dynamic data stream with 100 million examples, we

notice that the two learners similarly have the same tree size. We have done another experiment using 1 million examples generated on disk, and the result shows that they have same tree size.

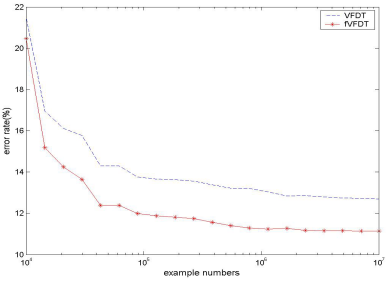

Fig. 4. Error rate as a function of the examples numbers

Figure 4 shows the error rate curves of VFDT and fVFDT. Both algorithms have 10% noise data, VFDT`s error rate trends to 12.5%, while the fVFDT`s error rate trends to 8%. Experiment results show that fVFDT get better accuracy by using soft discretization, and it overcomes the problem of noise.

5 Conclusions and Future Work

On top of VFDT and VFDTc, improve the soft discretization method, we propose a system fVFDT. Focusing on continuous attribute, we have developed and evaluated a new technique named TBST to insert new example and calculate best split-test point efficiently. It builds threaded binary search trees, and its processing time for values insertion is *O(nlogn)*. Comparing to the method used in VFDTc, it improves from *O(nlogn)* to *O(n)* in processing time for best split-test point calculating. As for noise data, we improve the soft discretization method in traditional data mining field, so the fVFDT can deal with noise data efficiently and improve the classification accuracy.

In the future, we would like to expand our work in some directions. First, we do not discuss the problem of time changing concept here, and we will apply our method to those strategies that take into account concept drift [4, 6, 10, 14, 15, 19, 24, 25]. Second, we want to apply other new fuzzy decision tree methods in data streams classification [8, 13, 17, 18, 26].

References

[1] Babcock, B., Babu, S., Datar, M., Motawani, R., Widom, J.: Models and Issues in Data Stream Systems. In: PODS (2002)

[2] Domingos, P., Hulten, G.: Mining High-Speed Data Streams. In: Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining, pp. 71–80 (2000)

[3] Mehta, M., Agrawal, A., Rissanen, J.: SLIQ: A Fast Scalable Classifier for Data Mining. In: Proceedings of The Fifth International Conference on Extending Database Technology, Avignon, France, pp. 18–32 (1996)
[4] Fan, W.: StreamMiner: A Classifier Ensemble-based Engine to Mine Concept Drifting Data Streams, VLDB'2004 (2004)
[5] Gama, J., Rocha, R., Medas, P.: Accurate Decision Trees for Mining High-Speed Data Streams. In: Domingos, P., Faloutsos, C. (eds.) Proceedings of the Ninth International Conference on Knowledge Discovery and Data Mining, ACM Press, New York (2003)
[6] Hulten, G., Spencer, L., Domingos, P.: Mining Time-Changing Data Streams, ACM SIGKDD (2001)
[7] Jin, R., Agrawal, G.: Efficient Decision Tree Construction on Streaming Data. In: proceedings of ACM SIGKDD (2003)
[8] Last, M.: Online Classification of Nonstationary Data Streams. Intelligent Data Analysis 6(2), 129–147 (2002)
[9] Muthukrishnan, S.: Data streams: Algorithms and Applications. In: Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms (2003)
[10] Wang, H., Fan, W., Yu, P., Han, J.: Mining Concept-Drifting Data Streams using Ensemble Classifiers. In: the 9th ACM International Conference on Knowledge Discovery and Data Mining, Washington DC, USA. SIGKDD (2003)
[11] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: STREAM: The Stanford Stream Data Manager Demonstration Description –Short Overview of System Status and Plans. In: Proc. of the ACM Intl Conf. on Management of Data (SIGMOD 2003) (June 2003)
[12] Aggarwal, C., Han, J., Wang, J., Yu, P.S.: On Demand Classification of Data Streams. In: Proc. 2004 Int. Conf. on Knowledge Discovery and Data Mining (KDD'04), Seattle, WA (2004)
[13] Guetova, M., Holldobter, Storr, H.-P.: Incremental Fuzzy Decision Trees. In: 25th German conference on Artificial Intelligence (2002)
[14] Ben-David, S., Gehrke, J., Kifer, D.: Detecting Change in Data Streams. In: Proceedings of VLDB (2004)
[15] Aggarwal, C.: A Framework for Diagnosing Changes in Evolving Data Streams. In: Proceedings of the ACM SIGMOD Conference (2003)
[16] Gaber, M.M., Zaslavskey, A., Krishnaswamy, S.: Mining Data Streams: a Review, SIGMOD Record, vol. 34(2) (June 2005)
[17] Cezary, Janikow, Z.: Fuzzy Decision Trees: Issues and Methods. IEEE Transactions on Systems, Man, and Cybernetics 28(1), 1–14 (1998)
[18] Utgoff, P.E.: Incremental Induction of Decision Trees. Machine Learning 4(2), 161–186 (1989)
[19] Xie, Q.H.: An Efficient Approach for Mining Concept-Drifting Data Streams, Master Thesis
[20] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)
[21] Hoeffding, W.: Probability Inequalities for Sums of Bounded Random Variables. Journal of the American Statistical Association 58, 13–30 (1963)
[22] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees, Wadsworth, Belmont, CA (1984)
[23] Maron, O., Moore, A.: Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing System (1994)

[24] Kelly, M.G., Hand, D.J., Adams, N.M.: The Impact of Changing Populations on Classifier Performance. In: Proc. of KDD-99, pp. 367–371 (1999)
[25] Black, M., Hickey, R.J.: Maintaining the Performance of a Learned Classifier under Concept Drift. Intelligent Data Analysis 3, 453–474 (1999)
[26] Maimon, O., Last, M.: Knowledge Discovery and Data Mining, the Info-Fuzzy Network(IFN) Methodology. Kluwer Academic Publishers, Boston (2000)
[27] Fayyad, U.M., Irani, K.B.: On the Handling of Continuous-valued Attributes in Decision Tree Generation. Machine Learning 8, 87–102 (1992)
[28] Wang, T., Li, Z., Yan, Y., Chen, H.: An Efficient Classification System Based on Binary Search Trees for Data Streams Mining, ICONS (2007)
[29] Wang, T., Li, Z., Hu, X., Yan, Y., Chen, H.: A New Decision Tree Classification Method for Mining High-Speed Data Streams Based on Threaded Binary Search Trees, PAKDD (2007) workshop (2007)
[30] Peng, Y.H., Flach, P.A.: Soft Discretization to Enhance the Continuous Decision Tree Induction. In: Proceedings of ECML/PKDD-2001 Workshop IDDM-2001, Freiburg, Germany (2001)