Hu XG, Li PP, Wu XD *et al.* A semi-random multiple decision-tree algorithm for mining data streams. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 22(5): 711~724 Sept. 2007

A Semi-Random Multiple Decision-Tree Algorithm for Mining Data Streams

Xue-Gang Hu[1] (胡学钢), Pei-Pei Li[1] (李培培), Xin-Dong Wu[1,2] (吴信东), and Gong-Qing Wu[1] (吴共庆)

[1] *School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China*

[2] *Department of Computer Science, University of Vermont, Burlington, VT 50405, U.S.A.*

E-mail: jsjxhuxg@hfut.edu.cn; peipeili_hfut@163.com; xwu@cs.uvm.edu; wugongqing@gmail.com

Received January 15, 2007; revised July 16, 2007.

Abstract Mining with streaming data is a hot topic in data mining. When performing classification on data streams, traditional classification algorithms based on decision trees, such as ID3 and C4.5, have a relatively poor efficiency in both time and space due to the characteristics of streaming data. There are some advantages in time and space when using random decision trees. An incremental algorithm for mining data streams, SRMTDS (Semi-Random Multiple decision Trees for Data Streams), based on random decision trees is proposed in this paper. SRMTDS uses the inequality of Hoeffding bounds to choose the minimum number of split-examples, a heuristic method to compute the information gain for obtaining the split thresholds of numerical attributes, and a Naïve Bayes classifier to estimate the class labels of tree leaves. Our extensive experimental study shows that SRMTDS has an improved performance in time, space, accuracy and the anti-noise capability in comparison with VFDTc, a state-of-the-art decision-tree algorithm for classifying data streams.

Keywords data streams, Naïve Bayes, random decision trees

1 Introduction

With the rapid development and broad applications of information technologies, streaming data have become universal, such as supermarket transactions, Internet search requests, telephone call records, data from satellites and astronomy etc. Because streaming data are continuous, high-volume and open-ended, traditional mining algorithms cannot mine databases from these data environments in real-time, which lead to the loss of useful information. Also, storing the entire data from such streaming environments is difficult (if not impossible), hence efficient mining of data streams has become a popular research topic in data mining. Some mining algorithms of data streams have been proposed in the literature, such as the VFDT[1] classification algorithm and its extension VFDTc[2] based on decision trees, Peano count trees for spatial data streams[3], clustering with data streams[4], streaming[5] based on K-kedians, STREAM[6] based on K-means, the K-means variants[7] to cluster binary data streams, CluStreams[8] for clustering large evolving data streams, SHStream[9] of subspace clustering over high dimensional data streams, AcluStream for arbitrary shapes in data streams[10], FP_STREAM[11] and MGRDS[12] for mining frequent items, INSTANT[13] for maximal frequent item sequences, and MILE[14] for mining multiple sequence patterns.

Classification is one of the most important branches of data mining, but it is a challenge to perform classification in data streams with traditional classification models. The VFDT algorithm based on decision trees designed in [1] and its recent successor, VFDTc developed in [2], are two successful classification algorithms for mining data streams. In comparison with ITI[15] and ID5R[16] which use a greedy search to incorporate new data for reconstructing decision trees in an incremental way, VFDT and VFDTc maintain a set of sufficient statistics at each decision node and install a split-test at that node when there is enough statistical evidence in favor of it. VFDT only works with discrete attributes, and VFDTc extends it to process numerical attributes and applies a Naïve Bayes classifier in tree leaves. Both algorithms reinforce the any-time characteristic, but are weak in the anti-noise capability and have a high memory cost. They are not suitable for data streams with high dimensions or noisy data.

The Random Decision Tree (RDT)[17] model is an ensemble of decision trees. The decision-tree construction process in RDT chooses the split attributes

Regular Paper

This research is supported by the National Natural Science Foundation of China (Grant No. 60573174) and the Natural Science Foundation of Anhui Province of China (Grant No. 050420207).

of nodes and the split thresholds of numerical attributes randomly, which is unrelated to the training database. RDT only needs to scan the training database once, which has the characteristics of a lower space cost and a strong anti-noise capability, but needs to know the value ranges of numerical attributes in advance in order to process them in a simple way that does not have to fit with the given data streams. To overcome these limitations, the incremental algorithm SRMTDS (Semi-Random Multiple decision Trees for Data Streams) presented in this paper uses the inequality of Hoeffding bounds[18,19] and a heuristic method to determine the split thresholds of nodes with numerical attributes, deals with the numerical attributes efficiently without the necessity of knowing the value ranges of them in advance, and uses Naïve Bayes classifiers at tree leaves. The adopted semi-random strategy improves the classification accuracy of one single tree under the condition of ensuring the robust ability of resilience to noise, and the height of each semi-random decision tree is adjusted dynamically with certain conditions that adapt to the environment of data streams. Our extensive performance study shows that SRMTDS outperforms VFDTc on the memory usage and the anti-noise capability. The accuracy is also improved and the time complexity of SRMTDS is much lower than that of VFDTc for databases with higher numerical attribute dimensions.

The rest of the paper is organized as follows. Section 2 reviews the algorithms of VFDT and VFDTc and the model of Random Decision Trees. Our Semi-Random Decision Tree (SRDT) model is described in detail in Section 3. Section 4 presents the SRMTDS algorithm and its characteristics. Section 5 provides our experimental study and Section 6 summarizes our results and future work.

2 Related Work

In this section we analyze related work in two dimensions. One dimension is related to the algorithms of VFDT and VFDTc that will be used in our comparative study, and the other refers to the models of random decision trees.

2.1 VFDT and VFDTc

VFDT was published in 2000 and can manage thousands of examples with a similar performance to a batch decision tree with enough examples. In VFDT, a decision tree is learned by recursively replacing leaves with decision nodes. Each leaf stores sufficient statistics about attribute values, which are used to evaluate the merit of split-tests in a heuristic evaluation function. If there is enough statistical support in favor of one test, the split-test is installed in the heuristic evaluation function of Information Gain at this node and the relevant information of this node is descended into its child nodes. The main innovation of the VFDT algorithm is the use of Hoeffding bounds to decide how many examples are necessary to observe before installing a split-test at each leaf.

The inequality of Hoeffding bounds is as follows. Consider a real-valued random variable r whose range is R. Suppose we have made n independent observations of this variable, and computed their mean $\bar{r}$, which shows that, with probability $1-\delta$, the true mean of the variable is at least $\bar{r}-\varepsilon$,

$$P(r \geqslant \bar{r}-\varepsilon) = 1-\delta, \quad \varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \qquad (1)$$

where the minimum value of n is the minimal number of split-examples $n_{\min}$ at the current node, which is user-defined and the value must satisfy the restricted conditions of other parameters in (1) (details in [1]).

The VFDT algorithm is only for data streams with discrete attributes. In 2003, its improved algorithm of VFDTc was proposed, which can deal with numerical attributes. VFDTc makes use of the structure of Binary tree (Btree) to store the values of numerical attributes, takes the Information Gain as the evaluation function and uses the Naïve Bayes classifier to classify test examples at the leaves, which introduces the overhead that is proportional to the depth of the Btree, at most $\log n$ (n represents the number of distinct values for the attribute seen so far). VFDTc maintains all the desirable properties of VFDT and both of them have the any-time characteristic for classification of data streams. However, the process of constructing a decision tree in VFDT or VFDTc depends on the database details so that the ability of classification is easy to be affected by noisy data, which leads to a weak performance on the anti-noise capability. In addition, both of them need to preserve the entire split instances at each node for descending into their child nodes to install the further split-tests, which would bring in a high memory cost. Therefore, when dealing with data streams with high attribute dimensions or the high-rate noise, these two algorithms become unsuitable.

2.2 Random Decision Trees

Randomized trees (RT)[20] were first proposed in 1997, to generate feature subsets from all features randomly and make use of the heuristic Information Entropy in traditional decision-tree learning to select split-nodes. Ho designed a "random subspace"[21] technique to select random subsets of the available features in training individual classifiers of an ensemble

in 1998. Dietterich introduced an approach termed "randomized C4.5"[22] in 2000, in which the 20 best candidate splits are determined and one of them is randomly selected for use at each node in decision tree construction. To verify the effectiveness of randomization, Bagging and Boosting for improving the performance of C4.5 were compared by experiments in [23] and the results show that randomization is slightly superior to Bagging under the condition of little or no noisy data, but not as accurate as Boosting. However, in situations with highly noisy data, randomized C4.5 does not do as well as Bagging, but performs better than Boosting. Breiman designed a model of Random Forests (RF)[24] in 2001, using the Bagging technique in tandem with random feature selection. RF is created in two ways: one is to form the simplest random forests with random features by choosing a small group of variables to split on, and the other is to create linear combinations of inputs. Contrary to the previous beliefs, [17] proposed a RDT model that selects the split attributes of nodes and split thresholds of numerical attributes randomly from the whole feature set instead of feature subsets.

The RDT model is divided into three phases. The first phase is to create a framework of decision trees, and construct N random trees with an equal height H. The experiments in [17] showed that the time and space complexities of this model are good and the accuracy is high if N is 10 and H equals half of the number of attribute dimensions. RDT selects a split attribute at each node randomly and proposes two schemes to deal with numerical attributes: one uses discretization to partition features, and the other chooses a value of a numerical attribute in its given range randomly as a split threshold. With the chosen split threshold, RDT generates two descendent branches ($<$ and $\geqslant$ the cut-point respectively). The second phase is to update node counts with the training database. In the N trees, each training instance travels from the root to a leaf and accumulates the number of nodes that it has passed through. The final phase is to classify the testing database with the posteriori probability formula, which computes the sum of probabilities for each class (i.e., $\Sigma_j P(i)$, where j is the j-th tree ($j \in \{1, \ldots, N\}$), i is the i-th class label ($i \in \{1, \ldots, M(classes)\}$), and $M(classes)$ is the number of class labels), and chooses the class label with the maximum average value ($\Sigma_j P(i)/N$) as one of the current testing instances.

RDT is not an incremental algorithm and the process of constructing the initial trees is independent of the training database. It only needs to scan the training database once to update the statistics of tree nodes; hence it has good space and runtime performances and an anti-noise capability, which is suitable for medium and large databases with discrete attributes. However, if the databases have numerical attributes that need to be processed in this model, the value ranges of the numerical attributes must be known in advance. Their split thresholds are determined randomly, which reduces the accuracy, especially if the rate of numerical attribute dimensions is high and the value ranges of the numerical attributes are large. The probability that the optimal split thresholds are correctly chosen with a random strategy in these cases decreases greatly, which affects the accuracy of classification drastically. RDT creates enough decision trees, which makes up for the deficiency of randomization to a certain extent but increases the overhead of space. Besides, RDT is not used to process data streams directly because of the constant sizes of the random decision trees. In this paper the SRDT model proposed in Section 3 not only has the advantages of RDT but also overcomes the deficiency of RDT. SRDT takes a heuristic method to deal with numerical attributes, which improves the classification accuracy of a single tree and is beneficial to reduce the whole scale of decision trees. In addition, with the increasing error rate of classification at leaves for testing data or due to space limitations, the created decision trees adjust their height dynamically, and the updating mechanism adapts to the characteristics of streaming data and deals with the classification of data streams efficiently.

3 Semi-Random Decision Tree Model

3.1 Model Description

Our SRDT model to be presented in this section is based on the RDT idea, and is much different from RDT in the processing of discrete and numerical attributes and the judgment for the class labels at leaves. The basic framework is illustrated as follows.

Input: Training set $DSTR = \{TR_1, TR_2, \ldots, TR_m\}$; Testing set $DSTE = \{TE_1, TE_2, \ldots, TE_n\}$; Attribute set $A = \{A_1, \ldots, A_{M(\mathrm{Att})}\}$; Initial height of tree h_0; The number of minimum split-examples $n_{\min}$; The parameter in Hoeffding bounds inequality δ; The threshold τ; Split estimator function $H(\bullet)$; The number of SRDT N; The size of training window *WinSize*; The checked period for constructing the alternative subtree *CheckPeriod*; The size of testing window for the alternative subtree *TestSize*; The classification error counts of N-SRDT *Errors*.

Output: A Semi-Random Decision Tree — SRDT.

Procedure of SRDT Algorithm {$DSTR$, $DSTE$, A, h_0, $n_{\min}$, δ, τ, $H(\bullet)$, N, *WinSize*, *CheckPeriod*, *TestSize*, *Errors*}

1) Create the framework of semi-random decision tree
   a) Generate the root of a single tree;
   b) Choose an available attribute A_j from A as the split-attribute at the current node;

if A_j is a discrete attribute
Generate $m+1$ child nodes (where m is the number of different values in A_j
else
Generate two child branches, i.e., $<$ and $\geqslant$ a cut-point value of A_j;
for each child node
Go to Step b) until the height of tree $= h_0$;

2) Update the node counts and determine the split-threshold of nodes with the training data
for each training data record TR_i $(i = 1, \ldots, m)$ from *DSTR*
a) Sort TR_i into a leaf of SRDT;
b) **if** the current node with A_j is a numerical attribute && the count of examples at this node $\geqslant n_{\min}$
{Compute the value of ϵ by Hoeffding bounds inequality;}
if $(((H(Value_x(A_j)) - H(Value_y(A_j))) > \varepsilon) \| (H(Value_x(A_j)) - H(Value_y(A_j))) \leqslant \varepsilon < \tau))$
{Set $Value_x(A_j)$ with the highest gain value as the split-point;}

3) Classify the testing data for each testing data record TE_i $(i = 1, \ldots, n)$ from *DSTE*
a) Travel the tree from the root to leaves and increase the counts of class labels at each passed node;
b) Classify the class label of this testing data record by the judging function of class labels;

return SRDT;

In Step 1), the split-attributes of tree nodes are determined by the random strategy that discrete attributes should not be chosen again in a decision path of the tree while numerical attributes can be chosen multiple times to create an initial framework of the decision tree. Step 2) firstly updates the statistic counts of nodes in the tree. An available training instance 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 instance. When the example reaches a leaf, the sufficient statistics are updated, such as, the number of instances passed the node, the counts of different class labels and the information of attribute values at nodes with numerical attributes or at leaves. When the statistic count of a node with numerical attribute A_j is up to the defined threshold $n_{\min}$, the split-test is installed to find the split-threshold by a heuristic method, i.e., compute the Information Gain of all different split-points of attribute A_j and choose the first two with the highest gain values of $H(Value_x(A_j))$ and $H(Value_y(A_j))$ (where $Value_x(A_j)$ and $Value_y(A_j)$ are the x-th and y-th attribute-values of A_j). When their absolute difference satisfies a certain condition, $Value_x(A_j)$ with the highest gain is determined as the split-threshold of the current node, which is called the "semi-random" strategy in comparison with the completely random method in RDT. In the final step, a certain classification function is chosen to classify the testing data by the statistic information from Step 2).

The advantages of the semi-random strategy are as follows: on one hand, the higher randomization the higher probability of a lower accuracy. Most of the attributes in real-world streaming data are numerical. If an arbitrary value is chosen randomly in the range of a numerical attribute as the cut-point, the probability that it is optimal in this case is only $|ovs - ovf|/TotalV$, where $TotalV$ is the length of the value range, ovf and ovs are the adjacent values of ovp, and ovp is the optimal value of the cut-point about the current numerical attribute. Nevertheless, the probability in SRDT with the heuristic method is up to 1. Obviously, with the more distant attribute values and the larger value ranges of numerical attributes, SRDT has more advantages of accuracy. Besides, the random strategy of choosing split-attributes makes our decision tree model independent of a specific database, which has a better performance of resilience to noise from the database. On the other hand, the higher probability, on which the chosen cut-points of the numerical attributes are optimal, is beneficial to decrease the total number of trees and the space cost. SRDT makes use of the heuristic method to deal with numerical attributes, which increases the classification accuracy of one single tree. Therefore, the initial height and the total number of decision trees are reduced on the condition of ensuring the classification accuracy, which saves the space overhead.

Detailed processes in the above framework will be discussed in Subsection 3.2.

3.2 Attribute Processing and Leaf Classification

Processing of Discrete Attributes

The $m+1$ child nodes generated at the first phase are m nodes with distinct values of A_j known previously plus an additional node which is designed for not yet observed values of A_j. If only m branches are generated in the traditional way, when the new values of attributes are met, it has no choice but to ignore them, which may lead to a larger error rate of classification. The nodes with known attributes take a random strategy recursively to choose split attributes and generate descendent nodes, while the additional branch node does not do that until having real training examples at this node, otherwise, only creates a framework of node. The reason is that it will enlarge the sizes of trees immensely and cause the heavy overhead of space if the child nodes are generated blindly. The corresponding statistics are defined to count the numbers of occurrences of new attribute values, and if

the counts are higher than the user-defined threshold, they will be added into the known feature information of the database. When a new node is created with an attribute, the number of its child nodes is not $m+1$, but $m'+1$ where $m' = (m +$ the number of the possible values of this attribute), while its ancestor nodes before this node do not need to add any branches with the values of this new attribute until test instances pass through the tree. It is sensible to take this strategy of generating only one additional child node, not k ($k > 1$) additional child nodes each time, because k is hard to confirm and is disadvantageous to save the space overhead. Actually, the adopted method with a dynamic update process is more efficient.

Processing of Numerical Attributes

In the second phase, SRDT discards the random idea about determining the split thresholds of numerical attributes, introduces the inequality of Hoffeding bounds, and takes advantage of the traditional information entropy method to find optimal cut-points, which changes the process of creating the tree randomly into a semi-random process, i.e., choosing discrete attributes still randomly but selecting the split thresholds of numerical attributes with a heuristic method of information entropy.

When an example is available at the node with attribute A_j, the value of A_j of this example is inserted into the list-array $list[A_j]$ by ascending order, where $list[A_j]$ is a list-array for storing the structure of numerical attribute values. If the minimum number of examples at the current node amounts to $n_{\min}$, the heuristic method is used (i.e., if the corresponding class label of the candidate cut-point i is the same as that of its adjacent cut-point $i+1$, the cut-point i is abandoned and the cut-point $i+1$ is chosen till the class label of the current candidate cut-point is different from that of the adjacent one, which avoids the exhaustive tests for all values of numerical attributes and reduces the time cost) to compute the Information Gain. Suppose $H(\bullet)$ is the estimator function of attribute values, according to the information entropy, the value of R in $H(\bullet)$ is $\log_2(M(classes))$. The equations of computing information gain for the i-th value of attribute A_j (i.e., the i-th candidate cut-point) are as follows.

$$H(A_j(i)) = info(A_j) - info(A_j(i)), \tag{2}$$

$$\begin{aligned} info(A_j(i)) = {} & P(A_j < Value_i) \times lower_i(A_j(i)) \\ & + P(A_j \geqslant Value_i) \times higher_i(A_j(i)), \end{aligned} \tag{3}$$

where $info(A_j)$ is the information gain of A_j at the current node, $Value_i$ is the value of cut-point i, $lower_i(A_j(i))$ is the information expectation of the values of $A_j < Value_i$ (4), $higher_i(A_j(i))$ is the information expectation of the values of $A_j \geqslant Value_i$ (5), and C_k is the k-th class label ((4) and (5)).

$$\begin{aligned} lower_i(A_j(i)) = {} & -\sum_K P(K = k | A_j < Value_i) \\ & \times \log_2(P(K = k | A_j < Value_i)), \end{aligned} \tag{4}$$

$$\begin{aligned} higher_i(A_j(i)) = {} & -\sum_K P(K = k | A_j \geqslant value_i) \\ & \times \log_2(P(K = k | A_j \geqslant Value_i)). \end{aligned} \tag{5}$$

Choose two candidate cut-points with the highest information gain, $A_j(x)$ and $A_j(y)$ ((2)), and compute the difference $\Delta H = H(A_j(x)) - H(A_j(y))$ and use the value of ε obtained from (1). For a given τ, if $\Delta H > \varepsilon$ or $\Delta H \leqslant \varepsilon < \tau$, the attribute value $Value_x$ with the highest information gain will be selected as the final cut-point.

Judgment Method of Class Labels for Leaves

The proposed SRDT adopts two kinds of classification strategies, one of which is to choose the most representative class (majority class) at leaves, and the other of which is to use a Naïve Bayes method taking into account not only the prior distribution of the classes but also the conditional probabilities of the attribute-values given the class.

The second strategy in SRDT makes use of the Naïve Bayes method from VFDTc as classification function of leaves. The process is as follows. Each example is denoted by an n-dimensional feature vector $\boldsymbol{E} = \{x_1, x_2, \ldots, x_n\}$, which describes n measures of attributes in the database. Supposing there are m classes, given an unlabeled class of example $\boldsymbol{E}$, the classification of Naïve Bayes maps it to the class label of C_i, iff $P(C_i|\boldsymbol{E}) > P(C_j|\boldsymbol{E})$, $1 \leqslant j \leqslant m$, $j \neq i$. In accordance with Bayes theorem: $P(C_i|\boldsymbol{E}) = P(\boldsymbol{E}|C_i) \times P(C_i)/P(\boldsymbol{E})$. Because $P(\boldsymbol{E})$ is a constant for all classes, it only needs to maximize $P(\boldsymbol{E}|C_i) \times P(C_i)$. Use the priori probability formula $P(C_i) = s_i/s$, where s_i is the number of examples with the class label of C_i and s is the total number of examples at the current leaf node. Supposing the attributes are independent of each other for the given class, $P(\boldsymbol{E}|C_i) = \Pi_k p(x_k|C_i)$ $(k = 1, \ldots, n)$. For discrete attributes, $p(x_k|C_i)$ can be computed directly with the statistics of leaves; for numerical attributes, SRDT takes advantage of discretization to divide the sequential attribute-values into Min(10, the number of discrete attribute-values of x_k) intervals, to compute the number of examples $Count_i[j]$ whose values belong to the j-th interval with the i-th class label in all values of attribute x_k, and to estimate $p(x_k|C_i)$ with the

equation $p(x_k|C_i) = Count_i[j]/s_i$.

4 Semi-Random Multiple Decision-Tree Algorithm for Mining Data Streams

4.1 Algorithm Description

The SRMTDS algorithm based on the above SRDT model is presented in this section. It creates N semi-random decision trees with the initial height h_0 for classifying data streams. When the number of training examples satisfies a certain threshold, rescan the leaves and change some nodes with a higher error rate of classification into the decision nodes to split. If the total size of the trees is over a certain restriction, release some space of leaves or nodes with numerical attributes and stop splitting. Finally, the minimum error rate of classification is chosen in N trees as the classification result of the testing database. The basic idea is described as follows.

Input: *DSTR*; *DSTE*; A; The number of semi-random decision trees N; h_0; $n_{\min}$; δ; τ; The maximum number of rescanning *MaxScanPeriod*; The maximum value of memory consumption *MaxMemorySize*; The classification error counts of N semi-random decision trees *Errors*; The indicator of classification methods *Flag*.

Output: The minimum classification error-rate on testing data Min (*Errors*).

Procedure of SRMTDS Algorithm {*DSTR*, *DSTE*, A, N, h_0, $n_{\min}$, δ, τ, *MaxScanPeriod*, *MaxMemorySize*, *Flag*}

```
{for (i = 1; i ⩽ N; i++)
   {//Construct the i-th SRDT-tree T_i with the height
    //of h_0
   Create_SRDT(A, T_i, h_0);
   for (TR_j (j = 1, ..., m) ∈ DSTR)
     {for (i = 1; i ⩽ N; i++)
        {//Update the counts of traveled nodes
        //of N-trees and determine the split-cuts
        Update_EstimateConSplitVal(T_i, DSTR, n_min,
        δ, τ);
          if (the total number of observed training instances at T_i-tree == MaxScanPeriod && the
          current total memory overhead of the T_i-tree
          ⩽ MaxMemorySize)
            {Change leaves into decision nodes;}
          if (the current total memory overhead of T_i-
          tree > MaxMemorySize)
            {Release some space of leaves or nodes with
            numerical attributes;}}}}
for (i = 1; i ⩽ N; i++)
  //Accumulate the error count of classification in T_i
  for testing instances into Error[i]
    for (TE_j (j = 1, ..., n) ∈ DSTE)
    //Flag = true: Naïve Bayes method;
    //Flag = false: Majority class method
    Errors[i] = ClassifyTestSet(T_i, DSTE, Flag);
    Choose the minimum error-rate from N-trees
return GetMinErrorRate(Errors);}
```

Considering the problem of space overflow, the SRMTDS algorithm takes the strategy of removing the space of leaves or nodes with numerical attributes, when the total statistic overhead of memory at the current tree is up to the defined threshold of memory bound — *MaxMemorySize*. Collect all the leaves and discard their statistic information one by one, i.e., release the heap space for storing the information of attribute values and class labels at a leaf. Check whether the total consumption of space is over *MaxMemorySize* or not at every removing operation of one leaf. This process is kept on at all times till the total consumption on space of this current tree is lower than the restriction value. If the total memory consumption is still higher than the value of *MaxMemorySize* after having removed all of the space at leaves, collect all the nodes with numerical attributes and release their space in a similar process. In the worst situation, the space overflow problem still exists after all of the above operations, and the nodes will be deleted from the highest level of the tree till to the root. In fact, this situation is nearly impossible.

4.2 Analysis

In the analysis of the strength and correlation of classifiers, an upper bound on the generalization error is given by Breiman for a model of random tree ensembling[20], i.e.,

$$PE \leqslant \bar{p} \cdot \frac{1 - s^2}{s^2} = \bar{p} \cdot \frac{1}{s^2 - 1} \tag{6}$$

where s is the strength of a decision tree, i.e., the measure of classification accuracy of a classifier, and $\bar{p}$ is the mean correlation, i.e., the dependence between classifiers. This upper bound is applicable to classifier based ensembling, including complete-random trees[21]. Apparently, the proposed ensemble model of SRDT based on RDT, i.e., SRMTDS, in this paper also satisfies the constraint of this upper bound. In (6), the generalization error of ensemble classifiers is directly proportional to the value of $\bar{p}$ and is inversely proportional to the value of s. Each semi-random decision tree in SRMTDS is independent in the generation process and there are no interactions between each other during classification. Therefore, the value of $\bar{p}$ is less, but the value of s in SRMTDS is unfixed because of the randomized method used in SRDT.

The probability that our model is optimal is proportional to the classification accuracy of the model, and we now estimate the probability that the ensemble of SRDT is an optimal model. In the worst case, the database has only one relevant attribute A_i and the remainder $M(Attrs) - 1$ attributes are irrelevant. Suppose each decision path from the root to a leaf

is completely independent and all of the discrete attributes are binary, the total number of decision paths in SRDT is up to 2^{h_0-1}. The posteriori probability estimate by any number of irrelevant attributes approximates the default probability in the training data. If at least one semi-random decision tree uses the only good attribute A_i as the split-attribute to classify the testing data, the best classification probability from N trees is better than the default probability. Because each attribute is chosen with an equal probability during tree construction and has the same probability to appear in a decision path, the probability of one attribute to appear in the L-th level of a decision path is as follows ($K = M(Attrs)$).

$$\begin{aligned} p = & [(K-1)/K][(K-2)/(K-1)]\cdots \\ & [(K-L)/(K-L+1)] \\ & \cdot [1/(K-L)] = 1/K = 1/M(Attrs). \end{aligned} \tag{7}$$

Therefore, the probability of the remaining irrelevant attributes to appear in all N trees satisfies:

$$P(irre) < (1-p)^{(2^{h_0-1})\square N} = \left(1 - \frac{1}{M(Attrs}\right)^{2^{h_0-1})\square N}, \tag{8}$$

and the least probability for at least one path to involve the only relevant attribute A_i is:

$$\begin{aligned} P(M(Attrs), N, h_0) = & 1 - (1-p)^{N\square 2^{h_0-1}} \\ = & 1 - \left(1 - \frac{1}{M(Attrs}\right)^{N\square 2^{h_0-1}} \end{aligned} \tag{9}$$

i.e., with the confidence of at least $P(M(Attrs), N, h_0)$, the ensemble model of SRDT is optimal. (8) and (9) show that the value of $P(irre)$ is in inverse proportion to the values of N and h_0 and is proportional to the number of attributes — $M(Attrs)$, but the situation of $P(M(Attrs), N, h_0)$ is contrary to $P(irre)$. In other words, the more the total number (N) of semi-random decision trees and the higher the height of trees-h_0, the higher the probability of relevant attributes to appear in the decision path. Therefore, the confidence that the proposed model is optimal is higher and the value of generalization error is less.

The analysis of the time complexity for a single semi-random decision tree in comparison with the algorithms of RDT and VFDTc is given as follows. When creating the framework of a semi-random decision tree with the initial height h_0, the time cost is relevant to h_0 and the number of distinct attribute values is the same as in RDT. In the worst case of dealing with discrete attributes, the time complexity is $O(M(Attrs)/(M(Attrs) - H(node) + 1))$ (where $M(Attrs)$ is the number of attributes, and $H(node)$ ($\leqslant h_0 \leqslant M(Attrs)/2$) is the height of the current node. This time complexity shows that a split attribute is determined by two choices at most; in the training phase of SRMDTS, the nodes with discrete attributes do not consume much time and the major overhead is on the nodes with numerical attributes, the time complexity of which is $O(m_j^A)$, and it is $O(\Sigma_j m_j^A)$ for a node in VFDTc ($j = 1, \ldots, M(Attrs)$) (where m_j^A is the number of distinct values of attribute A_j); while in RDT that is only $O(1)$ on the condition of having known the value bounds of all numerical attributes. It is obvious that the time cost of SRMTDS in the training phase is much less than that of VFDTc in the worst case of dealing with a database with numerical attributes, while it takes more time than that of RDT. With respect to the time complexity of classification, if the majority class method is chosen, it is $O(M(classes))$ for each leaf node. Because $M(classes)$ is generally quite small, the time consumption at each leaf node is little. Though the number of leaves in SRMTDS is different from that of VFDTc, their time costs are still roughly the same. While the trees in RDT need to vote for the best classification result, the time overhead of classification is more than that of SRMTDS. If the Naïve Bayes method is chosen (which is not used in RDT), the time complexities of each leaf node for SRMDS and VFDTc are: $O(1)$ for a node with a discrete attribute, $O(\Sigma_i(n\Sigma_j(\log_2 m_j^A + Count_i[k])))$ for a node with a numerical attribute (where $j = 1, \ldots, M(Attrs)$, $i = 1, \ldots, M(classes)$, $k = 1, \ldots, \min(10, m_j^A)$, and the meaning of $Count_i[k]$ is the same as that of Subsection 3.2). Therefore, the time discrepancy lies on the concrete value of n (the total number of examples at the current node), the count of different attribute values for attribute A_j (m_j^A), and the count of attribute-value distribution about partition intervals of attribute A_j ($Count_i[k]$).

With respect to the analysis of space complexity in a single tree, the overhead of space is relative to the features of attributes in the database for the algorithms of SRMTDS and VFDTc, i.e., $O(T_n M(Attrs) VM(classes))$ (where T_n is the total number of nodes in a tree; V is the maximum number of attribute-values in an attribute). Each non-leaf node only keeps the information about the current split-attribute in SRMTDS, while in VFDTc each node holds the information of all attributes. Therefore, the approximate ratio of their space consumption at each non-leaf node is $1/M(Attrs)$. However, the overhead of space at each leaf node is generally equivalent. While in RDT, the space consumption is directly proportional to the numbers of trees and attribute dimensions so that the whole overhead of space is $N \cdot 2^{h_0-1} \cdot Mem(node) + Mem(instance)$ (supposing all of the discrete attributes are binary and the to-

tal number of nodes in this tree is 2^{h_0-1}. $Mem(node)$ and $Mem(instance)$ represent the memory consumptions of a single node and a single instance respectively). This space overhead is nearly invariable for the constant values of h_0 and N. In fact, the initial overhead of memory in SRMTDS can also be represented by $N \cdot 2^{h_0-1} \cdot Mem(node)$). With the arrived training data, the memory consumption is changed with the number of nodes and the stored information.

About the setting of N: if $N = 1$, i.e., only creating a single semi-random decision tree, the property of randomization will cause an unstable classification result; however, creating multiple trees can make up for this deficiency and make sure that better results over a single tree occur. Due to the random strategy used in SRDT, the classification accuracy of a single tree is improved so that it benefits to decrease the values of h_0 and N adaptively to save space cost on the condition of ensuring classification accuracy as compared with RDT. Besides, the designs of creating N trees and having no interactive operation in the classification phase are useful for multi-CPU and multi-PC platforms, and only the learning time (training time) needs to be computed by the maximum training time cost from N trees. The relevant settings of N and h_0 are presented in Subsection 5.1.

5 Experimental Evaluations

The first part of this section discusses the heuristics to choose the initial height h_0 as well as the number of trees N for different types of databases, and the second part, based on the optimal values of parameters h_0 and N obtained from the first part, analyzes the performances of SRMTDS on run time, space, accuracy and the anti-noise capability as compared with VFDTc. (As the SRMTDS algorithm is designed to classify data streams, while the RDT algorithm is not suitable for dealing with streaming data directly, experimental comparisons between these two algorithms are not provided in this paper.)

The databases used for our experiments are from KDDCup99[26], the LED & WaveForm databases from UCI[27], and also some additional medium- to large-sized databases from UCI. The KDDCup99 database is simulated as streaming data of network intrusion detection. The choice of the LED and WaveForm databases is motivated by the existence of synthetic database generators at the UCI repository. We can generate arbitrary sizes of databases and plot "continuous" learning curves on the aspects of accuracy, anti-noise capability, time and space costs for better analyses; besides, both of them are also simulated as experimental data in VFDTc. The training sets of a varying number of examples are generated from these generators, starting from 100k to 1500k, and the test set contains 250k examples. SRMTD and VFDTc are used with the parameter values $\varepsilon = 10e^{-7}$ and $n_{\min} = 300$. All experiments are perf P4 2.66GHz PC with 512MB main memo Windows XP Professional. Both algorithn ten in Visual C++.net.

= 10e-7
this symbol is wrong!

5.1 Analysis of Parameters h_0 and

For finding a better balance between and accuracy, the value of h_0 needs to be chosen properly. If half of the number of attribute dimensions ($d/2$) is lower than 10, h_0 is always initialized to $d/2$. In the following study, we only consider the value of $d/2$ is more than 10.

Considering N is 1, according to the conclusion of $d/2$ as the tree height in RDT and the experience of 5 as the optimal height of decision trees, we test the values of h_0 exhaustively and run each test 10 times. Given the value range of h_0 from 4 to 8 for the database with both discrete and numerical attributes and from 2 to 12 for the database with only discrete attributes, we obtain the error rate, time and space costs averaged over all the runs in Fig.1 and Tables 1 to 3. Our extensive studies show that the space cost is proportional to h_0, while the classification error rate with the majority class method is inverse proportional to it. For the database with discrete attributes, the time cost is in proportion to its data size without the relation to

Table 1. Error, Time and Memory Ratios from WaveForm Databases

	Level/Level			
	$h_0 = 4/$ $h_0 = 5$	$h_0 = 6/$ $h_0 = 5$	$h_0 = 7/$ $h_0 = 5$	$h_0 = 8/$ $h_0 = 5$
WaveForm DataBase — 40 AttrsC-Max				
$errorR$	1.09	0.95	0.90	0.91
$timeR$	1.03	1.05	1.06	1.05
$memoR$	0.58	1.62	2.46	3.69
$MultiT$	0.66	1.62	2.33	3.52
WaveForm DataBase — 21 AttrsC-Max				
$errorR$	1.08	0.95	0.88	0.88
$timeR$	1.03	1.06	1.12	1.11
$memoR$	0.73	1.65	2.54	3.62
$MultiT$	0.80	1.67	2.52	3.54

Table 2. Error Rate, Time and Space Cost Ratios in the Databases of Hybrid Attributes with Naïve Bayes

	$h_0 = 4/$ $h_0 = 5$	$h_0 = 6/$ $h_0 = 5$	$h_0 = 7/$ $h_0 = 5$	$h_0 = 8/$ $h_0 = 5$
KDDCup99-334639-59542CD-41-Pro				
$errorR$	2.12	1.17	2.00	5.80
$timeR$	0.97	0.87	0.80	0.26
$memoR$	1.58	2.33	4.36	4.77
$MultiT$	2.79	2.36	6.95	7.21
Covertype-170266-73798C-54-Pro				
$errorR$	0.98	0.97	0.96	0.96
$timeR$	0.99	0.95	0.90	0.87
$memo\ R$	1.03	0.99	1.41	1.33
$MultiT$	1.00	0.91	1.21	1.12

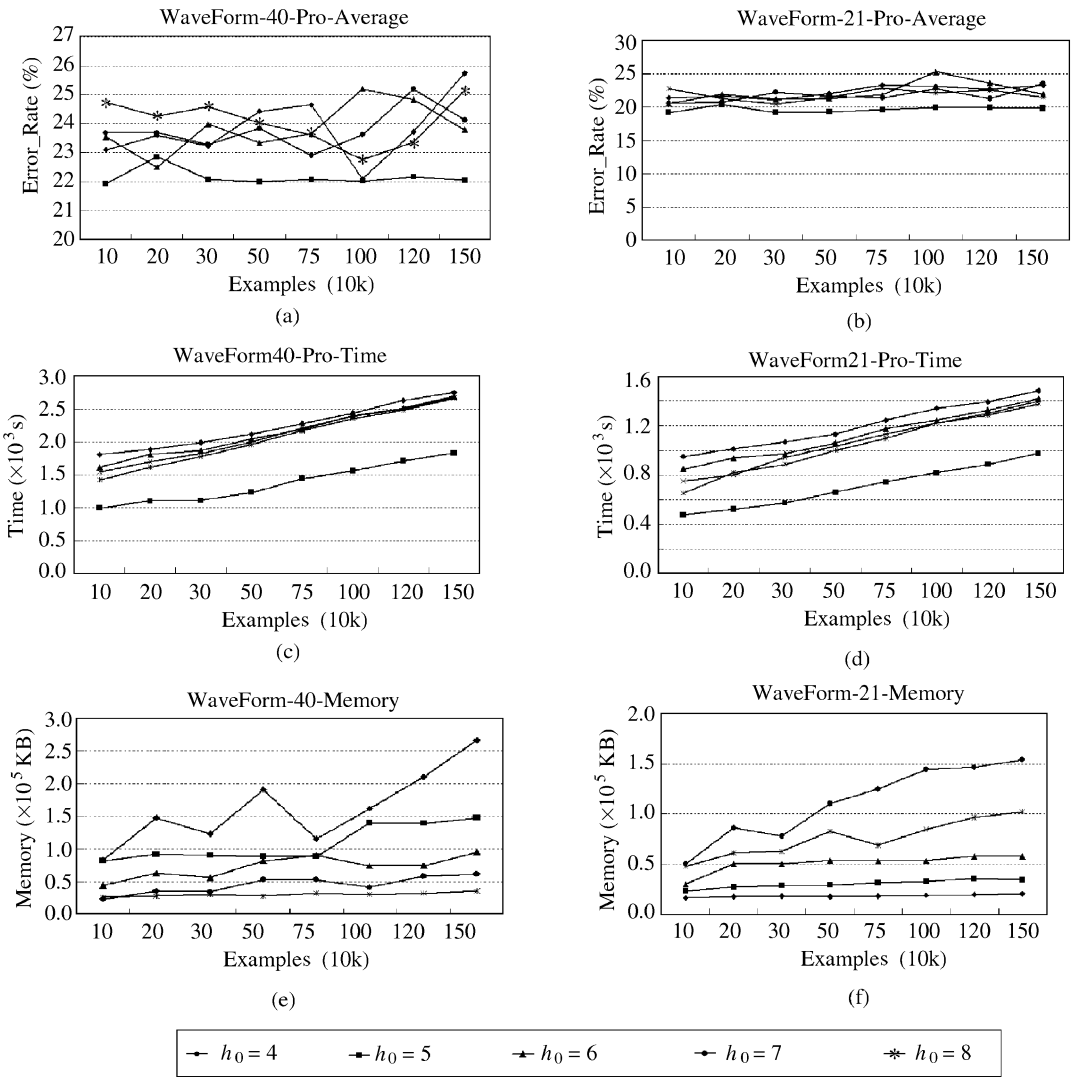

Fig.1. Relations of error rate (%) — h_0 ((a), (b)), time cost(s) — h_0 ((c), (d)) and space cost (KB) — h_0 ((e), (f)).

Table 3. Error Rate, Time and Space Cost Ratios in the Databases of Hybrid Attributes with Majority Class

	$h_0 = 4/$ $h_0 = 5$	$h_0 = 6/$ $h_0 = 5$	$h_0 = 7/$ $h_0 = 5$	$h_0 = 8/$ $h_0 = 5$
KDDCup99-334639-59542CD-41-Max				
errorR	0.97	1.86	1.45	2.00
timeR	0.97	1.02	1.01	1.03
memoR	1.41	1.47	1.55	2.18
MultiT	0.97	1.86	1.45	2.00
Covertype-170266-73798C-54-Max				
errorR	1.03	1.02	0.99	0.99
timeR	0.61	0.77	1.04	1.18
memoR	1.04	1.01	1.14	1.22
MultiT	0.65	0.79	1.18	1.43

h_0, so is the database with hybrid attributes when it is classified by means of the majority class method.

For the database with numerical attributes, the initial height $h_0 = 5$ is optimal when classified with the Naïve Bayes method. Although the space cost (in Fig.1(e) and (f)) is 0.687 more than the minimum cost of space with $h_0 = 4$ averagely, the average error rate (in Fig.1(a) and (b)) and the average time cost (in Fig.1(c) and (d)) are minimum, and they are only up to 58.9% and 93.1% of that of $h_0 = 4$ respectively. Thus, the value of error rate multiplied by runtime-cost rate and space-cost rate for $h_0 = 5$ is still minimum, which is 93.5% of the product value of $h_0 = 4$. The overhead of time with the majority class method has no relation to h_0 and the higher the value of h_0, the more the increasing rate of space consumption than the decreasing rate of error rate, in Table 1 (explanation: in Tables 1 to 3, what we list are the ratios of average error rate, average time or space cost of $h_0 = 5$ as a denominator and that corresponding value of $h_0 = i$ ($i \neq 5$) as a numerator for the databases with both numerical and discrete attributes, where the vari-

Table 4. Relation Between h_0 and Time Cost in the Databases with Discrete Attributes

Size	Level										
(10k)	$h_0=2$	$h_0=3$	$h_0=4$	$h_0=5$	$h_0=6$	$h_0=7$	$h_0=8$	$h_0=9$	$h_0=10$	$h_0=11$	$h_0=12$
LED DataBase — 24 Attrs — 0% NoiseD–Max (T+C time)											
10	17+48	18+46	18+47	17+48	17+46	18+46	17+48	17+50	18+48	18+51	18+44
20	34+48	35+49	35+48	34+49	35+46	35+47	34+49	36+50	36+43	36+46	36+46
30	53+47	51+48	54+48	51+48	52+47	53+47	52+48	53+51	54+46	54+47	55+45
40	62+49	62+48	61+49	69+48	62+48	63+48	69+50	71+47	71+43	73+48	77+48
50	85+46	85+46	84+49	85+48	86+48	86+47	88+48	89+50	90+48	92+53	94+46
LED DataBase — 24 Attrs — 0% NoiseD-Pro (T+C time)											
10	17+102	17+102	17+100	17+101	17+99	17+92	17+98	17+100	18+98	18+96	18+94
20	34+101	34+102	34+100	34+100	34+91	34+91	34+95	36+97	36+94	36+97	36+95
30	51+101	51+100	51+100	51+100	52+98	52+91	52+96	53+99	54+98	54+97	55+95
40	63+102	61+100	62+101	68+102	62+94	61+92	69+93	71+96	71+100	73+96	77+96
50	84+101	85+102	85+102	86+100	87+92	53+93	88+94	89+96	90+98	92+98	94+96
Conect-67557-67557D-42—Max(T+C time)											
6.8	26+29	29+32	29+33	28+32	28+32	28+32	30+34	33+36	32+35	–	–
Conect-67557-67557D-42—Pro(T+C time)											
6.8	26+44	29+46	30+48	28+46	27+46	30+49	31+49	32+50	57+56	–	–

able $MultiT$ is the product value of error rate ($errorR$), time-cost rate ($timeR$) and space-cost rate ($memoR$); the smaller the value of $MultiT$, the better the value of h_0). So only the values of $h_0 \leqslant 5$ need to be considered. Though the memory requirement of $h_0 = 4$ is less than that of $h_0 = 5$, the error rate is larger; hence we still select 5 as the initial height when classifying with the method of majority class. Similarly, according to the principle that it is better to choose the minimum value of $MultiT$ under the condition of balancing the three values of $errorR$, $timerR$ and $memoR$, the experimental results for the databases with hybrid attributes in Table 2 and Table 3 show that $h_0 = 5$ is optimal for these applications.

It is necessary to mention that the values of $timeR$ in Table 2 are decreasing with the increasing values of h_0 while the reverse is the case in Table 3. The reasons are as follows. In Table 2, the values of $timeR$ are the results classified by the Naïve Bayes method. It is apparent that the lower the height of a tree is, the higher the number of instances accumulated at the leaves is. The time consumption is directly proportional to the number of instances at the leaves in Naïve Bayes. In addition, the decreasing rate of classification time with the increasing value of h_0 is more than the increasing rate of training time. Therefore, when h_0 is 4, the overhead of time is the highest. However, Table 3 lists the results classified by the majority class method. The time consumption of classification is much lower. The main time is consumed in the training phase and is proportional to the height of the tree directly so that the overhead of time is the lowest when the height of h_0 equals 4.

No matter what the value of h_0 is, the overhead of time is always stable for the databases with discrete attributes (in Table 4). Therefore, we only take into account the classification error and memory consumption. Generally, the value range of h_0 is from 5 to $d/2$ for the databases with discrete attributes, which should satisfy the restricted condition of memory, i.e., the total number of nodes in the initial trees $k_0^h \leqslant (Mem_{\max}/N)/mem_{\text{node}}$, to find the maximum value of h_0, where k is the average number of attribute values in discrete attributes, $Mem_{\max}$ is the given maximum capacity of memory and mem_{node} is the memory cost of a node. $Mem_{\max}$ is either user-defined or designed by estimating the memory cost in VFDTc with the same database and the value of mem_{node} is often from 5KB to 50KB.

It should be pointed out that when we process the LED-noise% databases (containing noisy data) with the classification method of Naïve Bayes, the smaller the value of h_0, the larger the classification accuracy in Table 5, the reason of which is that the numbers of class labels and attribute values in the LED database used for experiments are fewer; meanwhile, the lower of the tree height means that more data accumulates at leaves. And the relation between class labels and attribute values in these examples is considered which brings the higher accuracy of Naïve Bayes probability. Therefore, the result is optimal if $h_0 = 2$ for the LED databases when classified with the Naïve Bayes strategy. The general conclusion from experiments is that the larger the value of h_0 in its value bounds, the better the experimental results for the databases with discrete attributes. Consequently, for the Connect4 database from UCI, the maximum value of h_0 is 6, though the error rate is improved averagely by 0.017, the memory cost is only 1/2.88 of that of $h_0 = 7$ and 1/127 of that of VFDTc. For the database LED, when classified with the majority class strategy, the maximum value of h_0 is 12. The error rate is the minimum and the memory cost is still lower in comparison to VFDTc (in Table 6), which is averagely 1/26 of that of VFDTc.

Table 5. Error Rate with Different Noise-Rate and h_0 in LED Database

Level	Noise_Rate 1%	5%	10%	20%	30%	40%	50%
LED DataBase — 24 Attrs — NoiseD-Pro Error_Rate (%)							
$h_0 = 2$	0.0	0.9	0.0	1.5	0.4	1.1	92.0
$h_0 = 3$	3.4	4.0	5.7	3.5	4.5	5.6	91.2
$h_0 = 4$	7.6	7.6	10.0	10.4	6.6	8.3	90.9
$h_0 = 5$	10.9	10.4	14.7	11.6	11.9	14.6	89.6
$h_0 = 6$	11.4	14.7	18.4	11.7	15.8	16.6	89.4
$h_0 = 7$	11.7	11.8	16.7	14.6	16.9	30.5	88.8
$h_0 = 8$	9.9	18.3	16.1	17.0	25.2	37.6	89.1
$h_0 = 9$	10.6	15.9	21.7	27.5	31.5	52.2	88.6
$h_0 = 10$	7.9	17.8	23.9	27.6	38.1	60.1	88.7
$h_0 = 11$	7.1	15.8	20.4	32.2	53.7	68.1	88.3
$h_0 = 12$	4.9	15.3	22.3	36.0	49.7	72.1	88.1

Table 6. Relation Between Memory and h_0 in the Database with Discrete Attributes

Size (10k)	10	20	30	40	50
	Memory (KB)				
	LED DataBase — 24 Attrs — 0% NoiseD-Max				
$h_0 = 8$	1 104	1 980	2 544	2 840	3 196
$h_0 = 9$	3 676	3 974	4 860	3 600	3 928
$h_0 = 10$	5 356	5 540	3 804	5 452	6 772
$h_0 = 11$	9 300	9 312	10 372	9 316	10 332
$h_0 = 12$	17 068	15 656	16 868	17 076	15 868
VFDTc	143 380	285 484	426 952	568 496	710 416
VFDTc/$h_{0\max}$	8.4005	18.235	25.311	33.292	44.77

Because the N-trees have no interactions with each other in the classification phase in SRMTDS, i.e., they do not vote for judging the class labels of testing database so that each tree generates a classification result. Finally we choose the minimum error rate from the N results of classification. Thus, the number of N determines the probability of the occurrence of better classification results. It is true that the larger the value of N the more the probability of occurrence of better results, but the costs of time and space restrict N's increase. The analysis by experiments demonstrates that SRMTDS has advantages in time, space and accuracy, if $N = 5$, see Subsection 5.2 for details.

5.2 Performance Analysis

Some denotations used for our experiments are illustrated as follows. V_M, V_P, S_M and S_P are the algorithms VFDTc (V) and SRMTDS (S) respectively with the classification strategies of majority class (M) and Naïve Bayes (P). Each database name consists of the name of database + the size of the training database + the size of the testing database + the type of the database (C: numerical, D: discrete, CD: hybrid) + the number of attribute dimensions. T+C Time is training + testing time (SRMTDS runs in a simulated parallel environment of multi-PC; therefore, the time cost is computed by the largest one of the N-trees, and the training time in SRMTDS includes the time consumption in creating the framework of a semi-random tree), and Memory is the memory consumption. The following tables only list the memory consumption of a single tree.

The experimental results of SRMTDS and VFDTc with some medium-sized databases from UCI and the KDDCup99 database are listed in Table 7, which shows the following facts: the accuracies of S_M and S_P are prominently higher than that of V_M and V_P respectively; meanwhile, the space cost of SRMTDS is much lower than that of VFDTc (the smallest memory requirement is 1/4.76 of that of VFDTc (Adult database) and the largest one is only 1/60.63 (KDDCup99 database)); in addition, S_M consumes the least time and the time costs of S_P and V_P for other databases are similar basically except for the KDDCup99 database. Although the time cost of S_P in the KDDCup99 database is 11.1 times more than that of V_P, its error rate of classification is only 1/68.5 of that of V_P, and time, space and accuracy are performed very well even when using the S_M algorithm alone.

Table 8 lists the results of dealing with numerical attributes in the WaveForm databases. It indicates that the space capability of SRMDTS outperforms that of VFDTc, the time consumption of S_M is the least while the accuracy is lower. It is notable that the cost of training time of S_P is lower than that of VFDTc and the accuracy is also improved, but the cost of classification time is higher. For instance, the average classification time of S_P for the WaveForm-40 database is 3.03 times more than that of V_P and 3.74 times for the WaveForm-21 database, while the total time consumption is averaged by 1.29 times as much as that of S_P for the WaveForm-40 database and by 1.62 times for the WaveForm-21 database. For large databases the classification time of S_P is larger than that of V_P. SRMTDS and VFDTc have the same time complexity in the classification phase for each leaf

Table 7. Experimental Results from UCI Databases and KDDCup99 Database

Databases Name	Error Rate (%)				(T+C) Time (s)				Memory (KB)	
	V_M	S_M	V_P	S_P	V_M	S_M	V_P	S_P	VFDTc	SRMTDS
Nursey-11000-11000D-8	62.63	26.40	9.07	2.03	0+1	1+2	1+2	0+2	6 976	504
Letter-20000-20000C-16	100.00	78.49	52.80	12.15	3+3	3+3	4+8	3+16	22 436	698
Adult-32561-16281CD-14	23.62	16.38	23.54	20.49	6+2	5+3	6+4	5+40	29 836	6 266
Conect-67557-67557D-42	34.14	32.25	35.85	37.52	20+16	19+20	21+19	19+23	50 040	3 932
Covertype-170266-73798C-54	38.65	34.68	28.81	26.77	90+24	68+28	92+31	71+283	182 128	8 204
KDDCup99-334639-59542CD-41	15.14	2.22	55.50	0.81	174+17	109+20	172+109	116+2998	291 984	4 816

Table 8. WaveForm Databases from UCI Data Generator

Size	Error Rate (%)				(T+C) Time (s)				Memory (KB)	
(10k)	V_M	S_M	V_P	S_P	V_M	S_M	V_P	S_P	VFDTc	SRMTDS
WaveForm DataBase — 40 AttrsC										
10	30.51	33.07	23.20	18.32	93+88	53+99	92+430	53+937	229 872	32 664
20	25.64	40.78	20.70	19.42	195+90	109+102	194+388	107+989	359 744	42 436
30	25.49	39.42	21.41	19.63	275+92	162+103	269+289	160+954	361 560	44 540
50	24.92	43.02	21.44	19.98	500+90	271+107	483+290	269+970	493 500	49 756
75	23.93	39.91	21.65	19.71	886+97	412+102	828+245	401+1039	524 744	49 756
100	23.93	34.08	21.43	19.42	2416+94	549+103	1920+198	539+1017	557 624	55 216
120	23.28	36.18	21.36	19.75	3047+87	656+101	3113+185	662+1051	582 004	58 676
150	23.10	38.00	21.27	19.80	5095+87	817+107	5096+168	819+1005	592 472	62 596
WaveForm DataBase — 21 AttrsC										
10	27.39	31.54	22.30	19.15	57+52	29+63	56+337	29+446	155 280	22 780
20	25.54	33.66	23.37	20.30	107+49	58+61	106+220	57+463	196 176	26 888
30	25.21	32.43	23.80	19.14	148+89	85+61	148+199	85+486	201 164	28 192
50	24.36	34.09	22.28	19.22	238+49	147+63	237+154	147+510	251 596	29 160
75	24.06	33.64	22.08	19.59	317+49	229+66	315+115	219+526	274 488	31 260
100	24.06	31.40	21.75	19.86	384+51	294+63	378+97	292+524	275 216	32 824
120	24.06	31.40	21.75	19.85	432+52	294+63	430+98	351+527	285 000	35 524
150	24.06	31.40	21.75	19.84	501+49	294+63	502+82	435+533	290 800	34 184

Table 9. LED Database from UCI Data Generator

Size	Error Rate (%)				(T+C) Time (s)				Memory (KB)	
(10k)	V_M	S_M	V_P	S_P	V_M	S_M	V_P	S_P	VFDTc	SRMTDS
LED DataBase – 24 Attrs – 0% NoiseD										
10	79.96	33.96	0.00	0.00	25+35	17+47	25+69	17+90	143 380	12 488
20	39.80	45.84	0.00	0.00	65+35	34+48	68+134	34+100	285 484	15 816
30	39.80	38.19	0.00	0.00	108+37	55+50	107+135	51+88	426 952	15 472
40	39.80	47.72	0.00	0.00	129+41	68+48	130+135	68+89	568 496	15 968
50	39.80	32.53	0.00	0.00	157+37	84+49	158+135	85+97	710 416	15 844

node, i.e., $O(\Sigma_i(n\Sigma_j(\log_2 m_j^A + Count_i[k])))$ (where the meanings of n, i, j, k, m_j^A and $Count_i[k]$ are the same as the explanations in Section 4), but because the values of n and m_j^A in SRMTDS are much more than that in VFDTc, this leads to more time cost to compute the probabilities of Naïve Bayes. For the databases with higher numerical attribute dimensions, the increasing rate of training time in VFDTc is much more than that of classification time in SRMTDS, which causes the fact that the total time consumption is distinctly higher (in Fig.2 and Table 8 for WaveForm-40 database). The reason is that the time cost of selecting the split attribute randomly could be negligible and the primary overhead is on determining the split threshold of numerical attribute A_j in the training phase of SRMTDS, the time complexity of which is $O(m_j^A)$, while the time complexity of each node in VFDTc is $O(\Sigma_j m_j^A)$ $(j = 1, \ldots, M(Attrs))$. So it is evident that the larger the number of attribute dimensions, the more the time consumption for VFDTc.

The results of the LED database with discrete attributes generated from its data generator are listed in Table 9, which show that except for the common characteristics of lower memory cost (the average memory cost of VFDTc is 26.56 times more than those of SRMTDS) and higher accuracy, SRMTDS has more advantages in training and classification times than that of VFDTc. The major reason is that the attributes in LED are discrete, the time consumption of training is little and the probabilities of Naïve Bayes are computed with class statistics stored in the main memory in the classification phase, the time cost of which for one node only amounts to the time complexity $O(1)$.

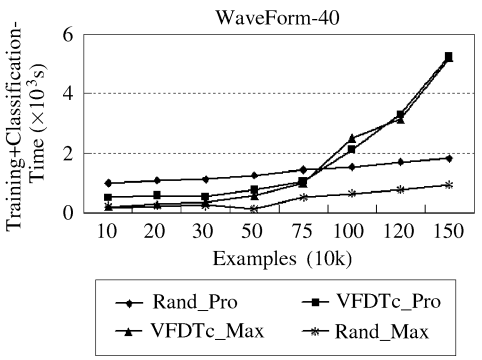

Fig.2. Training and classification time consumption of WaveForm-40 database.

In another dimension, we have compared the anti-noise capabilities between SRMTDS and VFDTc. Our experimental study shows that the anti-noise capability of S_P is the strongest, and it performs better than that of S_M as compared with VFDTc, in Fig.3. The reason is that the higher noisy ratio indicates the

fewer of examples containing true class labels in the database, but VFDTc takes advantage of the information entropy method to compute the maximum value of information gain for determining split information of nodes, which takes account of the distribution of classes, and it is possible that the chosen split attribute is the attribute with a smaller value of information gain in the database without noisy data; thus, the effect on selecting the split-attribute leads to a larger discrepancy of classification results. While SRMTDS makes use of a semi-random strategy, by which the process of selecting attributes is independent of the distribution of classes and reduces the impact of noisy data on classification largely.

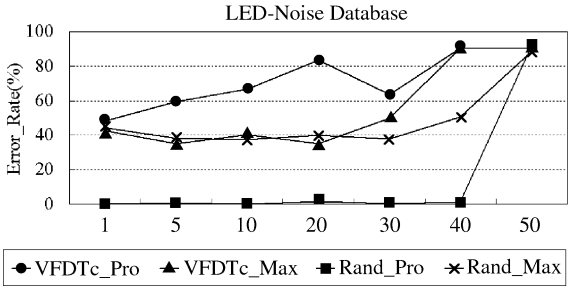

Fig.3. Relation between error rate and noise rate in LED-noise% database.

6 Conclusion

In this paper, we have proposed an incremental algorithm on Semi-Random Multiple decision Trees for Data Streams (SRMTDS) based on the Random Decision Tree model. SRMTDS makes use of the information entropy, combines the Hoeffding bounds to choose the cut-points of numerical attributes, and introduces Naïve Bayes classifiers from VFDTc. Our experiments have shown that SRMTDS has an improved performance in time, space, accuracy, and the anti-noise capability as compared with the state-of-the-art algorithm VFDTc. Meanwhile, SRMTDS still needs to create multiple decision trees so that it has a disadvantage in stand-alone environments. It is better suited for multi-CPU or multi-PC platforms. How to further decrease the runtime and improve the interactive voting mechanism in the classification phase among multiple trees, and how to deal with concept drifting over time are our future work with SRMTDS.

Acknowledgements The authors would like to thank the anonymous reviewers whose constructive comments and advice have helped improve this paper.

References

[1] Pedro Domingos, Geoff Hulten. Mining high-speed data streams. In *Proc. Knowledge Discovery and Data Mining*, Boston, MA, USA, 2000, pp.71~80.

[2] Gama J, Rocha R, Medas P. Accurate decision trees for mining high-speed data streams. In *Proc. the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington DC, USA, 2003, pp.523~528.

[3] Qiang Ding, Qin Ding, Perrizo W. Decision tree classification of spatial data streams using Peano count trees. In *Proc. ACM Symposium on Applied Computing (SAC'02)*, Madrid, Spain, March 2002, pp.413~417.

[4] Guha S, Mishra N, Motwani R, O'Callaghan L. Clustering data streams. In *Proc. IEEE FOCS, 41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, USA, 2000, pp.359~366.

[5] Moses Charikar, Liadan O'Callaghan, Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proc. the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, San Diego, CA, USA, 2003, pp.30~39.

[6] L O'Cllaghan, Nina Mishra, Adam Meyerson. Streaming-data algorithms for high-quality clustering. In *Proc. ICDE'02*, San Jose, CA, USA, 2002, pp.685~694.

[7] Ordonez C. Clustering binary data streams with K-means. In *Proc. the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, San Diego, CA, USA, 2003, pp.12~19.

[8] Aggarwal C, Han J, Wang J, Yu P S. A framework for clustering evolving data streams. In *Proc. the 29th VLDB Conference*, Berlin, Germany, 2003, pp.81~92.

[9] Xiaoyun Zhou, Zhihui Su, Baili Zhang, Yidong Yang. An efficient discovering and maintenance algorithm of subspace clustering over high dimensional data streams. *Journal of Computer Research and Development,* 2006, 43(5): 834~840.

[10] Xuli Jun, Xiekang Lin, Xu Hong. Discovering frequent itemsets over data streams. *Journal of Shanghai Jiaotong University*, 2006, 40(3): 502~506.

[11] Chang J H, Lee W S. Finding recent frequent itemsets adaptively over online data streams. In *Proc. KDD-2003*, 2003, Washington DC, USA, pp.487~492.

[12] Lijun Xu, Kanglin Xie, Hong Xu. Discovering frequent itemsets over data streams. *Journal of Shanghai Jiaotong University*, Washington DC, USA, 2006, 40(3): 502~506.

[13] Guojun Mao, Xindong Wu, Xingquan Zhu, Gong Chen, Chunnian Liu. Mining maximal frequent itemsets from data streams. *Journal of Information Science*, 2007, 33(3): 251~262.

[14] Gong Chen, Xindong Wu, Xingquan Zhu. Sequential pattern mining in multiple streams. In *Proc. ICDM'05*, Houston, TX, USA, 2005, pp.585~588.

[15] Utgoff P E, Berkman N C, Clouse J A. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 1997, 29(1): 5~44.

[16] Kalles D, Morris T. Efficient incremental induction of decision trees. *Machine Learning*, 1996, 24(3): 231~242.

[17] Fan W, Wang H, Yu P S, Ma S. Is random model better? On its accuracy and efficiency. In *Proc. ICDM'03*, Melbourne, FL, USA, 2003, pp.51~58.

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

[19] Maron O, Moore A. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. Advances in Neural Information Processing Systems, Cowan J D, Tesauro G, Alspector J (eds.), San Mato: Morgan Kaufmann, CA, 1994.

[20] Amit Y, Geman D. Shape quantization and recognition with randomized trees. *Neural Computation*, 1997, 9(7): 1545~1588.

[21] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Analysis and*

Machine Intelligence, Aug. 1998, 20(8): 832∼844.
[22] Dietterich T G. Ensemble Methods in Machine Learning. First International Workshop on Multiple Classifier Systems, Kittler J, Roli F (eds.), New York: Springer Verlag, 2000, pp.1∼15.
[23] Dietterich T G. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 2000, 40(2): 139∼157.
[24] Leo Breiman. Random forests. *Machine Learning*, 2001, 45(1): 5∼32.
[25] Liu T F, Ting K M, Fan W. Maximizing tree diversity by building complete-random decision trees. In *Proc. the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, May 2005, pp.605∼610.
[26] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[27] Blake C, Keogh E, Merz C. UCI repository of machine learning databases, 1998, http://www.ics.uci.edu/~mlearn/MLRepository.html.

Xue-Gang Hu received his B.S. degree from the Department of Mathematics at Shandong University, China, and his M.S. and Ph.D. degrees in computer science from Hefei University of Technology, China. Currently, he is a professor and a vice dean of the School of Computer Science and Information Engineering, Hefei University of Technology, China. Prof. Hu is engaged in research on data mining and knowledge engineering.

Pei-Pei Li is a graduate student at Hefei University of Technology, China. Her research interests are in data mining and knowledge engineering in general, and streaming data mining in particular.

Xin-Dong Wu is a professor and the chair of the Department of Computer Science at the University of Vermont, USA. He holds a Ph.D. degree in artificial intelligence from the University of Edinburgh, Britain. His research interests include data mining, knowledge-based systems, and Web information exploration. He has published extensively in these areas in various journals and conferences, including IEEE TKDE, TPAMI, ACM TOIS, DMKD, KAIS, IJCAI, AAAI, ICML, KDD, ICDM, and WWW, as well as 14 books and conference proceedings. Dr. Wu is the Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering (by the IEEE Computer Society), the founder and current Steering Committee Chair of the IEEE International Conference on Data Mining (ICDM), an Honorary Editor-in-Chief of Knowledge and Information Systems (by Springer), and a Series Editor of the Springer Book Series on Advanced Information and Knowledge Processing (AI&KP). He was Program Committee Chair for ICDM'03 (the 2003 IEEE International Conference on Data Mining) and is Program Committee Co-Chair for KDD-07 (the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining). He is the 2004 ACM SIGKDD Service Award winner, the 2006 IEEE ICDM Outstanding Service Award winner, and a 2005 Chaired Professor in the Cheung Kong (or Yangtze River) Scholars Programme at the Hefei University of Technology sponsored by the Ministry of Education of China and the Li Ka Shing Foundation. He has been an invited/keynote speaker at numerous international conferences including PAKDD-07, IEEE EDOC'06, IEEE ICTAI'04, IEEE/WIC/ACM WI'04/IAT'04, SEKE 2002, and PADD-97.

Gong-Qing Wu received his M.S. degree from the Department of Computer Science and Technology at the University of Science and Technology of China in 2003. He is currently a lecturer in the School of Computer Science and Information Engineering at Hefei University of Technology, China. His research interests are in data mining and data streams.