Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Clustering feature decision trees for semi-supervised classification from high-speed data streams*

Wen-hua XU†1, Zheng QIN†‡2, Yang CHANG2

(1Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

(2School of Software, Tsinghua University, Beijing 100084, China)

†E-mail: {xwh07, zhqing}@mails.tsinghua.edu.cn

Received Sept. 25, 2010; Revision accepted Mar. 9, 2011; Crosschecked July 4, 2011

Abstract: Most stream data classification algorithms apply the supervised learning strategy which requires massive labeled data. Such approaches are impractical since labeled data are usually hard to obtain in reality. In this paper, we build a clustering feature decision tree model, CFDT, from data streams having both unlabeled and a small number of labeled examples. CFDT applies a micro-clustering algorithm that scans the data only once to provide the statistical summaries of the data for incremental decision tree induction. Micro-clusters also serve as classifiers in tree leaves to improve classification accuracy and reinforce the any-time property. Our experiments on synthetic and real-world datasets show that CFDT is highly scalable for data streams while generating high classification accuracy with high speed.

Key words: Clustering feature vector, Decision tree, Semi-supervised learning, Stream data classification, Very fast decision tree
doi:10.1631/jzus.C1000330 **Document code:** A **CLC number:** TP391

1 Introduction

Historically, machine learning has concentrated on learning models from small numbers of examples. More recently, the need to process larger amounts of data has motivated the field of data mining. Nowadays, we are faced with a tremendous number of data streams from sensor networks, social networks, Web applications, scientific experiments, and financial activities, etc. In most cases, such data are massive, temporally ordered, fast changing, and potentially infinite (Domingos and Hulten, 2000). Mining from data streams is a research topic of growing interest.

Most approaches in classifying data streams assume that a major portion of the data streams are labeled, which can be used in training to build classification models; the remaining small amount of unlabeled data can then be classified with the models. However, data streams are not inherently accompanied by class labels in real-world applications. They are usually labeled manually. Such operations are both costly and time consuming. Therefore, in a streaming environment, labeled examples that can be used in training may be very scarce, leading to poorly trained classifiers. In comparison with the supervised algorithms that use only labeled examples, one can hope for a more accurate prediction by taking into account the unlabeled examples. Semi-supervised learning from data streams has emerged as an exciting direction in machine learning research.

In a streaming environment the algorithms run under time and space constraints. With the fast and continuous supply of data, the algorithms could not obtain all data at the beginning of training. Moreover, the algorithms have no control over the order of the data seen, so they must update incrementally whenever an example is inspected. In other words, the fresh knowledge derived from new examples must be incorporated without repeating the entire training

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 60673024) and the "Eleventh Five" Preliminary Research Project of PLA (No. 102060206)

processes. Therefore, the fundamental problem to solve in the context of data streams is that: given an infinite amount of continuous data examples, how to model them in order to capture their trends and patterns, and make time-critical predictions (Wang *et al*., 2003).

With regard to semi-supervised learning, suppose that the examples of each class tend to form a cluster; therefore, the unlabeled data could assist in finding the boundary of each cluster. The motivation behind this approach is the cluster assumption. If two examples are in the same cluster, they are likely to be of the same class (Chapelle *et al*., 2006). This assumption should be reasonable on the basis of the sheer existence of classes. If there is a densely populated continuum of objects, it may seem unlikely that they are ever distinguished into different classes. Moreover, the assumption does not imply that each class forms a single compact cluster. It addresses that usually one does not observe objects of two distinct classes in the same cluster. With more examples in a cluster, labeled or unlabeled, the boundary can be more accurate, so is the classification model.

Motivated by the assumption, we propose CFDT, an acronym for the clustering feature decision trees, learning from data streams having both unlabeled and a small amount of labeled examples. Very fast decision tree (VFDT) is one of the most successful and prominent algorithms designed specifically for supervised classification of data streams. CFDT applies an efficient semi-supervised clustering algorithm on VFDT, and is able to scan the data only once to provide the statistical summaries of the data in the form of micro-clusters for incremental decision tree induction.

Micro-clusters also serve as classifiers in tree leaves to improve classification accuracy and reinforce the any-time property. The classification is performed with the decision tree to seek an appropriate leaf and then the nearest neighbor (NN) algorithm in the leaf.

We have made three contributions. First, we propose an efficient semi-supervised classification algorithm for data streams based on the micro-clustering technique. Second, we provide an extension to the state-of-the-art VFDT algorithm with the ability to deal with numeric attributes for data streams. Third, we apply more powerful classifiers in leaves of VFDT, which can improve its accuracy and reinforce its any-time property on data streams. Evaluations on both synthetic and real-world datasets comprising up to one million examples show that our approach can achieve better classification accuracy than other stream classification approaches, using only a fraction of the labeled examples.

2 Related works

A classification algorithm for data streams must meet several different requirements from the traditional setting (Bifet *et al*., 2009). The most significant are the following. First, process one example at a time, and inspect it at most once. The data examples flow in and out of a system one after another. Each example must be accepted in the order in which it arrives. Once inspected or ignored, the example is discarded with no way to retrieve it. Second, use a limited amount of memory. Memory will be easily exhausted without limiting its allocation since the amount of the data is potentially infinite. Third, work in a limited amount of time. Though most conventional algorithms are fast enough when classifying examples, the training processes are time consuming. For an algorithm to scale comfortably to any number of examples, its training complexity must be linear to the number of examples, such that online learning is possible. Fourth, be ready to perform classification at any time. This is the so-called any-time property, which indicates that the induction model is ready to be applied at any point between training examples.

Our work is related to both decision tree induction algorithms and semi-supervised classification techniques on data streams. We briefly discuss both of them.

2.1 Decision tree induction algorithms on data streams

Decision tree is one of the most often used techniques in the data mining literature. Each node of a decision tree contains a test on an attribute. Each branch from a node corresponds to a possible outcome of the test and each leaf contains a class prediction. A decision tree is constructed by recursively replacing leaves by test nodes, starting at the root. The attribute to test in a leaf is chosen by comparing all available attributes and choosing the best one according to some heuristic evaluation function. Classic decision tree learners like ID3, C4.5, and CART assume that all training examples can be stored simultaneously in memory, and thus are severely limited in the number of examples from which they can learn.

Previous works on scaling up decision tree learning algorithms are SLIQ (Mehta *et al*., 1996), SPRINT (Shafer *et al*., 1996), BOAT (Gehrke *et al*., 1999), and RAINFOREST (Gehrke *et al*., 2000). These approaches perform static learning of decision trees from large data sources in limited memory by performing multiple passes over the data and using external storage. Such operations are not suitable for high-speed stream processing.

Domingos and Hulten (2000) proposed an incremental decision tree algorithm called the Hoeffding tree. For streams made up of discrete types of data, Hoeffding bounds guarantee that the output model is asymptotically nearly identical to that of a conventional decision tree. The Hoeffding tree algorithm is the basic theoretical algorithm and represents current state-of-the-art for classifying high-speed data streams. Its implementation is VFDT which adopts several enhancement techniques for practical applications, such as grace period, pre-pruning, and tie breaking.

Since the majority voting strategy used in the Hoeffding tree uses only a small part of the available information in leaves, Gama *et al*. (2003) proposed the VFDTc algorithm, which incorporates two main extensions to VFDT. These are the ability to deal with numeric attributes and apply naïve Bayes classifiers in tree leaves. In this way, there is a much better exploitation of the available information. VFDTc can obtain a performance similar to that of the C4.5 algorithm even on medium size datasets.

Furthermore, the CVFDT proposed by Hulten *et al*. (2001) induces a decision tree from concept drifting data streams based on VFDT. It can keep the latest knowledge by growing a sub-tree from new examples and replacing the old when the new becomes more accurate.

Hoeffding option tree (Pfahringer *et al*., 2007) is a regular Hoeffding tree containing extra option nodes besides the internal decision nodes and leaves. The structure makes it possible for an example to travel down multiple paths and arrive at multiple leaves. An adaptive Hoeffding option tree (Bifet *et al*., 2009) is a Hoeffding option tree with several improvements. Each leaf stores an estimation of the current classification error and the weight of each node in the major voting process which is proportional to the square of the inverse of the error. A Hoeffding perceptron tree (Bifet *et al*., 2010) is a Hoeffding tree that has a perceptron in each leaf.

Bifet *et al*. (2009) also proposed two new ensemble learning strategies, adaptive window bagging and adaptive size Hoeffding tree bagging, which combine Hoeffding trees into ensemble classifiers to achieve better accuracy.

In all the extensions mentioned above, the structure of the Hoeffding tree is not modified. Only some extra features are appended to the nodes, such as features to deal with numeric attributes, additional option nodes, and more sophisticated classifiers in leaves (e.g., naïve Bayes, perceptron). However, to build such classifiers from the examples in leaves is time consuming, especially when there are enormous leaves.

2.2 Semi-supervised classification algorithms on data streams

The supervised classification problem is generally defined as follows. A set of N_l training examples $S_l=\{(x_i, y_i)|i=1, 2, \dots, N_l\}$ of the form $(\boldsymbol{x}, y)$ is given, where $\boldsymbol{x}=(x_1, x_2, \dots, x_J)$ is a value vector of J-dimensional attributes, each of which may be discrete or numerical. The attributes are represented as $\boldsymbol{X}=(X_1, X_2, \dots, X_J)$. y_i is a nominal class label, $y_i\in\{c_1, c_2, \dots, c_K\}$. In addition to labeled examples, the algorithm of semi-supervised learning is provided with a set of unlabeled examples $S_u=\{(x_i, y_i)|i=N_l+1, N_l+2, \dots, N\}$, where $N>>N_l$ and $y_i=\varphi$ indicates the example is unlabeled. Therefore, the training examples are $S=S_l\cup S_u=\{(x_i, y_i)|i=1, 2, \dots, N\}$ and $y_i\in\{\varphi, c_1, c_2, \dots, c_K\}$. The semi-supervised learning in this work is to build a model $y=f(\boldsymbol{x})$ from these examples that can predict the class labels of future examples with high accuracy.

There have been several works in semi-supervised stream data classification, most of which are based on clustering techniques. The self-training strategy is adopted in the cluster-training algorithm, which applies k-prototype clustering to select confidently unlabeled examples and uses them to retrain the classifier iteratively (Wu *et al*., 2006). Though the unlabeled data can be fully exploited in cluster-training, the retraining operation is time consuming, which may be undesirable in a high speed streaming environment. SmSCluster is an ensemble classification model specialized for concept drifting data streams (Masud *et al*., 2008), which can be trained from a dataset having both unlabeled and a small number of labeled examples. Each base model in SmSCluster is built as micro-clusters using the semi-supervised *k*-means clustering technique, and classification is performed with the *k*-nearest neighbor algorithm. Such models are combined together to classify the unlabeled data. SmSCluster is a moderately practical approach to handling the real-world stream classification problem; yet the runtime of the expectation maximization approach used in clustering could be reduced. SUN is a semi-supervised classification algorithm for data streams with concept drifts (Li *et al*., 2010), based on an evolved decision tree. The *k*-mode clustering algorithm is developed to generate concept clusters in leaves of the decision tree to detect concept drifts, and an unlabeled example is labeled with the majority class of the cluster to which it belongs. Though unlabeled examples are exploited in SUN, they are used not for training the model but for detecting concept drifts. Therefore, only a small portion of the information obtained from them is used.

3 Very fast decision tree and clustering feature decision trees

3.1 Very fast decision tree

The Hoeffding tree algorithm is a classification model that constructs a decision tree model from data streams incrementally with no need to store them or repeat the entire induction process on all of the observed examples. The only information needed in memory is the tree itself, which stores sufficient statistics in its leaves in order to grow.

The Hoeffding tree is constructed by making recursive splits of leaves and subsequently obtaining internal decision nodes, such that a tree structure is formed. The splits are decided by heuristic evaluation functions to choose the best cut-point of an attribute. The main innovation in the Hoeffding tree is the use of a Hoeffding bound to decide how many examples are necessary to be collected for heuristic evaluation in each leaf. Assuming a real-valued random variable r whose range is R_a (e.g., for a probability, R_a=1; for an information gain, $R_\mathrm{a}=\log_2 K$, where K is the number of classes), suppose there are N independent observations of this variable whose mean value is $\bar{r}$. The Hoeffding bound indicates that, with probability $1-\delta$, the true mean of the variable is at least $\bar{r}-\varepsilon$, where

$$\varepsilon=\left[R_\mathrm{a}^2\ln(1/\delta)\big/(2N)\right]^{1/2}. \tag{1}$$

Let $H(\cdot)$ be a heuristic evaluation function, and assume H is maximized. Let X_a be the attribute with the highest $\bar{H}$ observed after observing N examples, and X_b be the second highest attribute. Let $\Delta\bar{H}=\bar{H}(X_\mathrm{a})-\bar{H}(X_\mathrm{b})$ be the difference between the two best observed heuristic values. Then, given a desired δ, the Hoeffding bound guarantees that X_a is the correct choice with probability $1-\delta$ if N examples have been observed in this node and $\Delta\bar{H}>\varepsilon$. That is, if the observed $\Delta\bar{H}>\varepsilon$, then the Hoeffding bound guarantees that the true value $\Delta H\geq\Delta\bar{H}-\varepsilon>0$ with probability $1-\delta$, and therefore X_a is indeed the best attribute with probability $1-\delta$. At this point, the leaf can be split using the current best attribute, and is transformed into an internal decision node.

Each leaf stores only the sufficient statistics about attribute values, instead of the whole example. When an example traverses to a leaf, the statistics at the leaf are updated, and the heuristic measure is evaluated to check whether or not to split the leaf. Assuming the heuristic function is the information gain, the statistics are the counts n_{ijk}, representing the number of examples of class c_k that reach the leaf, where the attribute X_j takes the value i. Then $H(X_j)$ for attribute X_j is calculated using

$$H(X_j)=\mathrm{Info(examples)}-\mathrm{Info}(X_j), \tag{2}$$

where Info(examples) is the information of the examples that reach the leaf, and

$$\mathrm{Info}(X_j)=-\sum_i P_i\left(\sum_k P_{ik}\log_2 P_{ik}\right) \tag{3}$$

is the information of attribute X_j. In Eq. (3),

$$P_{ik} = n_{ijk} \Big/ \sum_t n_{tjk} \tag{4}$$

is the probability of the observed value i of attribute X_j given class c_k, and

$$P_i = \sum_t n_{ijt} \Big/ \sum_s \sum_t n_{sjt} \tag{5}$$

is the probability of the observed value i of attribute X_j.

The Hoeffding tree algorithm uses the majority voting strategy in classification. When classifying an unlabeled test example, the example is filtered down to an appropriate leaf and classified with the majority class of the training examples in the leaf.

VFDT applies several extensions, such as the grace period, pre-pruning, and tie breaking, to enhance the basic Hoeffding tree algorithm.

3.2 Clustering feature decision trees

We present a decision tree induction algorithm CFDT, an acronym for the clustering feature decision tree for the semi-supervised learner, which is the extension of VFDT. By using micro-clustering and clustering feature techniques, CFDT has the ability of learning from data streams having both labeled and unlabeled examples. Micro-clusters are used in calculating and storing the statistics used in the heuristic evaluation function for splits of leaves, as well as in making classification on testing examples.

The micro-clustering technique was originally exploited by Zhang *et al*. (1996). The concept of ‘micro-cluster’ in this paper is similar to those in Zhang *et al*. (1996), Jin *et al*. (2001), and Yu *et al*. (2003), and denotes a statistically summarized representation of a group of examples with the same class label which are so close together that they are likely to belong to the same cluster.

3.3 Clustering feature vector

A clustering feature (CF) vector is a tuple that summarizes the sufficient information of a cluster. The concept of CF and its characteristic make the clustering incremental without expensive computations. Given N examples $\{\boldsymbol{x}_i|i=1, 2, \ldots, N\}$ in a cluster, the CF vector of the cluster is a tuple $\mathbf{CF}=(N, \mathbf{LS}, \mathbf{SS})$, where $\mathbf{LS}=\sum_{i=1}^{N}\boldsymbol{x}_i$ is the linear sum of the examples, and $\mathbf{SS}=\sum_{i=1}^{N}\boldsymbol{x}_i^2$ is the square sum of the examples (Zhang *et al*., 1996).

The additivity of the CF vector indicates that if $\mathbf{CF}_1=(N_1, \mathbf{LS}_1, \mathbf{SS}_1)$ and $\mathbf{CF}_2=(N_2, \mathbf{LS}_2, \mathbf{SS}_2)$ are the CF vectors of two disjoint clusters, the CF vector of the cluster formed by merging the two disjoint clusters is

$$\mathbf{CF}_1+\mathbf{CF}_2=(N_1+N_2, \mathbf{LS}_1+\mathbf{LS}_2, \mathbf{SS}_1+\mathbf{SS}_2). \tag{6}$$

The centroid $\boldsymbol{C}$ and radius R of the cluster are defined as follows:

$$\boldsymbol{C}=\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i, \tag{7}$$

$$R=\left(\frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{x}_i-\boldsymbol{C}\|^2\right)^{1/2}, \tag{8}$$

where R is the average distance from the cluster member to the centroid. A pure cluster is the cluster that contains only examples with the same class label and unlabeled examples.

From the CF definition and additivity characteristic, we know that the CF vector of a cluster can be calculated incrementally and accurately as new examples are incorporated. The centroid $\boldsymbol{C}$ and the radius R of a cluster can be calculated from the CF vector of the cluster using the following equations:

$$\boldsymbol{C}=\mathbf{LS}/N, \tag{9}$$

$$R=\left(\frac{1}{N}\sum_{j=1}^{J}\mathrm{SS}_j-\frac{1}{N^2}\sum_{j=1}^{J}(\mathrm{LS}_j)^2\right)^{1/2}. \tag{10}$$

One usually thinks of a cluster as a set of data points, but the CF vector is stored only as a statistical summary. Managing the CF vector is efficient and accurate, saves space significantly, and is sufficient for calculating all the information to build a decision tree. Therefore, the CF vector serves as the micro-cluster in this paper.

A CFDT is a decision tree with two parameters for leaves, the threshold $R_{\max}$ and the maximum number of micro-clusters W. $R_{\max}$ is a constraint for micro-clusters to ensure that the radius of each micro-cluster has to be less than $R_{\max}$. Each leaf of the CFDT consists of at most K entries of the form

[c_k, m-cluster$_k$], with K being the number of classes, and each entry consists of at most W micro-clusters whose members are from c_k or φ, i.e., m-cluster$_k$={CF$_{ik}$|i=1, 2, …, W}.

Each example is inspected by CFDT only once to obtain information. The process that an example ($\boldsymbol{x}$, y) is inserted into a micro-cluster of a leaf is as follows:

1. Identify an appropriate leaf. The example traverses the CFDT from the root to a leaf, testing the appropriate attribute at each internal node and following the branch corresponding to the attribute value of the example.

2. Seek an entry. Two situations should be considered. For a labeled example (i.e., $y\neq\varphi$), an entry is chosen whose class label is also y, and then a micro-cluster is selected whose centroid is the closest. For an unlabeled example (i.e., $y=\varphi$), a micro-cluster is selected whose centroid is the closest from all entries in the leaf. In this way, information obtained from labeled and unlabeled examples is absorbed by micro-clusters.

3. Modify the leaf. If the micro-cluster can incorporate the example without violating the threshold $R_{\max}$ condition, update the micro-cluster, i.e., the CF vector. If not, construct a new one for the example. Then the example is discarded to leave memory for future examples.

If we treat the example ($\boldsymbol{x}$, y) as a new cluster, the CF vector of this cluster is represented as $\mathbf{CF}_{\boldsymbol{x}}$. Adding an example to a cluster at the (r+1)th step is equivalent to merging two disjoint clusters. That is,

$$\begin{aligned}\mathbf{CF}_{r+1}&=\mathbf{CF}_r+\mathbf{CF}_x\\&=(N_r+N_x,\ \mathbf{LS}_r+\mathbf{LS}_x,\ \mathbf{SS}_r+\mathbf{SS}_x)\\&=(N_r+1,\ \mathbf{LS}_r+\boldsymbol{x},\ \mathbf{SS}_r+\boldsymbol{x}^2).\end{aligned}\quad(11)$$

After the entries of a leaf are constructed, examples in a micro-cluster can be substituted by the centroid of the cluster to simplify the calculation. Moreover, the numerical attribute values are discretized, so that the counts n_{ijk} can be derived directly. Note that a micro-cluster is excluded from the computation of the heuristic evaluation function if it contains no labeled examples.

The value of threshold $R_{\max}$ will greatly affect the number of micro-clusters in an entry. The initial value $R_{\max}^0$ can be set conservatively, which may produce more than W micro-clusters in an entry. $R_{\max}^0$ is determined at the very beginning of the model training process, which can be achieved using an offline process. A standard k-means clustering algorithm is employed on a batch of examples to create $W\cdot K$ clusters. Once these clusters have been established, the minimum value of the radii of all clusters can be assigned to $R_{\max}^0$. When new blank leaves are constructed, $R_{\max}$ should be adjusted to produce an appropriate number of micro-clusters. We use $R_{\max}^{i+1}=R_{\max}^{i}\cdot(W_i/W)^{1/J}$, where W_i is the minimum number of micro-clusters among all entries in the previous leaf.

3.4 Heuristic evaluation function

Fig. 1 shows the structure of CFDT. In CFDT a leaf that contains a heuristic evaluation function for continuous attributes creates two descendant leaves. For an attribute X_j of the vector $\boldsymbol{x}$, which is numerical, the heuristic evaluation function is a condition of the form X_j<cutp$_j$ to determine if the attribute value x_j of an example $\boldsymbol{x}$ is less than cutp$_j$ or not. Then the cut-point cutp$_j$ divides the training dataset into two subsets, which correspond to the values TRUE (subset A) and FALSE (subset B) for the function. What needs to be determined is how to split subsets of $\boldsymbol{X}$ to produce the best tree-structured classifier. CFDT uses the recursive binary partition strategy.

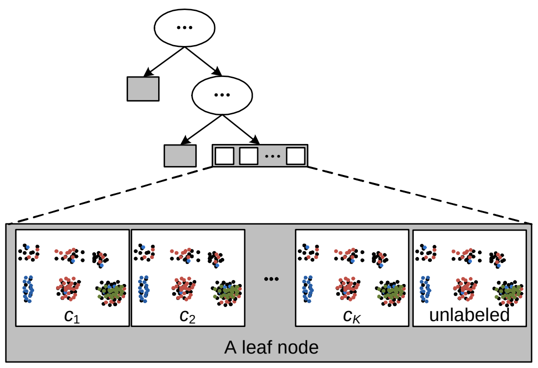

Fig. 1 Structure of the clustering feature decision trees (CFDT) model

At each stage in the recursive split, we use an exhaustive method. All attributes $\boldsymbol{X}$ of the examples and all possible cut-points for each attribute are evaluated, and the class distributions at both sides of the cut-point are calculated.

To calculate the class distribution of the examples, for each possible cut-point, we compute the information of the two partitions using the following heuristic function (Gama *et al.*, 2003):

$$\begin{aligned}\text{Info}(X_j^t) &= P(X_j < \text{cutp}_{jt}) \cdot \text{Less}(X_j^t) \\ &\quad + P(X_j > \text{cutp}_{jt}) \cdot \text{Great}(X_j^t),\end{aligned} \tag{12}$$

where $\text{Info}(X_j^t)$ is the information for attribute X_j and cutp$_{jt}$, cutp$_{jt}$ is the *t*th possible cut-point for attribute X_j, $\text{Less}(X_j^t)$ is the information of X_j<cutp$_{jt}$, and $\text{Great}(X_j^t)$ is the information of X_j>cutp$_{jt}$, which are calculated by

$$\begin{aligned}\text{Less}(X_j^t) &= -\sum_{i=1}^{K}\Big[P(y=c_i|X_j<\text{cutp}_{jt}) \\ &\quad \cdot \log_2 P(y=c_i|X_j<\text{cutp}_{jt})\Big],\end{aligned} \tag{13}$$

$$\begin{aligned}\text{Great}(X_j^t) &= -\sum_{i=1}^{K}\Big[P(y=c_i|X_j>\text{cutp}_{jt}) \\ &\quad \cdot \log_2 P(y=c_i|X_j>\text{cutp}_{jt})\Big].\end{aligned} \tag{14}$$

The fundamental variables for calculating all these necessary statistics are the counts n_{ijk}, which can be derived directly from the micro-clusters in leaves as presented above.

Without loss of generality, we consider cases where each entry has an equal number of W micro-clusters for convenience of explanation. Suppose [c_k, m-cluster$_k$] is an entry in a leaf corresponding to class c_k, and m-cluster$_k$={CF$_{ik}$|i=1, 2, …, W}. Therefore, the centroids of the micro-clusters are {$\boldsymbol{C}_{ik}$|i=1, 2, …, W}, and the numbers of examples in the micro-clusters are {N_{ik}|i=1, 2, …, W}. Since examples in a micro-cluster can be substituted by the centroid of the cluster, the number of examples that have the value $\boldsymbol{C}_{ik}$ is N_{ik}. Moreover, since we have $\boldsymbol{C}_{ik}=(c_{i1k}, c_{i2k}, \ldots, c_{ijk}, \ldots, c_{iJk})$ where c_{ijk} is the attribute value of $\boldsymbol{C}_{ik}$ corresponding to attribute X_j, the number of examples that have the value c_{ijk} is N_{ik}, i.e., $n_{ijk}=N_{ik}$.

To calculate the class distribution of the examples in a leaf, the attribute values of the centroids of all the micro-clusters need to be sorted for every attribute. The operation is time saving since the number of centroids is much smaller than that of examples, which are J groups with $W \cdot K$ examples in each group.

Letting $c_{j(1)}<c_{j(2)}<\ldots<c_{j(W\cdot K)}$ be the ordered distinct values of $c_{1j1}, c_{2j1}, \ldots, c_{Wj1}, \ldots, c_{1j2}, \ldots, c_{Wj2}, \ldots, c_{WjK}$ in a leaf, then the *t*th possible cut-point can be

$$\text{cutp}_{jt} = \left(c_{j(t)} + c_{j(t+1)}\right)\big/2, \quad t = 1, 2, \ldots, WK-1. \tag{15}$$

Therefore, even if there are a large number of examples, there are a finite number of possible cut-points to be considered.

Let $n_{j(1)}, n_{j(2)}, \ldots, n_{j(WK)}$ be the numbers of examples corresponding to $c_{j(1)}, c_{j(2)}, \ldots, c_{j(WK)}$, and N be the total number of examples in the leaf. Therefore,

$$P(X_j < \text{cutp}_{jt}) = \frac{1}{N}\sum_{i=1}^{t} n_{j(i)}, \tag{16}$$

$$P(y=c_i|X_j<\text{cutp}_{jt}) = \sum_{\substack{d=1\\ y=c_i}}^{t} n_{j(d)} \Bigg/ \sum_{\substack{d=1\\ y=c_i}}^{WK} n_{j(d)}. \tag{17}$$

$H(X_j^t)$ for attribute X_j at cutp$_{jt}$ is calculated as

$$H(X_j^t) = \text{Info(examples)} - \text{Info}(X_j^t). \tag{18}$$

Therefore, we can use the same method as in the induction of the Hoeffding tree to choose the appropriate attribute and the correct cut-point.

3.5 Discrete attributes

We create one independent ‘1–0’ feature for each value of the discrete attributes such that ‘1’ indicates the existence of the value and ‘0’ indicates the absence of the value. The method for dealing with discrete attributes is not the optimal one for building a decision tree or clustering, but a general one for categorical attributes, discrete ordinal attributes, and ratio-scaled attributes. Using more sophisticated approaches could further improve the performance, yet may take too much runtime for calculation to be suitable for classification of data streams.

3.6 Construction of CFDT

The complete algorithm of CFDT is presented in Algorithm 1. The construction process of a CFDT is similar to that of a VFDT. With the continuous supply of data stream examples, the first ones are used to choose the root test. Once the root attribute is chosen, the succeeding examples are passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on, recursively. Annotations are added to the pseudo code of the algorithm.

Algorithm 1 CFDT algorithm

```
Inputs:
$S$          a sequence of examples,
$\boldsymbol{X}$          a vector of attributes,
$H(\cdot)$       a heuristic evaluation function,
$\delta$          one minus the desired probability of choosing
             the correct attribute at any given node,
$R_{\max}$       the initial threshold,
$W$          the maximum number of micro-clusters.
Output:
$T$          a decision tree.
 1 Procedure CFDT($S$, $\boldsymbol{X}$, $H(\cdot)$, $\delta$, $R_{\max}$ , $W$)
   // Start from a tree with one blank root
 2   Let $T$ be a tree with a root;
 3   /* build the tree by making recursive splits of
     leaves and subsequently obtaining internal
     decision nodes */
     for each example ($\boldsymbol{x}$, $y$) in $S$     // $y=c_k$ or $y=\varphi$
     /* filter the example into a leaf and assign it to a
     cluster */
 4     Filter the example into a leaf $l$ using $T$;
 5     if $y=c_k$
 6       Seek a proper entry having the same class
         label and a closest micro-cluster to incorporate the example;
 7     end if
 8     if $y=\varphi$
 9       Seek a closest micro-cluster to incorporate
         the example;
10     end if
11     if the micro-cluster cannot absorb the example
12       Construct a new micro-cluster;
13     end if
14     // update the statistics of the cluster
       Update the CF vector of the micro-cluster in
       the entry;
15     /* If a specific number of examples are accumulated, try to calculate the heuristic function
       and find the best cut-point. $N_l$ is the number of
       examples seen at leaf $l$ */
       if ($N_l$ mod $n_{\min}$=0) and the examples seen so far
       at $l$ are not all of the same class
16       Sort the attribute values of the centroids of
         all the micro-clusters for every attribute;
17       Compute $\bar{H}_l(X_j^t)$ for each attribute and each
         possible cut-point using Eqs. (12) and (18);
18       Compute the Hoeffding bound;
19       Let $X_{\rm a}$ be the attribute with the highest $\bar{H}_l$ ;
20       Let $X_{\rm b}$ be the attribute with the second
         highest $\bar{H}_l$ ;
21       /* If the condition of split is satisfied, turn
         the leaf into an internal node, and add new
         leaves to it */
         if $\bar{H}(X_{\rm a})-\bar{H}(X_{\rm b})>\varepsilon$ or $\varepsilon<\tau$
22         Replace $l$ with an internal node that splits
           on $X_{\rm a}$ and the corresponding cut-point;
23         Adjust $R_{\max}$ using
           $R_{\max}^{i+1}=R_{\max}^{i}\cdot(W_i/W)^{1/J}$ ;
24         for the two branches of the split
25           Add a new leaf with initialized sufficient statistics;
26         end for
27       end if
28     end if
29   end for
30 end Procedure
```

3.7 Functional tree leaves

One of the innovations of our algorithm is that the current micro-clusters in tree leaves can be reused and serve as classification models. The majority class strategy of VFDT uses only the information about class distributions and does not consider the attribute values. It uses only a small part of the available information, a crude approximation to the distribution of the examples. In contrast, micro-clusters of CFDT take into account not only the class distribution, but also the attribute values given the class. By this method, there is a much better exploitation of the available information in leaves.

To predict the class label of a testing example $\boldsymbol{x}$, the example traverses the tree from the root to a leaf, testing the appropriate attribute at each internal node and following the branch corresponding to the attribute value of the example. After $\boldsymbol{x}$ falls into a leaf, two cases are considered. For a leaf having only one entry [c_k, m-cluster$_k$], c_k is the predicted label for $\boldsymbol{x}$. For a leaf having different entries, the nearest neighbor algorithm is applied to find the nearest micro-cluster from all micro-clusters in the leaf and the corresponding class label is the predicted value for $\boldsymbol{x}$.

4 Experimental evaluation

We applied CFDT on both synthetic datasets and real-world datasets to evaluate its performance in six aspects. These were the ability to handle numeric attributes, accuracy in semi-supervised classification, memory cost, training and classification speed, scalability, and sensitivity to parameters. We compare

CFDT with the supervised baseline method Hoeffding Tree Naïve Bayes Adaptive (HT-NBA) and the semi-supervised method SmSCluster.

4.1 Datasets and experiment setup

The synthetic datasets include GAUSS, Hyperplane, Random RBF, Random Tree, SEA, and Waveform. The approach to generating the GAUSS dataset follows the way in Masud *et al*. (2008). In this work, the number of class labels is 5, and the number of attributes is 15. Examples belonging to each class are generated by following a normal distribution having a different mean within (−5.0, 5.0), and variance within (0.5, 9) for different classes. Hyperplane was used to compare CVFDT with VFDT under the environment of concept drift, where the number of attributes can be assigned by users (Hulten *et al*., 2001). Random RBF generates examples from a fixed number of random centroids. Each time a centroid is selected at random, a data point around the centroid is drawn randomly. The centroid determines the class label of the example, and the data point determines the attributes of the example. Examples in Random Tree are generated by assigning random values to each attribute first and the class label is determined via a pre-constructed decision tree (Domingos and Hulten, 2000). SEA generates examples with three attributes and two class labels, where the first two attributes are relevant (Street and Kim, 2001). For the waveform dataset, there are two versions in massive online analysis (MOA). The first has 21 numeric attributes and 3 class labels. The second introduces 19 additional irrelevant attributes, which can be viewed as noises. The real-world datasets include Forest Covertype and KDD-99. The Forest Covertype dataset is taken from the UCI repository. It contains 581 012 examples, 54 attributes, and 7 class labels. The classification goal is to predict the forest cover type from cartographic variables. The KDD-99 training dataset is from the UCI KDD archive, consisting of 41 attributes, 23 class labels, and approximately five million examples. We use 10% (i.e., 494 021) examples of KDD-99, and 34 numeric attributes in our evaluation. All attribute values are normalized into [0, 1], divided by their maximum values. We assume that no concept drift exists in these datasets. The basic properties of each dataset and the initial $R_{\max}$ for every dataset are listed in Table 1.

Table 1 Properties of the datasets

Dataset	n_{nom}	n_{num}	K	$R^0_{\max}$
GAUSS	0	15	5	13.0
Hyperplane	0	10	2	1.1
Random RBF	0	10	2	1.8
Random Tree	0	10	2	1.1
SEA	0	3	2	4.0
WAVE21	0	21	3	7.0
WAVE40	0	40	3	8.5
Covertype	44	10	7	60.0
KDD-99	0	34	23	15.0

n_{nom}: number of nominal attributes; n_{num}: number of numeric attributes; K: number of classes

MOA (Bifet *et al*., 2007) is a software environment for stream data mining, including evaluation measures and a collection of implemented Hoeffding tree based algorithms, such as VFDT and HT-NBA. The CFDT algorithm was integrated into MOA. SmSCluster was implemented in RapidMiner, which is another open source data mining system. The experiments were performed on a 2.0 GHz Intel Core Duo PC with 2 GB RAM, running Windows XP. Parameter settings are: τ=0.05, $n_{\min}$=1000, $\delta=5\times10^{-6}$ for both HT-NBA and CFDT, W=8 for CFDT.

4.2 Comparison of approaches handling numeric attributes

Several approaches handling numeric attributes when constructing Hoeffding trees have been proposed in the data stream literature. Examples include VFML (very fast machine learning) implementation (Hulten and Domingos, 2003), Exhaustive Binary Tree (Gama *et al*., 2003), Quantile Summaries (Greenwald and Khanna, 2001), and Gaussian Approximation (Pfahringer *et al*., 2008). Pfahringer *et al*. (2008) investigated these methods and compared them empirically to eight algorithm configurations. From the results of the experimental comparison, GAUSS-10 and VFML-10 are highly competitive on most testing configurations. Since CFDT has the natural ability to deal with numeric attributes by using the micro-clustering technique, we compare it against two representative methods, Gaussian Approximation and VFML implementation, to evaluate its performance.

GAUSS-10 and VFML-10 are tested based on the HT-NBA classification algorithm. HT-NBA contains Naïve Bayes learners in the leaves, making it more accurate than the basic Hoeffding Tree Majority Class algorithm. Seven synthetic datasets are used in the evaluation, and learning is limited to a maximum memory of 32 MB. We use a holdout evaluation strategy similar to that in Pfahringer *et al*. (2008). Each classifier is trained for 1 hr and then testing is conducted on a holdout dataset comprising up to 10^6 examples. All training datasets in this subsection are 100% labeled.

Tables 2–4 list the experiment results of the three algorithms on seven synthetic datasets. Under the predefined setting, GAUSS-10 is the most accurate on all datasets, and builds the deepest trees with the most nodes and leaves amongst the three algorithms. There are two reasons for this. First, GAUSS-10 is faster than the other two algorithms, which means that GAUSS-10 can learn more examples under the same training time constraint. Second, GAUSS-10 needs less space for numeric approximation since only a few statistics are stored in the leaves, permitting the growth with a maximum number of active tree nodes in memory. CFDT and VFML-10 achieve similar accuracy in most cases but smaller accuracy than GAUSS-10, while the training speed of CFDT is a little higher than that of VFML-10 except on Random Tree. Without iteration, micro-clustering is much more efficient than conventional clustering methods. This makes CFDT fast enough to handle high-speed data streams. CFDT tends to create more nodes than VFML-10, suggesting that fewer examples are needed to determine a split point during the tree growth. The depths of CFDT are on average smaller than those of the other two algorithms, suggesting that the splits on all attributes are even, not on some specific attributes.

According to space complexity analysis, to store sufficient statistics of J numeric attributes and K class labels, the memory required for each leaf is $5JK$ for GAUSS-10, and $10J+10JK$ for VFML-10. Each leaf of the CFDT consists of K entries of [c_k, m-cluster$_k$], each entry consists of W micro-clusters, and each micro-cluster needs $1+2J$ for storage of N, **LS**, and **SS**. To store statistics of W micro-clusters, the memory required for CFDT is $K(1+W(1+2J))$, that is $(W+1)K+2WJK$. Given W=8, the memory cost is $9K+16JK$. Considering its ability for semi-supervised classification, the space complexity is acceptable.

Table 2 GAUSS-10 algorithm over seven datasets

Dataset	Accuracy (%)	n_{te} ($\times10^6$)	n_{tn}	n_{al}	n_{il}	Tree depth
GAUSS	99.98	132	4267	2134	0	37
Hyperplane	92.01	42	31215	12686	2922	21
Random RBF	95.36	52	24923	12462	0	44
Random Tree	99.96	202	4599	2300	0	18
SEA	89.92	72	55919	27960	0	28
WAVE21	85.44	38	16435	5866	2352	31
WAVE40	85.15	28	11105	3134	2419	33

n_{te}: number of training examples; n_{tn}: number of total nodes; n_{al}: number of active leaves; n_{il}: number of inactive leaves

Table 3 VFML-10 algorithm over seven datasets

Dataset	Accuracy (%)	n_{te} ($\times10^6$)	n_{tn}	n_{al}	n_{il}	Tree depth
GAUSS	99.71	70	2917	1459	0	25
Hyperplane	86.21	36	20001	2995	7006	25
Random RBF	93.94	38	14181	3049	4042	28
Random Tree	99.97	152	1631	816	0	23
SEA	89.86	42	34199	9426	7674	31
WAVE21	82.93	34	9515	1389	3439	23
WAVE40	82.82	30	7077	734	2805	20

n_{te}: number of training examples; n_{tn}: number of total nodes; n_{al}: number of active leaves; n_{il}: number of inactive leaves

Table 4 CFDT algorithm over seven datasets

Dataset	Accuracy (%)	n_{te} ($\times10^6$)	n_{tn}	n_{al}	n_{il}	Tree depth
GAUSS	99.92	52	6341	3171	0	18
Hyperplane	87.35	40	27415	8144	5564	16
Random RBF	92.77	42	23809	6033	5872	22
Random Tree	97.75	58	7897	3949	0	18
SEA	87.91	54	35213	13652	3955	15
WAVE21	82.58	38	12373	3021	3166	17
WAVE40	82.17	34	8897	1926	2523	17

n_{te}: number of training examples; n_{tn}: number of total nodes; n_{al}: number of active leaves; n_{il}: number of inactive leaves

4.3 Classification accuracy, runtime, and memory usage

In this subsection, we compare CFDT with supervised baseline method HT-NBA with GAUSS-10 as the numeric handler and semi-supervised method SmSCluster. When evaluating the performance, all training datasets for semi-supervised CFDT and SmSCluster are 20% labeled, meaning that 20% of the examples selected at random are assumed to have class labels, whereas the remaining 80% examples are unlabeled. For supervised HT-NBA, we use the same number of training examples for comparison, which are all labeled. Parameters in SmSCluster are set to the default values, which are: Clusters=50 (number of clusters), ChunkSize=5000 (number of examples in each chunk), L=1 (ensemble size) represents the single classifier, and L=8 represents the ensemble classifier. The evaluation strategy is interleaved test-then-train. Each example is used for testing the model before being used for training. It assures that the model is always being tested on examples it has not seen. Seven synthetic datasets comprising up to 10^6 examples and two real-world datasets are used in the evaluation. We disable memory management during the training to evaluate the memory requirements.

Table 5 is the summary of the experiment results, where three metrics of accuracy, training and classification time, and memory usage are compared. Amongst all the algorithms, HT-NBA is apparently the most accurate, efficient, and space saving over all datasets. However, as a supervised method, it achieves 2.13% higher accuracy on average than CFDT by learning four times more labeled examples.

CFDT has comparable accuracy to HT-NBA and ensemble SmSCluster. There are three reasons for this. First, with a great amount of unlabeled data involved in training, more information can be learned from them, which makes the boundary of examples from different classes more accurate. Second, by using a micro-clustering approach, CFDT can group examples not only by their class labels or by the value of a single attribute, but also by their overall similarity. Therefore, the count n_{ijk} in CFDT is more accurate and representative than that in VFDT. Then the heuristic evaluation function derived from n_{ijk} can choose more appropriate attributes and cut-points. Third, the micro-cluster classifiers in leaves ensure a better exploitation of the available information of examples compared with the majority voting strategy in VFDT.

CFDT also has moderately lower evaluation runtime and memory requirements than ensemble SmSCluster on all datasets. The runtime of single SmSCluster is close to that of ensemble SmSCluster, yet the accuracy is much lower. The most time consuming operation in SmSCluster is creating clusters using the EM approach, especially the operation where data points in one cluster are reassigned to others in the expectation step. On average, a data chunk having 2000 examples requires approximately 14 EM iterations to converge (Masud *et al*., 2008). In CFDT, however, there is no iteration when creating micro-clusters. The number of micro-clusters in a leaf of CFDT is not fixed. This is greatly affected by threshold $R_{\max}$. If the number of micro-clusters in one

Table 5 Comparison of accuracy, training and classification time, and memory usage among the algorithms

Algorithm	Accuracy (%)								
	GAUSS	Hyper	R-RBF	R-Tree	SEA	WAVE21	WAVE40	CovT	KDD-99
HT-NBA	99.85	89.39	88.76	97.42	89.01	83.09	82.85	80.45	99.19
CFDT	98.63	86.68	86.27	94.97	87.84	80.90	81.27	76.35	97.86
SmSC-S	98.94	65.73	66.13	78.59	65.54	61.90	55.76	65.44	85.15
SmSC-E	99.82	85.69	86.23	95.43	88.78	85.42	84.75	75.77	95.32
Algorithm	**Evaluation time (s)**								
	GAUSS	Hyper	R-RBF	R-Tree	SEA	WAVE21	WAVE40	CovT	KDD-99
HT-NBA	14.48	14.02	15.01	8.60	5.23	31.21	61.23	36.63	33.41
CFDT	20.87	19.09	27.47	18.77	12.65	69.73	143.57	73.91	65.12
SmSC-S	730.00	703.00	6957.00	684.00	402.00	2420.00	4280.00	3090.00	2670.00
SmSC-E	815.00	758.00	7385.00	811.00	536.00	2709.00	4900.00	3750.00	2910.00
Algorithm	**Memory (KB)**								
	GAUSS	Hyper	R-RBF	R-Tree	SEA	WAVE21	WAVE40	CovT	KDD-99
HT-NBA	271	925	1060	292	368	1169	2073	1152	154
CFDT	932	1469	2227	688	673	2731	3540	1846	780
SmSC-S	11937	13794	14735	9473	5230	21584	23716	18304	11046
SmSC-E	19868	22781	26379	16561	6871	29473	31050	24281	14384

SmSC-S: SmSCluster (single); SmSC-E: SmSCluster (ensemble). Hyper: Hyperplane; R-RBF: Random RBF; R-Tree: Random Tree; CovT: Covertype leaf is eventually larger than W, $R_{\max}$ is adjusted so that the number of micro-clusters in the next leaf will be smaller. Moreover, micro-clusters represented by CF vectors can be calculated incrementally and efficiently, so the approach also ensures a low runtime. Third, all numerical attribute values of an example can be discretized simultaneously by finding an appropriate micro-cluster. This operation can also greatly reduce runtime. For the above reasons, the training and classification time of CFDT can be much lower than that of SmSCluster.

Fig. 2 shows the learning curves of four algorithms on the Random RBF dataset. When more examples are learned, Hoeffding tree based algorithms can slowly improve their accuracy, while SmSCluster remains almost constant if there is no concept drift in datasets. This is because the ensemble strategy always keeps limited numbers of base models in memory and discards the previous useful models.

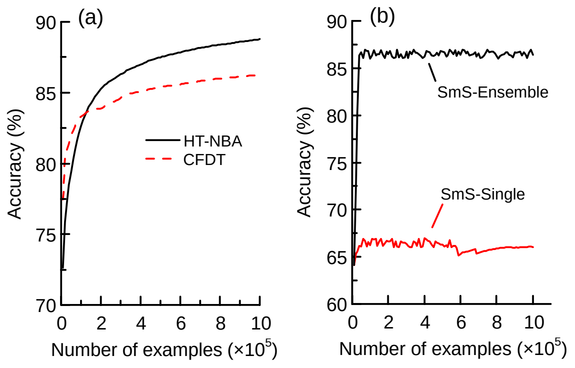

Fig. 2 Learning curves of HT-NBA and CFDT (a) and SmSCluster-Single and SmsCluster-Ensemble (b) on the Random RBF dataset

4.4 Runtime and scalability

Figs. 3 and 4 report the scalability of CFDT on high-dimensional and multi-class data. The size of the tree including internal nodes and leaves and runtime on the GAUSS dataset comprising 10^6 examples are measured. The tree size curves for a different number of attributes (J) with a different number of classes (K) are plotted in Fig. 3. We observe that a higher value of K leads to a larger size of tree. This could occur because when K is larger, $\text{Less}(X_j^t)$ and $\text{Great}(X_j^t)$ tend to be larger according to Eqs. (13) and (14), and consequently the value of $\text{Info}(X_j^t)$ is enlarged according to Eq. (12), as are the values of $H(X_j^t)$ from Eq. (18) and $\Delta\bar{H}$ from Eq. (2). Therefore, $\Delta\bar{H} > \varepsilon$ can be true when fewer examples are involved in training. Since fewer examples are used in one leaf split, more splits can be made with a fixed number of training examples, leading to a decision tree with more internal nodes and leaves. However, the size of the tree keeps almost invariable with different values of J for a particular value of K. This is due to the fact that increasing J does not change the values of $\text{Less}(X_j^t)$ and $\text{Great}(X_j^t)$. The number of examples needed for one leaf split is not changed either, resulting in a constant size of the tree.

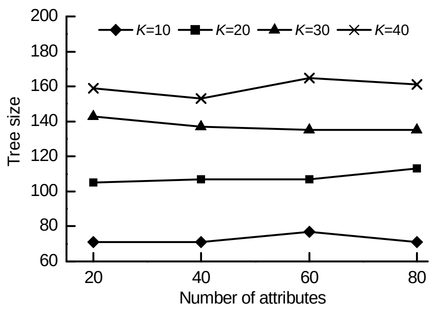

Fig. 3 The tree size curves for a different number of attributes with a different number of classes (K) on the GAUSS dataset

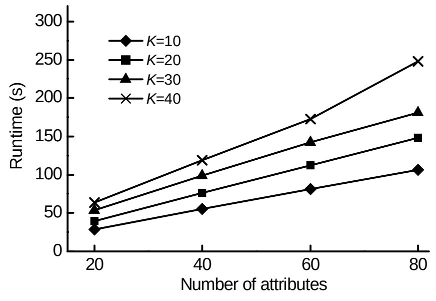

Fig. 4 Runtime curves for a different number of attributes with a different number of classes (K) on the GAUSS dataset

Fig. 4 shows how the runtime varies with the number of attributes and the number of classes. For example, the runtime is 76.14 s when J=40 and K=20. The runtime increases linearly with the number of attributes. This is because the runtimes of constructing CF vectors, computing the heuristic evaluation function, and classifying unlabeled examples are all linear with the number of attributes. Also, note that the runtime increases almost linearly with the number of classes. This is because the runtimes of constructing CF vectors, computing the heuristic evaluation function, and classifying unlabeled examples are also all linear with the number of class labels. This is desirable. Therefore, we can conclude that CFDT scales linearly to higher dimensionality and higher percentage of class labels.

4.5 Sensitivity to parameters

Fig. 5 shows how the classification accuracy varies for CFDT with the percentage of labeled data (*P*) in the dataset and the number of micro-clusters (*W*). The results are obtained from the GAUSS dataset comprising 10^6 examples with 10 class labels and 20 attributes. Higher values of *W* lead to higher accuracies. This is due to the fact that when *W* is larger, micro-clusters are getting smaller and purer, leading to an accurate computation of the heuristic evaluation function for leaf split and finer-grained classifiers in leaves. However, there is no significant improvement for accuracy after *W* reaches 10. The accuracy improves with an increasing number of labeled data in the dataset. The reason is evident. With more labeled data, the boundary of each micro-cluster can be more accurate, resulting in the improvement of the performance.

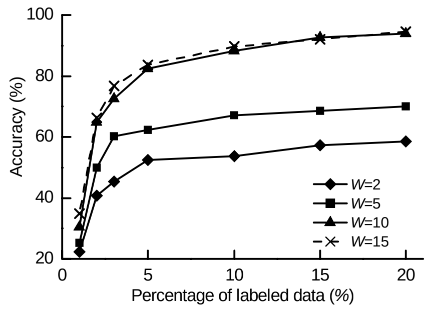

Fig. 5 Sensitivity to the percentage of labeled data in the dataset and the number of micro-clusters (*W*)

5 Conclusions

This paper presents a new algorithm called CFDT that integrates a scalable micro-clustering method into the state-of-the-art algorithm VFDT and runs effectively and efficiently for semi-supervised classification of data streams. CFDT also incorporates another two extensions to VFDT. These are the ability to deal with numeric attributes and apply more elaborate classifiers in leaves. CF vectors which serve as micro-clusters are the basic structure in CFDT besides the decision tree, and play different roles at different phases. Micro-clusters in leaves are constructed to discretize numeric attributes and store the sufficient statistics of examples for calculating the heuristic evaluation function at the training phase. Moreover, unlabeled examples are assigned into micro-clusters based on the similarity to the centroids of clusters, enabling knowledge driven by them to be learned. Whenever a testing example arrives, the nearest neighbor algorithm is applied on the micro-clusters in a leaf to obtain the predicted class label at the classification phase. By efficiently constructing micro-clusters and with no extra constructing classifiers in leaves, CFDT can be very fast. Our experiments on synthetic and real-world datasets show that CFDT is a very competitive algorithm in accuracy, training and classification time, and scalability.

Mining from concept drifting data streams is another core issue (Hulten *et al.*, 2001). Ensembles of classifiers are the most ideal and promising methods for this. Several ensemble strategies on VFDT have been proposed (Bifet *et al.*, 2009). Since CFDT maintains all the desirable properties of VFDT, it can be incorporated directly into such strategies.

References

Bifet, A., Kirkby, R., Holmes, G., Pfahringer, B., 2007. MOA: Massive Online Analysis. Available from http://moa.cs.waikato.ac.nz/ [Accessed on Jan. 31, 2010].

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavalda, R., 2009. New Ensemble Methods for Evolving Data Streams. Proc. 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.139-148. [doi:10.1145/1557019.1557041]

Bifet, A., Holmes, G., Pfahringer, B., Frank, E., 2010. Fast perceptron decision tree learning from evolving data streams. *LNCS*, **6119**:299-310. [doi:10.1007/978-3-642-13672-6_30]

Chapelle, O., Scholkopf, B., Zien, A., 2006. Semi-supervised Learning. MIT Press, Cambridge, USA, p.5.

Domingos, P., Hulten, G., 2000. Mining High-Speed Data Streams. Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.71-80. [doi:10.1145/347090.347107]

Gama, J., Rocha, R., Medas, P., 2003. Accurate Decision Trees for Mining High-Speed Data Streams. Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.523-528. [doi:10.1145/956750.956813]

Gehrke, J., Ganti, V., Ramakrishnan, R., Loh, W., 1999. BOAT—Optimistic Decision Tree Construction. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.169-180. [doi:10.1145/304182.304197]

Gehrke, J., Ramakrishnan, R., Ganti, V., 2000. RainForest—a framework for fast decision tree construction of large datasets. *Data Min. Knowl. Disc.*, **4**(2/3):127-162. [doi:10.1023/A:1009839829793]

Greenwald, M., Khanna, S., 2001. Space-Efficient Online Computation of Quantile Summaries. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.58-66. [doi:10.1145/375663.375670]

Hulten, G., Domingos, P., 2003. VFML—a Toolkit for Mining High-Speed Time-Changing Data Streams. Available from http://www.cs.washington.edu/dm/vfml [Accessed on Apr. 25, 2010].

Hulten, G., Spencer, L., Domingos, P., 2001. Mining Time-Changing Data Streams. Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.97-106. [doi:10.1145/502512.502529]

Jin, W., Tung, A.K.H., Han, J., 2001. Mining Top-*n* Local Outliers in Large Databases. Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.293-298. [doi:10.1145/502512.502554]

Li, P., Wu, X., Hu, X., 2010. Learning from Concept Drifting Data Streams with Unlabeled Data. Proc. 24th AAAI Conf. on Artificial Intelligence, p.1495-1496.

Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B., 2008. A Practical Approach to Classify Evolving Data Streams: Training with Limited Amount of Labeled Data. Proc. 8th IEEE Int. Conf. on Data Mining, p.929-934. [doi:10.1109/ICDM.2008.152]

Mehta, M., Agrawal, R., Rissanen, J., 1996. SLIQ: a fast scalable classifier for data mining. *LNCS*, **1057**:18-32. [doi:10.1007/BFb0014141]

Pfahringer, B., Holmes, G., Kirkby, R., 2007. New options for Hoeffding trees. *LNCS*, **4830**:90-99. [doi:10.1007/978-3-540-76928-6_11]

Pfahringer, B., Holmes, G., Krikby, R., 2008. Handling Numeric Attributes in Hoeffding Trees. Proc. 12th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, p.296-307. [doi:10.1007/978-3-540-68125-0_27]

Shafer, J.C., Agrawal, R., Mehta, M., 1996. SPRINT: a Scalable Parallel Classifier for Data Mining. Proc. 22nd Int. Conf. on Very Large Data Bases, p.544-555.

Street, W.N., Kim, Y., 2001. A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification. Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.377-382.

Wang, H., Fan, W., Yu, P.S., Han, J., 2003. Mining Concept-Drifting Data Streams Using Ensemble Classifiers. Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.226-235. [doi:10.1145/956750.956778]

Wu, S., Yang, C., Zhou, J., 2006. Clustering-Training for Data Stream Mining. Proc. 6th IEEE Int. Conf. on Data Mining, p.653-656. [doi:10.1109/ICDMW.2006.45]

Yu, H., Yang, J., Han, J., 2003. Classifying Large Data Sets Using SVMs with Hierarchical Clusters. Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.306-315. [doi:10.1145/956750.956786]

Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: an Efficient Data Clustering Method for Very Large Databases. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.103-114. [doi:10.1145/235968.233324]