6

Incremental Optimization Mechanism for Constructing a Balanced Very Fast Decision Tree for Big Data

Hang Yang and Simon Fong

Introduction

Big data is a popular topic that highly attracts attentions of researchers from all over the world. How to mine valuable information from such huge volumes of data remains an open problem. Although fast development of hardware is capable of handling much larger volumes of data than ever before, in the author's opinion, a well-designed algorithm is crucial in solving the problems associated with big data. Data stream mining methodologies propose one-pass algorithms that are capable of discovering knowledge hidden behind massive and continuously moving data. Stream mining provides a good solution for such big data problems, even for potentially infinite volumes of data.

H. Yang (✉)
China Southern Power Grid,
Guangzhou, Guangdong, China

S. Fong
Department of Computer and Information Science,
University of Macau, Macau SAR, Zhuhai Shi, China

L. Moutinho, M. Sokele (eds.), *Innovative Research Methodologies in Management*,
https://doi.org/10.1007/978-3-319-64394-6_6

Decision tree learning is one of the most significant classifying techniques in data mining and has been applied to many areas, including business intelligence, health-care and biomedicine, and so forth. The traditional approach to building a decision tree, designed by greedy search, loads a full set of data into memory and partitions the data into a hierarchy of nodes and leaves. The tree cannot be changed when new data are acquired, unless the whole model is rebuilt by reloading the complete set of historical data together with the new data. This approach is unsuitable for unbounded input data such as data streams, in which new data continuously flow in at high speed. A new generation of algorithms has been developed for incremental decision tree, a pioneer of which using a Hoeffding bound (*HB*) in node-splitting is so called Very Fast Decision Tree (VFDT) (Pedro and Geoff 2000), which can build a decision tree simply by keeping track of the statistics of the attributes of the incoming data. When sufficient statistics have accumulated at each leaf, a node-splitting algorithm determines whether there is enough statistical evidence in favor of a node-split, which expands the tree by replacing the leaf with a new decision node. This decision tree learns by incrementally updating the model while scanning the data stream on the fly. In the past decade, VFDT has been extended to some improved algorithms, inheriting the use of HB (see in section "Background"). This powerful concept is in contrast to a traditional decision tree that requires the reading of a full dataset for tree induction. The obvious advantage is its real-time mining capability, which frees it from the need to store up all of the data to retrain the decision tree because the moving data streams are infinite.

On one hand, the challenge for data stream mining is associated with the imbalanced class distribution. The term "imbalanced data" refers to irregular class distributions in a dataset. For example, a large percentage of training samples may be biased toward class *A*, leaving few samples that describe class *B*. Both noise and imbalanced class distribution significantly impair the accuracy of a decision tree classifier through confusion and misclassification prompted by the inappropriate data. The size of the decision tree will also grow excessively large under noisy data. To tackle these problems, some researchers applied data manipulation techniques to handle the imbalanced class distribution problems, including under-sampling, resampling, a recognition-based induction scheme

(Nitesh et al. 2004), and a feature subset selection approach (Mladenic and Grobelnik 1999).

On the other hand, despite the difference in their tree-building processes, both traditional and incremental decision trees suffer from a phenomenon called over-fitting when the input data are infected with noise. The noise confuses the tree-building process with conflicting instances. Consequently, the tree size becomes very large and eventually describes noise rather than the underlying relationship. With traditional decision trees, the under-performing branches created by noise and biases are commonly pruned by cross-validating them with separate sets of training and testing data. Pruning algorithms (Elomaa 1999) help keep the size of the decision tree in check; however, the majority are post-pruning techniques that remove relevant tree paths after a whole model has been built from a stationary dataset. Post-pruning of a decision tree in high-speed data stream mining, however, may not be possible (or desirable) because of the nature of incremental access to the constantly incoming data streams. Incremental optimization seeks for solutions that evolve over time in response to environmental changes. In general, there are three performance metrics for incremental problems: *ratio*, *sum*, and *demand* (Hartline 2008). *Ratio metric* uses a worst-case measurement to determine the distance between the optimal solution and the solution made at each time step. *Sum metric* is the expected value metric over all time steps. A weight function while summing solution values can easily settle the problem of natural bias for late-stage solution. *Demand metric* is a decision metric measuring the degree of specific quantitative requirements satisfaction.

VFDT handles streaming data that tree structure keeps on updating when new data arrive. It only requires reading some samples satisfying the statistical bound (referring to the HB) to construct a decision tree. Since it cannot analyze over the whole training dataset in one time, normal optimization methods using full dataset to search for an optima between the accuracy and tree size do not work well here.

Our previous work has provided a solution for sustainable prediction accuracy and regulates the growth of the decision tree to a reasonable extent, even in the presence of noise. Moderated Very Fast Decision Tree (MVFDT) is a novel extension of the VFDT model (Yang and Fong 2011) that includes optimizing the tree-growing process via *adaptive tie-breaking threshold* instead of a user pre-defined value in VFDT.

In this chapter, for optimizing VFDT, we devise a new version, so called optimized VFDT (OVFDT), which can provide an incremental optimization on prediction accuracy, tree size, and learning time. The contributions of OVFDT are: (1) it contains four types of functional tree leaf that improve the classification accuracy; (2) it inherits the mechanism of MVFDT that uses an adaptive tie-breaking threshold instead of a user pre-defined. To this end, it may suit for the aforementioned real applications; (3) it contains an incremental optimization mechanism in the node-splitting test that obtains an optimal tree structure as a result. By running simulation experiments, the optimized value of adaptive tie is proved to be ideal for constraining the optimal tree growth.

Background

Decision Tree in Data Stream Mining

A decision tree classification problem is defined as follows: N is a set of examples of the form (X, y), where X is a vector of d attributes and y is a discrete class label. k is the index of class label. Suppose a class label with the k_{th} discrete value is y_k. Attribute X_i is the i_{th} attribute in X, and is assigned a value of $x_{i1}, x_{i2}\ldots x_{ij}$, where $1 \leq i \leq d$ and J is the number of different values X_i. The classification goal is to produce a decision tree model from N examples, which predicts the classes of y in future examples with high accuracy. In data stream mining, the example size is very large or unlimited, $N \rightarrow \infty$.

VFDT algorithm (Pedro and Geoff 2000) constructs an incremental decision tree by using constant memory and constant time-per-sample. VFDT is a pioneering predictive technique that utilizes the Hoeffding bound. The tree is built by recursively replacing leaves with decision nodes. Sufficient statistics n_{ijk} of attribute X_i with a value of x_{ij} are stored in each leaf with a class label assigning to a value y_k. A heuristic evaluation function $H(\cdot)$ is used to determine split attributes for converting leaves to nodes. Nodes contain the split attributes and leaves contain only the class labels. The leaf represents a class according to the sample label. When a sample enters, it traverses the tree from the root to a leaf, evaluating the relevant attributes at

every node. Once the sample reaches a leaf, the sufficient statistics are updated. At this time, the system evaluates each possible condition based on the attribute values; if the statistics are sufficient to support one test over the others, then a leaf is converted to a decision node. The decision node contains the number of possible values for the chosen attribute according to the installed split test. The main elements of VFDT include, first, a tree-initializing process that initially contains only a single leaf and, second, a tree-growing process that contains a splitting check using a heuristic function $H(\cdot)$ and Hoeffding bound (HB). VFDT uses information gain as $H(\cdot)$.

The formula of HB is shown in (6.1). HB controls over errors in the attribute-splitting distribution selection, where R is the range of classes' distribution and n is the number of instances that have fallen into a leaf. To evaluate a splitting-value for attribute X_i, it chooses the best two values. Suppose x_{ia} is the best value of $H(\cdot)$ where $x_{ia} = \arg\max H(x_{ij})$; suppose x_{ib} is the second best value where $x_{ib} = \arg\max H(x_{ij}), \forall j \neq a$; suppose $\Delta H(X_i)$ is the difference of the best two values for attribute X_i, where $\Delta H(X_i) = \Delta H(x_{ia}) - \Delta H(x_{ib})$. Let n be the observed number of instances, HB is used to compute high confidence intervals for the true mean r_{true} of attribute x_{ij} to class y_k that $r - \mathrm{HB} \leq r_{\mathrm{true}} < r + \mathrm{HB}$ where $r = (1/n)\sum_i^n r_i$.

If after observing $n_{\min}$ examples, the inequality $r + \mathrm{HB} < 1$ holds, then $r_{\mathrm{true}} < 1$, meaning that the best attribute x_{ia} observed over a portion of the stream is truly the best attribute over entire stream. Thus, a splitting-value x_{ij} of attribute X_i can be found without full attribute values even when we don't know all values of X_i. In other words, it does not train a model from full data and the tree is growing incrementally when more and more data come.

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

In the past decade, several research papers have proposed different methodologies to improve the accuracy of VFDT. Such incremental decision tree algorithms using HB in node-splitting test are so called Hoeffding Tree (HT). HOT (Pfahringer et al. 2007) proposes an algorithm producing

some optional tree branches at the same time, replacing those rules with lower accuracy by optional ones. The classification accuracy has been improved significantly while learning speed is slowed because of the construction of optional tree branches. Some of options are inactive branches consuming computer resource. Functional tree leaf is originally proposed to integrate to incremental decision tree in VFDTc (Gama et al. 2003). Consequently, Naïve Bayes classifier on the tree leaf has improved classification accuracy. The functional tree leaf is able to handle both continuous and discrete values in data streams, but no direct evidence shows it can handle such imperfections like noise and bias in data streams. FlexDT (Hashemi and Yang 2009) proposes a Sigmoid function to handle noisy data and missing values. Sigmoid function is used to decide what true node-splitting value, but sacrificing algorithm speed. For this reason, the lightweight algorithm with fast learning speed is favored for data streams environment. CBDT (Stefan et al. 2009) is a forest of trees algorithm that maintains a number of trees, each of which is rooted on a different attributes and grows independently. It is sensitive to the concept-drift in data streams according to the sliding-window mechanism. VFDR (Gama and Kosina 2011) is a decision rule learner using HB. The same as VFDT, VFDR proposes a rule-expending mechanism that constructs the decision rules (ordered or unordered) from data stream on the fly.

There are two popular platforms for implementing stream-mining decision tree algorithms. Very Fast Machine Learning (VFML) (Hulten and Domingos 2003) is a C-based tool for mining time-changing high-speed data streams. Massive Online Analysis (MOA) (Bifet et al. 2001) is Java-based software for massive data analysis, which is a well-known open source project extended from WEKA data mining. In both platforms, the parameters of VFDT must be pre-configured. For different tree induction tasks, the parameter setup is distinguished.

MOA is an open source project with a user-friendly graphic interface. It also provides several ways to evaluate algorithm's performance. Hence, some VFDT-extended algorithms have been built-in this platform. For example, the VFDT algorithms embedded in MOA (released on Nov. 2011) are: Ensemble Hoeffding Tree (Oza and Russell 2001) is an online bagging method with some ensemble VFDT classifiers. Adaptive Size Hoeffding Tree (ASHT) (Bifet et al. 2009) is derived from VFDT adding

a maximum number of split nodes. ASHT has a maximum number of split nodes. After one node splits, if the number of split nodes is higher than the maximum value, then it deletes some nodes to reduce its size. Besides, it is designed for handling concept-drift data streams AdaHOT (Bifet et al. 2009) is also derived from HOT. Each leaf stored an estimation of current error. The weight of node in voting process was proportional to the square of inverse of error. AdaHOT combines HOT with a voting mechanism on each node. It also extends the advantages using optional trees to replace the tree branches of bad performance. Based on an assumption "there has been no change in the average value inside the window", ADWIN (Bifet and Gavalda 2007) proposes a solution to detect changes by a variable-length window of recently seen instances. In this chapter, the OVFDT algorithm is developed on the fundamental of MOA platform. All experiments are also run on MOA platform.

Relationship Among Accuracy, Tree Size, and Time

When data contains noisy values, it may confuse the result of heuristic function. The difference of the best two heuristic evaluation for attribute X_i, where $\Delta\bar{H}\left(X_i\right) = H\left(x_{ia}\right) - H\left(x_{ib}\right)$, may be negligible. To solve this problem, a fixed tie-breaking τ, which is a user pre-defined threshold for incremental learning decision tree, is proposed as pre-pruning mechanism to control the tree growth speed (Hulten et al. 2001). This threshold constrains the node-splitting condition that $\Delta\bar{H}\left(X_i\right) \le \mathrm{HB} < \tau$. An efficient τ guarantees a minimum tree growth in case of tree size explosion problem. τ must be set before a new learning starts; however, so far there has not been a unique τ suitable for all problems. In other words, there is not a single default value that works well in all tasks so far. The choice of τ hence depends on the data and their nature. It is said that the excessive invocation of tie breaking brings the performance of decision tree learning declining significantly on complex and noise data, even with the additional condition by the parameter τ.

A proposed solution (Geoffrey et al. 2005) to overcome this detrimental effect is an improved tie-breaking mechanism, which not only considers the best (x_{ia}) and the second best (x_{ib}) splitting candidates in terms of

heuristic function but also uses the worst candidate (x_{ic}). At the same time, an extra parameter is imported, α, which determines how many times smaller the gap should be before it is considered as a tie. The attribute-splitting condition becomes: when $\alpha \times (H(x_{ia}) - H(x_{ib})) < (H(x_{ib}) - H(x_{ic}))$, the attribution x_{ia} shall be split as a node. Obviously, this approach uses two extra elements, α and x_{ic}, which bring extra computation to the original algorithm.

In addition to the tie-breaking threshold τ, $n_{\min}$ is the number of instances a leaf should observe between split attempts. In other words, τ is a user-defined value to control the tree-growing speed, and $n_{\min}$ is a user-defined value to control the interval time to check node-splitting. The former is used to constrain tree size and the latter is used to constrain the learning speed. In order to optimize accuracy, tree size, and speed for decision tree learning, first of all, an example is given for demonstrating the relationship among these three factors for data streams.

In this example, the testing datasets are synthetic added bias class. We use MOA to generate the tested datasets. *LED24* is the nominal data structure and *Waveform21* is the numeric data structure. Both datasets share the origins with the sample generators donated by UCI machine learning repository. *LED24* problem uses 24 binary attributes to classify 10 different classes. The goal of *Waveform21* task is to differentiate between three different classes of waveform, each of which is generated from a combination of two or three base waves. It has 21 numeric attributes. The data stream problem is simulated by large number of instances, which are as many as one million for both datasets. The accuracy, tree size, and time are recorded with changing the pre-defined values of τ and $n_{\min}$. From Table 6.1, we can see that:

- In general, the bigger tree size brings a higher accuracy, even caused by the over-fitting problem, but taking more learning time.
- τ is proposed to control the tree size growing. A bigger τ brings a faster tree size growth, but longer computation time. But because the memory is limited, the tree size does not increase while τ reaches a threshold ($\tau = 0.7$ for LED24; $\tau = 0.4$ for Waveform21).
- $n_{\min}$ is proposed to control the learning time. A bigger $n_{\min}$ brings a faster learning speed, but smaller tree size and lower accuracy.

Table 6.1 Comparison of VFDT using different τ and n_{min}

τ	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**	**1.00**
LED24 (n_{min} = 200)										
Accuracy (%)	75.88	76.97	77.14	77.42	77.47	77.50	77.56	77.56	77.56	77.56
#Leaf	143.00	522.00	1124.00	1857.00	2618.00	3723.00	3743.00	3743.00	3743.00	3743.00
Time (sec)	8.70	9.91	10.92	11.51	11.92	12.78	12.32	12.32	12.43	12.45
Waveform21 (n_{min} = 200)										
Accuracy	85.00	86.53	86.61	86.72	86.72	86.72	86.72	86.72	86.72	86.72
#Leaf	506.00	1565.00	2492.00	2623.00	2623.00	2623.00	2623.00	2623.00	2623.00	2623.00
Time	17.80	18.50	18.89	18.69	18.77	18.72	18.53	18.72	18.74	18.72
n_{min}	**200**	**300**	**400**	**500**	**600**	**700**	**800**		**900**	**1000**
LED24 (τ = 0.7)										
Accuracy	77.5611	77.5867	77.4565	77.3472	77.2557	77.1417	77.1412		77.0847	76.9887
#Leaf	3743	2405	1826	1383	1244	1057	935		804	689
Time	11.66887	10.93567	10.40527	10.07766	9.469261	9.42246	9.032458		9.172859	8.642455
Waveform21 (τ = 0.4)										
Accuracy	86.7218	86.5226	86.3028	85.9499	85.9119	85.6378	85.6707		85.7318	85.2165
#Leaf	2623	1800	1363	1103	940	806	703		644	572
Time	18.31452	17.70611	17.34731	17.03531	16.84811	16.58291	16.61411		16.61411	16.2709

However, the only way to detect the best tie-breaking threshold for a certain task is trying all the possibilities in VFDT. It is impractical for real-world applications. In this chapter, we propose the adaptive tie-breaking threshold using the incremental optimization methodology. The breakthrough of our work is the optimized node-splitting control, which will be specified in the following sections.

Incrementally Optimized Decision Tree

Motivation and Overview

OVFDT, which is based on the original VFDT design, is implemented on a test-then-train approach (Fig. 6.1) for classifying continuously arriving data streams, even for infinite data streams. The whole test-then-train process is synchronized such that when the data stream arrives, one segment at a time, the decision tree is being tested first for prediction output and training (which is also known as updating) of the decision tree then occurs incrementally. The description of testing process will be explained in section "OVFDT Testing Approach" in detail, and the training process will be explained in section "OVFDT Training Approach". Ideally, the node-splitting test updates the tree model in order to improve the accuracy, while a bigger tree model takes longer computation time. The situation to do the node-splitting check

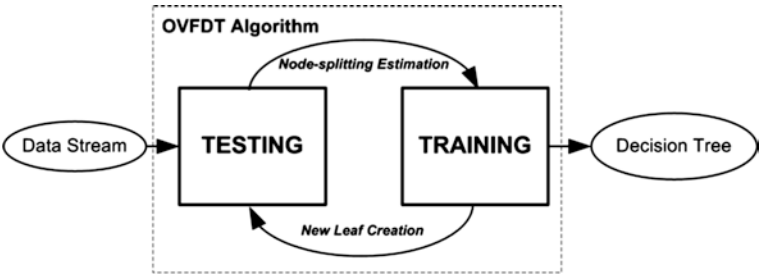

Fig. 6.1 A test-then-train OVFDT workflow

is when the number of instances in a leaf l is greater than the predefined value $n_{\min}$.

Imperfect data streams, including noisy data and bias class distribution, decline the performance of VFDT. Figure 6.2 shows the results of accuracy, tree size, and computation time using VFDT, the same dataset structure added with imperfect values. The ideal stream is free from noise and has a uniform proportion of class samples, which is rare in real world. From the experiment result comparing ideal data streams to imperfect data streams, we conclude lemma 1:

Lemma 1 Imperfections in data streams worsen the performance of VFDT. The tree size and the computation time are increased, but the accuracy is declined. In other words, the optimization goal is to increase the accuracy but not enlarge the tree size, within an acceptable computation time. Naturally a bigger tree size takes longer computation time. For this reason, the computation time is dependent on the tree size.

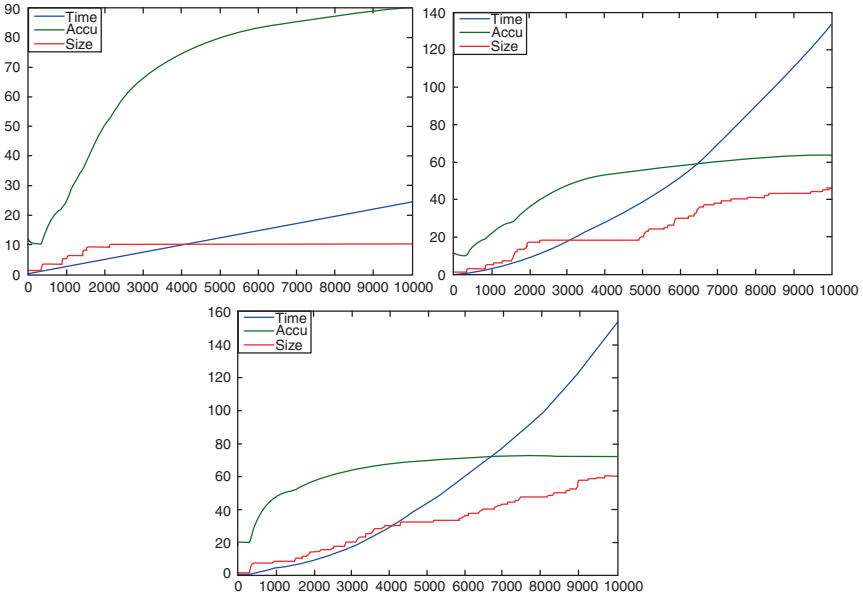

Fig. 6.2 VFDT performance for: (**a**) ideal data, (**b**) data with noise, (**c**) data with noise and bias. X-axis presents the accuracy and y-axis the number of samples

In the decision tree model, each path from the root to a leaf is considered as a way to present a rule. To ensure a high accuracy, there must be sufficient number of rules, which is the number of leaves in the tree model. Suppose Hoeffding Tree (HT) is the decision tree algorithm using Hoeffding bound (HB) as the node-splitting test. Let $\mathrm{Accu}\left(\mathrm{HT}_{m^{th}}\right)$ be the accuracy function for the decision tree structure HT at the *mth* node-splitting estimation, and let $\mathrm{Size}\left(\mathrm{HT}_{m^{th}}\right)$ be the tree size, then:

$$\mathrm{Accu}\left(\mathrm{HT}_{m^{th}}\right) = R\left(\mathrm{Size}\left(\mathrm{HT}_{m^{th}}\right)\right)$$

where $R(\cdot)$ is a mapping function of tree size to accuracy. Most incremental optimization functions can be expressed as the sum of several sub-objective functions:

$$\Phi\left(x\right) = \sum_{m=1}^{M} \Phi_m\left(x\right)$$

where $\Phi_m : \chi \subset \mathbb{R}^p \to \mathbb{R}$ is a continuously differentiable function whose domain χ is a nonempty, convex and closed set. We consider the following optimization problems:

$$\mathrm{maximize}\ \Phi\left(x\right) \mathrm{subject}\ to\ x \in \chi$$

Based on Lemma 1, we propose a solution to optimize the decision tree structure by improving the original VFDT that:

$$\Phi_m\left(x\right) = \frac{\mathrm{Accu}\left(\mathrm{HT}_{m^{th}+1}\right) - \mathrm{Accu}\left(\mathrm{HT}_{m^{th}}\right)}{\mathrm{Size}\left(\mathrm{HT}_{m^{th}+1}\right) - \mathrm{Size}\left(\mathrm{HT}_{m^{th}}\right)}$$

The tree model is updated when a node-splitting appears. Original VFDT considers the HB as the only index to split node. However, it is not enough. In terms of the above optimization goal, OVFDT

proposes an optimized node-splitting control during the tree-building process.

OVFDT Test-Then-Train Process

Data streams are open-ended problem that traditional sampling strategies are not viable in the non-stopping streams scenario. OVFDT is an improved version of the original VFDT and its extensions using *HB* to decide the node-splitting. The most significant contribution is OVFDT that can obtain an optimal tree structure by balancing the accuracy and tree size. It is useful for data mining especially in the events of the tree size explosion, when the decision tree is subject to imperfect streams including noisy data and imbalanced class distribution.

HT algorithms run a test-then-train approach to build a decision tree mode. When new stream arrives, it will be sorted from the root to a predicted leaf. Comparing the predictive class to the true class of this data stream, we can maintain an error matrix for every tree leaf in the testing process. In terms of the stored statistics matrix, the decision tree model is being updated in the training process. Table 6.2 presents the differences between OVFDT and HT algorithms (including the original VFDT and its extensions). Figure 6.3 shows the input parameters and the output of OVFDT and the approach presented as pseudo code.

Table 6.2 The comparison between VFDT and OVFDT

Approach	Hoeffding Tree algorithms	OVFDT
Testing	Sort the new stream by current HT	Sort the new stream by current HT
	Update the sufficient statistics	Update the sufficient statistics
	Construct FTL, by MC, NB, or ADP classifier	Construct FTL, by MC, NB, WNB, or ADP classifier
	Assign a predicted class by FTL	Assign a predicted class by FTL
		Update incremental sequential-error
Training	Check node-splitting by HB	Check node-splitting by HB
	Check node-splitting by fixed τ	Check node-splitting by adaptive τ
	HT update	Check node-splitting by incremental sequential-error
		HT update

INPUT:
S : A stream of sample
X : A set of symbolic attributes
G(.) : Heuristic function using for node-splitting estimation
δ : One minus the desired probability of choosing a correct attribute at any given node
n_{min} : The minimum number of samples between check node-splitting estimation
$\mathcal{F}$: A functional tree leaf strategy
OUTPUT:
HT : A decision tree

PROCEDURE: ***OVFDT****(S, X, G(.), δ, n_{min}, F)*
1. A data stream *S* arrives
2. IF *HT* is null, THEN ***initializeHT****(S, X, G(.), δ, n_{min}, F)* *ELSE* ***traverseHT****(S, HT, F)* and update *ΔC*
3. Label *l* as the predicted class among the samples seen so far at the leaf *l*.
4. Let n_l be the number of samples seen at the leaf *l*.
5. IF the samples seen so far at leaf *l* do not all belong to the same class, and (n_l mod n_{min}) is zero, THEN ***doNodeSplittingEstimation****(ΔC, S, X, G(.), δ, n_{min})*
6. Return *HT*

Fig. 6.3 Pseudo code of input and the test-then-train approach

OVFDT Testing Approach

Suppose X is a vector of d attributes, and y is the class with k different values included in the data streams. For decision tree prediction learning tasks, the learning goal is to induce a function of $\hat{y}_k = HT_{\mathcal{F}}(X)$, where $\hat{y}_k$ is the predicted class by Hoeffding Tree (HT) according to a functional tree leaf strategy F. When a new data stream (X, y_k) arrives, it traverses from the root of the decision tree to an existing leaf by the current decision tree structure, provided that the root has existed initially. Otherwise, the heuristic function is used to constructs a tree model with a single root node.

When new instance comes, it will be sorted from the root to a leaf by the current tree model. The classifier on the leaf can further enhance the prediction accuracy via the embedded Naïve Bayes classifier. OVFDT embed four different classifiers F to improve the performance of prediction. They are *Majority Class* (F^{MC}), *Naïve Bayes* (F^{NB}), *Weighted Naïve Bayes* (F^{WNB}) and *Adaptive* (F^{Adaptive}).

Suppose $\hat{y}_k$ the predicted class value and y_k is actual class in data streams with a vector of attribute X. A sufficient statistics matrix stores the number of passed-by samples, which contain attribute X_i with a value x_{ij} belonging to a certain y_k so far. We call this statistics table Observed Class Distribution (OCD) matrix. The size of OCD is $J \times K$, where J is the total number of distinct values for attribute X_i and K is the number of distinct class values. Suppose n_{ijk} is the sufficient statistic that reflects the number of attribute X_i with a value x_{ij} belonging to class y_k. Therefore, OCD on node X_i is defined as:

$$\mathrm{OCD}_{X_i} = \begin{bmatrix} n_{i11} & \cdots & n_{iJ1} \\ \vdots & \ddots & \vdots \\ n_{i1K} & \cdots & n_{iJK} \end{bmatrix}$$

For a certain leaf that attribute X_i with a value of x_{ij}:

$$\mathrm{OCD}_{x_{ij}} = \left\{ n_{ij1} \ \ldots \ n_{ijK} \right\}$$

Majority Class classifier chooses the class with the maximum value as the predicted class in a leaf. Thus, F^{MC} predicts the class with a value that:

$$\arg\max k = \left\{ n_{ij1} \quad \ldots \quad n_{ijk} \quad \ldots \quad n_{ijK} \right\}$$

Naïve Bayes classifier chooses the class with the maximum possibility computed by *Naïve Bayes*, as the predictive class in a leaf. The formula of *Naïve Bayes* is:

$$p_{ijk} = \frac{\mathrm{P}\left(x_{ij} | y_k\right) \cdot \mathrm{P}\left(y_k\right)}{\mathrm{P}\left(x_{ij}\right)}$$

OCD of leaf with value x_{ij} is updated incrementally. Thus, F^{NB} predicts the class with a value that:

$$\arg\max k = \left\{ p_{ij1} \quad \ldots \quad p_{ijk} \quad \ldots \quad p_{ijK} \right\}$$

Weighted Naïve Bayes classifier proposes to reduce the effect of imbalanced class distribution. It chooses the class with the maximum possibility computed by weighted *Naïve Bayes*, as the predictive class in a leaf:

$$p_{ijk} = \omega_{ijk} \frac{\mathrm{P}\left(x_{ij}|y_k\right) \cdot \mathrm{P}\left(y_k\right)}{\mathrm{P}\left(x_{ij}\right)} \text{ where } \omega_{ijk} = \frac{n_{ijk}}{\sum_{k=1}^{K} n_{ijk}}$$

OCD of leaf with value x_{ij} is updated. Thus, F^{WNB} predicts the class with a value that:

$$\arg\max k = \left\{ p_{ij1} \quad \ldots \quad p_{ijk} \quad \ldots \quad p_{ijK} \right\}$$

Adaptive classifier chooses the classifier with the least error from the alternative F^{MC}, F^{NB} and F^{WNB}. For each time classifier is implemented on the leaf, suppose γ is the index of classifier implementation on leaf assigned to x_{ij}, and suppose Γ is the total number of implementation, where $\Gamma = \sum_{k=1}^{K} n_{ijk}$. The error of a classifier F to class y_k is calculated by:

$$\mathrm{Err}\left(F, y_k\right) = \sum_{\gamma=1}^{\Gamma} \mathrm{Error}_k^{\gamma}, \text{where } \mathrm{Error}_k^{\gamma} = \begin{cases} 1, \text{if } \hat{y}_k \neq y_k \\ 0, \text{otherwise} \end{cases}$$

Therefore, F^{Adaptive} predicts the class with a value that is chosen by the classifier F with minimum error:

$$\arg\min F = \left\{ \mathrm{Err}\left(F^{\mathrm{MC}}, y_k\right), \quad \mathrm{Err}\left(F^{\mathrm{NB}}, y_k\right), \quad \mathrm{Err}\left(F^{\mathrm{WNB}}, y_k\right) \right\}$$

After the stream traverses the whole HT, it is assigned to a predicted class $\hat{y}_k$, which $\hat{y}_k \leftarrow Classifier(HT, \mathbb{F}, X)$ according to the functional tree leaf F. Comparing the predicted class $\hat{y}_k$ to the actual class y_k, the statistics of correctly C_T and incorrectly C_F prediction are updated immediately. Meanwhile, the sufficient statistics n_{ijk}, which is a count of attribute x_i with value j belongs to class y_k, are updated in each node. This series of actions is so called a testing approach in this chapter. Figure 6.4 gives the pseudo code of this approach. According to the functional tree leaf strategy, the current HT sorts a newly arrived sample (X, y_k) from the

PROCEDURE: ***traverseHT(****S, HT, F****)***

1. Sort S from the root to a leaf by HT. Update OCD in each node: $n_{ijk}(l)$ ++
2. Switch (F)
3. Case F^{MC}: predict the class y'_k with max $n_{ijk}(l)$
4. Case F^{NB}: predict the class y'_k with max NB prob.
5. Case F^{WNB}: predict the class y'_k with max WNB prob.
6. Case $F^{Adaptive}$: predict the class y'_k using F with $\text{Error}_{\min}$
7. IF y'_k equals to the actual class label in S, THEN C_T ++
8. ELSE C_F ++
9. $\Delta C = C_T - C_F$
10. Return ΔC

Fig. 6.4 Pseudo code of testing approach

root to a predicted leaf $\hat{y}_k$. Comparing the predicted class $\hat{y}_k$ to the actual class y_k, the sequential-error statistics of C_T and C_F prediction are updated immediately.

To store OCD for OVFDT, F^{MC}, F^{NB}, and F^{WNB} require memory proportional to $O(N \cdot I \cdot J \cdot K)$, where N is the number of nodes in tree model; I is the number of attributes; J is the maximum number of values per attribute; K is the number of classes. OCD of F^{NB} and F^{WNB} are converted from that of F^{MC}. In other words, we don't require extra memory to store three different OCD for F^{Adaptive} respectively. When required, it can be converted from F^{MC}.

OVFDT Training Approach

Right after the testing approach, the training follows. Node-splitting estimation is used to initially decide if HT should be updated or not; that depends on the amount of samples received so far that can potentially be represented by additional underlying rules in the decision tree. In principle, the optimized node-splitting estimation should apply on every single new sample that arrives. Of course this will be too exhaustive, and it will slow down the tree-building process. Instead, a parameter $n_{\min}$ is proposed in VFDT that only do the node-splitting estimation when $n_{\min}$ examples have been observed on a leaf. In the node-splitting estimation, the tree model should be updated when a heuristic function $H(\cdot)$ chooses

the most appropriate attribute with highest heuristic function value $H(x_a)$ as a node-splitting according to HB and tie-breaking threshold. The heuristic function is implemented as an information gain here. This situ of node-splitting estimation constitutes to the so-called training phase.

The node-splitting test is modified to use a dynamic tie-breaking threshold τ, which restricts the attribute splitting as a decision node. The τ parameter traditionally is pre-configured with a default value defined by the user. The optimal value is usually not known until all of the possibilities in an experiment have been tried. An example has been presented in section "Relationship Among Accuracy, Tree Size, and Time". Longitudinal testing of different values in advance is certainly not favorable in real-time applications. Instead, we assign a dynamic tie threshold, equal to the dynamic mean of *HB* at each pass of stream data, as the splitting threshold, which controls the node-splitting during the tree-building process. Tie-breaking that occurs close to the *HB* mean can effectively narrow the variance distribution. *HB* mean is calculated dynamically whenever new data arrives.

The estimation of splits and ties is only executed once for every $n_{\min}$ (a user-supplied value) samples that arrive at a leaf. Instead of a pre-configured tie, OVFDT uses an adaptive tie that is calculated by incremental computing. At the i_{th} node-splitting estimation, the HB estimates the sufficient statistics for a large enough sample size to split a new node, which corresponds to the leaf l. Let T_l be an adaptive tie corresponding to leaf l, within k estimations seen so far. Suppose μ_l is a binary variable that takes the value of 1 if *HB* relates to leaf l, and 0 otherwise. T_l is computed by:

$$T_l = \frac{1}{k}\sum_{i=1}^{k} \mu_l \times HB_i$$

To constrain HB fluctuation, an upper bound TlUPPER and a lower bound TlLOWER are proposed in the adaptive tie mechanism. The formulas are:

$$T_l^{\text{UPPER}} = \arg\max T_l$$

$$T_l^{\text{LOWER}} = \arg\min T_l$$

For lightweight operations, we propose an error-based pre-pruning mechanism for OVFDT, which stops non-informative split node before it splits into a new node. The pre-pruning takes into account the node-splitting error both globally and locally.

According to the optimization goal mentioned in section "Motivation and Overview", besides the HB, we also consider the global and local accuracy in terms of the sequential-error statistics of C_T and C_F prediction computed by functional tree leaf. Let ΔC_m be the difference between C_T and C_F, and m is the index of testing approach. Then ΔC_m is computed by (6.4), which reflects the global accuracy of the current HT prediction on the newly arrived data streams. If $\Delta C_m \geq 0$, the number of correct predictions is no less than the number of incorrect predictions in the current tree structure; otherwise, the current tree graph needs to be updated by node-splitting. In this approach, the statistics of correctly C_T and incorrectly C_F prediction are updated. Suppose $\Delta C_m = C_T - C_F$, which reflects the accuracy of HT. If ΔC declines, it means the global accuracy of current HT model worsens. Likewise, compare ΔC_m and ΔC_{m+1}, the local accuracy is monitored during the node-splitting. If ΔC_m is greater than ΔC_{m+1}, it means the current accuracy is declining locally. In this case, the HT should be updated to suit the newly arrival data streams.

Lemma 2 Monitor Global Accuracy The model's accuracy varies whenever a node splits and the tree structure is updated. Overall accuracy of current tree model is monitored during node-splitting by comparing the number of correctly and incorrectly predicted samples. The number of correctly predicted instances and otherwise is recorded as global performance indicators so far. This monitoring allows the global accuracy to be determined.

Lemma 3 Monitor Local Accuracy The global accuracy can be tracked by comparing the number of correctly predicted samples with the number of wrongly predicted ones. Likewise, comparing the global accuracy measured at the current node-splitting estimation with the previous splitting, the increment

in accuracy is being tracked dynamically. This monitoring allows us to check whether the current node-splitting is advantageous at each step by comparing with the previous step.

Figure 6.5 gives an example why our proposed pre-pruning takes into account both the local and the global accuracy in the incremental pruning. At the i_{th} node-splitting estimation, the difference between correctly and incorrectly predicted classes was ΔC_i, and ΔC_{i+1} was at $i{+}1_{\text{th}}$ estimation. $(\Delta C_i - \Delta C_{i+1})$ was negative that the local accuracy of $i{+}1_{\text{th}}$ estimation was worse than its previous one, while both were on a global increasing trend. Hence, if accuracy is getting worse, it is necessary to update the HT structure.

Combining the prediction statistics gathered in the testing phase, Fig. 6.6 presents the pseudo code of the training phase in OVFDT in building an upright tree. The optimized node-splitting control is presented in Fig. 6.6 Line 7. In each node-splitting estimation process, *HB* value that relates to a leaf *l* is recorded. The recorded *HB* values are used

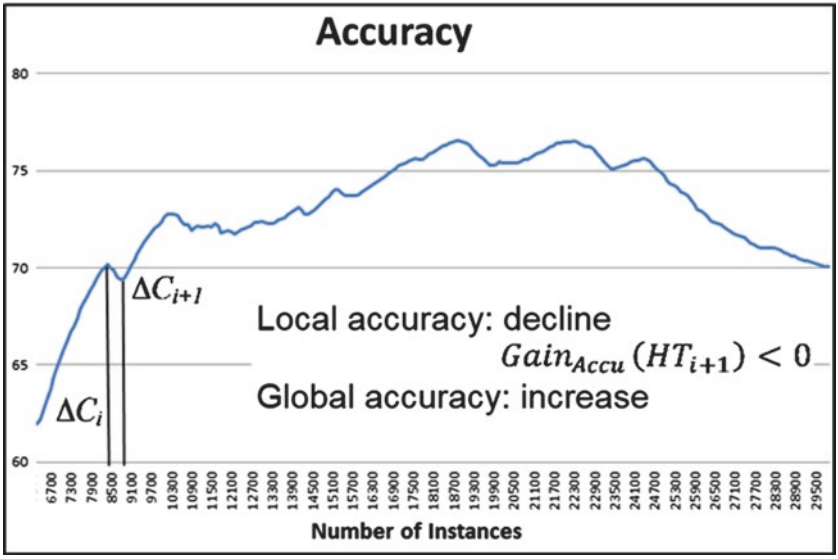

Fig. 6.5 Example of incremental pruning

PROCEDURE ***doNodeSplittingEstimation***$(\Delta C, S, X, G(.), \delta)$

1. FOR each attribute $X_i \in X_l - \{X_\emptyset\}$ at the leaf l
2. Compute $G_l(X_i)$
3. Let X_a be the attribute with highest $G_l(.)$ and X_b with the 2nd highest $G_l(.)$
4. Compute HB with δ
5. Let $\Delta G_l = G_l(X_a) - G_l(X_b)$
6. END-FOR
7. IF ($\Delta G_l > HB$) or ($\Delta G \leq T_l^{LOWER}$ and $\Delta C_i < \Delta C_{i-1}$) *or* ($\Delta G \leq T_l^{LOWER}$ and $\Delta C_i < 0$) or ($T_l^{LOWER} < \Delta G \leq T_l^{UPPER}$ and $\Delta C_i < \Delta C_{i-1}$)
8. Replace l by an internal node splits on X_a
9. Update Adaptive tie T_l^{LOWER} and T_l^{UPPER}
10. FOR each branch of splitting
11. Add a new leaf l_m and let $X_m = X - \{X_a\}$
12. Let $G(X_\emptyset)$ be $G(.)$ obtained by predicting the class in S, according to F at l_m
13. FOR each class y_k and each value x_{ij} of each attribute
14. $X_i \in X_m - \{X_\emptyset\}$ and reset OCD: $n_{ijk}(l) = 0$
15. END-FOR
16. END-FOR
17. END-IF
18. Return updated HT

Fig. 6.6 Pseudo code of training approach

to compute the adaptive tie, which uses the mean of *HB* to each leaf *l*, instead of a fixed user-defined value in VFDT.

Evaluation

Evaluation Platform and Datasets

A Java package with OVFDT and an embedded MOA toolkit was constructed as a simulation platform for experiments. The running environment was a Windows 7 PC with Intel Quad 2.8GHz CPU and 8G RAM. In all of the experiments, the parameters of the algorithms were $\delta = 10^{-6}$ and $n_{\min} = 200$, which are default values suggested by MOA. *δ is the allowable error in split decision and values closer to zero will take longer to decide; $n_{\min}$ is the number of instances a leaf should observe between split attempts*. The main goal of this section is to provide evidence of the improvement of OVFDT compared to the original VFDT.

The experimental datasets, including pure nominal datasets, pure numeric datasets, and mixed datasets, were either synthetics generated by the MOA Generator or extracted from real-world applications that are publicly available for download from the UCI repository. The descriptions of each experimental dataset are listed in Table 6.3. The generated datasets were also used in previous VFDT-related studies.

The testing ran using a test-then-train approach that is common in stream mining. When a new instance arrived that represented a segment of the incoming data stream, it was sorted by the tree model into a predicted class. This was the testing approach for deriving the predicted class via the latest form of the decision tree. Compared to the actual class label it belonged to, the tree model was updated in the training approach because the prediction accuracy was known. The decision tree can typically take either form of incoming data instances; if the instances are labeled they will be used for training or learning for the decision tree to update itself. If unlabeled, the instances are taken as unseen samples and a prediction is made in the testing phase. In our experiments, all instances were labeled because our objective was to measure the performance of model learning and prediction accuracy.

Synthetic Data *LED24* was generated by MOA. In the experiment, we added 10% noisy data to simulate imperfect data streams. The LED24 problem used 24 binary attributes to classify 10 different classes. *Waveform* was generated by the MOA Generator. The dataset was donated by David

Table 6.3 Description of experimental datasets

Name	*Nom#*	*Num#*	*Cls#*	*Type*	*Instance#*
LED24 10% Noise	24	0	10	Synthetic	10^6
Waveform 21	0	21	3	Synthetic	10^6
Waveform 40	0	40	3	Synthetic	10^6
Random Tree Simple (RTS)	10	10	2	Synthetic	10^6
Random Tree Complex (RTC)	50	50	2	Synthetic	10^6
RBF Simple (RBFS)	0	10	2	Synthetic	10^6
RBF Complex (RBFC)	0	50	2	Synthetic	10^6
Connect-4	42	0	7	UCI	67,557
Person Activity Data (PAD)	2	3	11	UCI	164,860
Cover Type (COVTYPE	42	12	7	UCI	581,012
Nursery	8	0	5	UCI	12,960

Aha to the UCI repository. The goal of the task was to differentiate between three different classes of Waveform. There were two types of waveform: Wave21 had 21 numeric attributes and Wave40 had 40 numeric attributes, all of which contained noise. *Random Tree (RTS and RTC)* was also generated by the MOA Generator. It built a decision tree by choosing attributes to split randomly and assigning a random class label to each leaf. As long as a tree was constructed, new samples were generated by assigning uniformly distributed random values to attributes. Those attributes determined the class label through the tree. *Radial basis Function (RBFS and RBFC)* is a fixed number of random centroids generator. A random position, a single standard deviation, a class label and weight are generated by a centroid.

UCI Data *Connect-4* contained all of the legal 8-ply positions in a two-player game of Connect-4. In the game, the player's next move was not forced and the game was won once four chessmen were connected. *Personal activity data (PAD)* recorded the data streams collected from 4 sensors on the players' bodies. Each sensor collected 3 numeric data. *Cover Type* was used to predict forest cover type from cartographic variables. *Nursery* was from a hierarchical decision model originally developed to rank applications for nursery schools. Because of its known underlying concept structure, this dataset can be useful for testing constructive learning induction and structure discovery algorithms.

Accuracy Comparison

Table 6.4 shows the comparison results of the accuracy tests. On average, OVFDT obtained a higher accuracy for the pure nominal datasets than the mixed datasets. For the numeric Waveform datasets, OVFDT also displayed better accuracy than the other VFDTs. This phenomenon was particularly obvious in OVFDT with the adaptive Functional Tree Leaf. For each dataset, a detailed comparison of its accuracy with the new arrival data streams is illustrated in the Appendix.

Table 6.4 Accuracy (%) comparison

Methods	VFDT tie 0.05				VFDT tie 0.5				OVFDT			
	MC	NB	WNB	ADP	MC	NB	WNB	ADP	MC	NB	WNB	ADP
Datasets												
LED_NP10	63.781	73.73	73.55	73.88	73.68	72.53	73.71	73.26	73.636	73.44	73.82	**73.89**
CONNECT-4	67.3	72.28	73.48	73.48	69.01	72.87	74.26	74.09	69.01	73.55	74.58	**74.98**
NURSERY	83.78	88.5	88.21	89.07	82.09	89.16	86.81	89.82	82.75	89.24	87.56	**90.29**
Nominal AVG	71.62	78.17	78.41	78.81	74.93	78.19	78.26	79.06	75.13	78.74	78.65	**79.72**
WAVE21	76.36	83.15	84.55	83.91	80.9	82.02	82.59	82.47	78.6	83.24	84.47	**84.57**
WAVE40	76.42	83.14	84.37	83.75	80.89	81.31	81.79	81.90	79.1	83.16	84.31	**84.77**
RBFS	82.62	85.09	85.58	86.47	87.71	89.18	89.43	**89.47**	84.87	85.63	86.68	87.73
RBFC	80.75	90.01	90.71	90.74	88.38	92.72	**92.88**	92.82	78.67	89.09	89.45	89.32
Numeric AVG	79.04	85.35	86.30	86.22	84.47	86.31	**86.67**	**86.67**	80.31	85.28	86.23	86.60
RTS	91.78	94.86	94.24	94.77	93.16	95.8	95.52	95.72	92.41	95.56	95.1	**95.84**
RTC	95.14	95.62	95.59	95.63	95.55	95.72	95.71	95.74	95.4	95.67	95.63	**95.76**
COVTYPE	67.45	77.16	78.6	77.77	74.19	90.76	95.52	95.71	92.41	95.55	95.1	**95.84**
PAD	43.75	61.04	59.69	61.01	55.9	72.65	71.56	71.29	51.22	71.07	70.05	**72.7**
Mixed AVG	74.53	82.17	82.03	82.30	79.70	88.73	89.58	89.62	82.86	89.46	88.97	**90.04**
AVG	75.06	81.90	82.25	82.44	79.70	84.41	84.84	85.11	79.43	84.50	84.62	**85.45**

Note: The figures in bold are the greatest numbers obtained per row of algorithms experimented with a specific dataset

Our comparison of the four functional tree leaf strategies revealed that $OVFDT_{ADP}$ generally had the highest accuracy in the most experimental datasets. An improvement comparison of functional tree leaf strategies is given in Table 6.5. The majority class functional tree leaf strategy was chosen as a benchmark. As a result, the adaptive functional tree leaf strategy obtained the best accuracy, with $F^{Adaptive} \succ F^{WNB} \succ F^{NB} \succ F^{MC}$. This result appeared in both VFDT and OVFDT methods.

Tree Size Comparison

For all of the datasets, a comparison of tree size is shown in Table 6.6. For the pure nominal and mixed datasets, VFDT with a smaller τ generally had smaller tree sizes, but OVFDT obtained the smallest tree size with the pure numeric datasets. For each dataset, a detailed comparison of accuracy with the new arrival data streams is illustrated in the Appendix. The charts in the Appendix essentially show the performance on the *y*-axis and the dataset samples on the *x*-axis.

Tree Learning Time Comparison

A comparison of tree learning time is shown in Table 6.7. For all of the datasets, the majority class functional tree leaf consumed the least time in this experiment due to its simplicity. The computation times of the other three Functional Tree Leaves, using the *Naïve Bayes* classifier, were close.

Stability of Functional Tree Leaf in OVFDT

Stability is related to the degree of variance in the prediction results. A stable model is translated into a useful model and its prediction accuracy over the same datasets does not vary significantly, regardless of how many times it is tested. To show the stability of different functional tree leaf mechanisms in OVFDT, we ran the evaluation based on those synthetic datasets ten times. In this experiment, the synthetic datasets were generated using different random seeds. Hence, the generated data streams had

Table 6.5 Accuracy improvement by Functional Tree Leaf

		VFDT tie 0.05				VFDT tie 0.5				OVFDT			
	Methods	MC	NB	WNB	ADP	MC	NB	WNB	ADP	MC	NB	WNB	ADP
Datasets													
Nominal	Accuracy %	71.62	78.17	78.41	78.81	74.93	78.19	78.26	79.06	75.13	78.74	78.65	79.72
	FL Imp. %	0.00	9.15	9.49	10.04	0.00	4.35	4.44	5.51	0.00	9.95	9.82	11.31
Numeric	Accuracy %	79.04	85.35	86.30	86.22	84.47	86.31	86.67	86.67	80.31	85.28	86.23	86.60
	FL Imp. %	0.00	7.98	9.19	9.08	0.00	2.18%	2.61%	2.60	0.00	6.19	7.37	7.83
Mixed	Accuracy %	74.53	82.17	82.03	82.30	79.70	88.73	89.58	89.62	82.86	89.46	88.97	90.04
	FL Imp. %	0.00	10.25	10.06	10.42	0.00	11.33	12.39	12.44	0.00	7.97	7.37	8.66

Table 6.6 Tree size comparison

	VFDT tie 0.05				VFDT tie 0.5				OVFDT			
Methods	MC	NB	WNB	ADP	MC	NB	WNB	ADP	MC	NB	WNB	ADP
Datasets												
LED_NP10	46	46	46	46	2440	2440	2440	2440	370	227	219	172
CONNECT-4	23	23	23	23	437	437	437	437	141	96	92	94
NURSERY	72	72	72	72	13	13	13	13	21	13	13	13
Nominal AVG	47	47	47	47	963	963	963	963	177	112	108	93
WAVE21	160	160	160	160	3557	3557	3557	3557	263	197	181	178
WAVE40	150	150	150	150	3607	2340	3607	3607	413	194	154	152
RBFS	420	420	420	420	2723	2723	2723	2723	398	372	389	391
RBFC	443	443	443	443	2052	3145	3145	3145	604	381	372	396
Numeric AVG	293	293	293	293	2985	2941	3258	3258	420	286	274	279
RTS	1680	1680	1680	1680	2683	2683	2683	2683	1940	1691	1812	1739
RTC	620	620	620	620	1492	1492	1492	1492	670	682	670	678
COVTYPE	127	127	127	127	1882	1882	2683	2683	1940	1691	1812	1739
PAD	167	167	167	167	1829	1614	1614	1614	905	950	855	953
Mixed AVG	649	649	649	649	1972	1918	2118	2118	1364	1254	1287	1277
AVG	330	330	330	330	1973	1941	2113	2113	654	551	556	550

Table 6.7 Tree learning time comparison

	VFDT tie 0.05				VFDT tie 0.5				OVFDT			
Methods	MC	NB	WNB	ADP	MC	NB	WNB	ADP	MC	NB	WNB	ADP
Datasets												
LED_NP10	8.25	13.15	13.23	14.76	14.13	18.21	18.08	19.11	10.58	23.15	22.82	20.63
CONNECT-4	1.28	1.53	1.48	1.53	1.67	1.86	1.83	1.93	1.50	1.95	1.97	2.17
NURSERY	0.34	0.36	0.36	0.37	0.33	0.34	0.34	0.37	0.37	0.39	0.39	0.39
Nominal AVG	3.29	5.01	5.02	5.55	5.38	6.80	6.75	7.14	4.15	8.50	8.39	7.73
WAVE21	17.13	23.18	23.06	26.89	21.54	24.84	24.96	26.94	18.08	34.27	34.38	42.15
WAVE40	30.87	41.90	42.07	49.34	38.19	45.05	45.57	48.61	33.06	64.23	63.37	76.70
RBFS	9.39	11.29	11.37	12.32	11.01	12.43	12.45	13.74	10.06	15.79	15.80	17.34
RBFC	38.20	47.94	47.99	55.43	44.40	52.20	51.71	57.28	42.51	69.40	69.64	86.32
Numeric AVG	23.90	31.08	31.12	36.00	28.79	33.63	33.67	36.64	25.93	45.92	45.80	55.63
RTS	15.43	17.58	17.83	18.78	16.80	18.39	18.30	19.39	16.19	21.93	21.84	22.11
RTC	64.66	74.60	75.19	78.74	67.60	76.47	77.06	80.61	67.03	96.58	96.21	92.84
COVTYPE	11.04	15.24	15.05	18.36	12.68	15.23	18.30	19.39	16.19	21.93	21.84	22.11
PAD	1.31	2.04	1.97	2.48	1.53	1.92	1.90	2.17	1.56	2.68	2.78	3.49
Mixed AVG	23.11	27.37	27.51	29.59	24.65	28.00	28.89	30.39	25.24	35.78	35.67	35.14
AVG	16.77	21.15	21.22	23.71	19.60	22.81	23.10	24.72	18.44	30.07	29.95	32.83

exactly the same data formats, but different random values. The average and its variance of accuracy in the testing are shown in Table 6.8. Generally, datasets that only contained the homogenous attribute types (numeric attributes only or nominal attributes only) had smaller variances. The proposed adaptive functional tree leaf obtained the least variances because it had more stable and comparable accuracy than the other functional tree leaves.

Optimal Tree Model

Figure 6.7 presents a comparison of the optima ratio, which was calculated by optimization function accuracy/tree size in (6.5). The higher this ratio is, the better the optimal result. Comparing VFDT to OVFDT, the ratios of OVFDT were clearly higher than those of VFDT. In other words, the optimal tree structures were achieved in the pure numeric and mixed datasets.

Conclusion

Imperfect data stream leads to tree size explosion and detrimental accuracy problems. In original VFDT, a tie-breaking threshold that takes a user-defined value is proposed to alleviate this problem by controlling the node-splitting process that is a way of tree growth. But there is no single default value that always works well and that user-defined value is static throughout the stream mining operation. In this chapter, we propose an extended version of VFDT which we called it Optimized-VFDT (OVFDT) algorithm that uses an adaptive tie mechanism to automatically search for an optimized amount of tree node-splitting, balancing the accuracy, the tree size and the time, during the tree-building process. The optimized node-splitting mechanism controls the attribute-splitting estimation incrementally. Balancing between the accuracy and tree size is important, as stream mining is supposed to operate in limited memory computing environment and a reasonable accuracy is needed. It is a known contradiction that high accuracy requires a large tree with many

Table 6.8 The average and variance of accuracy in four types of Functional Tree Leaves

	Average				Variance			
Method	MC	NB	WNB	ADP	MC	NB	WNB	ADP
Data								
Led_np10	73.6454	73.4987	73.8416	73.9080	0.0106	0.0067	0.0019	0.0027
Led_np15	61.9010	60.7587	61.8477	61.9785	0.0025	0.0255	0.0013	0.0014
Led_np20	51.1407	47.5278	50.7018	51.0752	0.0026	0.1113	0.0029	0.0011
Led_np25	41.4070	35.7713	39.9215	41.1294	0.0027	0.1118	0.0255	0.0029
AVG.Nom.	57.0235	54.3891	56.5782	57.0228	0.0046	0.0638	0.0079	0.0020
Waveform21	79.0604	83.2396	84.5739	84.5634	0.0785	0.0151	0.0025	0.0025
Waveform40	79.0791	83.1607	84.3698	84.3527	0.0697	0.0067	0.0051	0.0058
RBFS	89.8142	90.8911	91.6129	92.4029	2.3677	2.0308	2.3580	1.7588
RBFC	98.1270	98.0033	98.3046	98.9693	0.0710	0.1518	0.1020	0.0329
AVG.Num.	86.5202	88.8237	89.7153	90.0721	0.6467	0.5511	0.6169	0.4500
RTS	89.9290	93.0060	92.4382	93.0394	10.0790	9.6065	9.7757	9.3052
RTC	89.5048	84.9352	87.1039	87.4002	44.1081	84.7361	80.1807	80.7949
AVG.Mix.	89.7169	88.9706	89.7711	90.2198	27.0936	47.1713	44.9782	45.0501

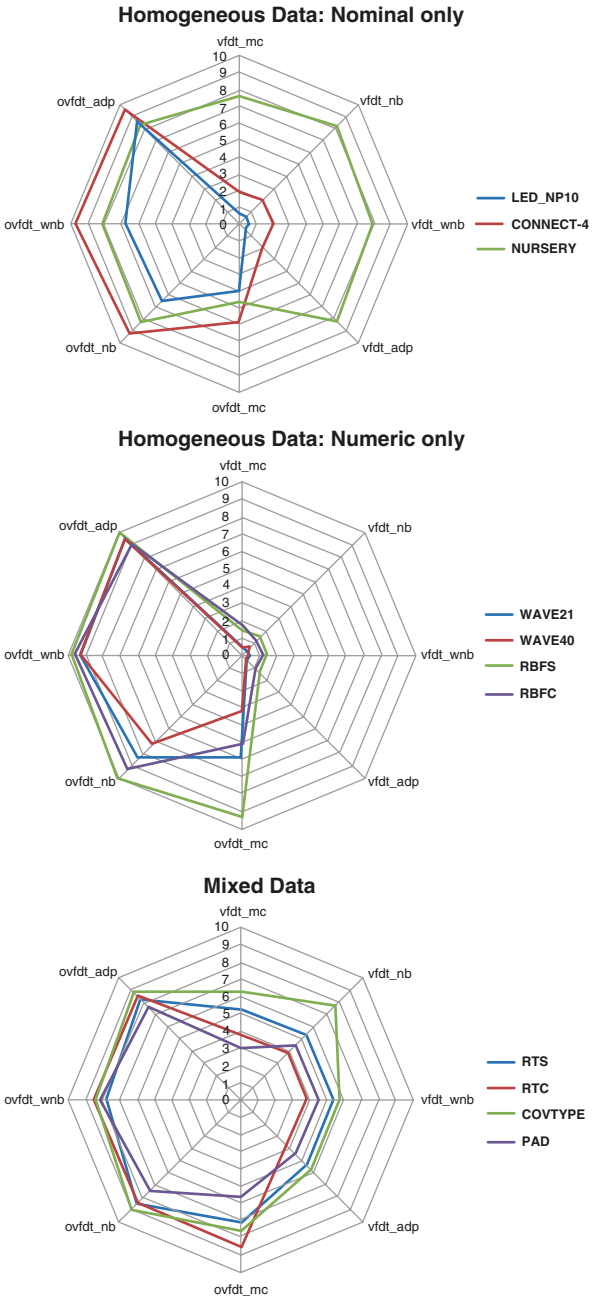

Fig. 6.7 Comparison of optimal tree structures between VFDT and OVFDT

decision paths, and too sparse the decision tree results in poor accuracy. The experiment results show that OVFDT meet the optimization goal and achieve a better performance gain ratio in terms of high prediction accuracy and compact tree size than the other VFDTs. That is, with the minimum tree size, OVFDT can achieve the highest possible accuracy. This advantage can be technically accomplished by means of simple incremental optimization mechanisms as described in this chapter. They are light-weighted and suitable for incremental learning. The contribution is significant because OVFDT can potentially be further modified into other variants of VFDT models in various applications, while the best possible (optimal) accuracy and minimum tree size can always be guaranteed.

Acknowledgment The authors are thankful for the financial support from the research grants "Temporal Data Stream Mining by Using Incrementally Optimized Very Fast Decision Forest (iOVFDF)", Grant no. MYRG2015-00128-FST offered by the University of Macau, FST, and RDAO, and "A scalable data stream mining methodology: stream-based holistic analytics and reasoning in parallel", Grant no. FDCT-126/2014/A3, offered by FDCT Macau.

Reference

Bifet, A., & Gavalda, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of SIAM International Conference on Data Mining* (pp. 443–448).

Bifet A., Geoff, H., Bernhard, P., Jesse, R., Philipp, K., Hardy, K., Timm, J., & Thomas, S. (2001). MOA: A Real-Time Analytics Open Source Framework. In *Machine Learning and Knowledge Discovery in Databases* (pp. 617–620). Lecture Notes in Computer Science, Volume 6913/2011.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., & Gavalda, R. (2009). New Ensemble Methods for Evolving Data Streams. In *Proceedings 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 139–147). New York: ACM.

Elomaa, T. (1999). *The Biases of Decision Tree Pruning Strategies, Advances in Intelligent Data Analysis* (pp. 63–74). Lecture Notes in Computer Science, Volume 1642/1999. Berlin/Heidelberg: Springer.

Gama, J., & Kosina, P. (2011). Learning Decision Rules from Data Streams. In T. Walsh (Ed.), *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence – Volume Two* (Vol. 2, pp. 1255–1260). Menlo Park: AAAI Press.

Gama J, Rocha R., & Medas P. (2003). Accurate Decision Trees for Mining High-Speed Data Streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 523–528). ACM, New York.

Geoffrey H., Richard K., & Bernhard P. (2005). Tie Breaking in Hoeffding Trees. In *Proceedings Workshop W6: Second International Workshop on Knowledge Discovery in Data Streams* (pp. 107–116).

Hartline J. R. K. (2008). *Incremental Optimization* (PhD Thesis). Faculty of the Graduate School, Cornell University.

Hashemi, S., & Yang, Y. (2009). Flexible Decision Tree for Data Stream Classification in the Presence of Concept Change, Noise and Missing Values. *Data Mining and Knowledge Discovery, 19*(1), 95–131.

Hulten G., & Domingos P. (2003). *VFML – A Toolkit for Mining High-Speed Time-Changing Data Streams*. http://www.cs.washington.edu/dm/vfml/

Hulten G., Spencer L., & Domingos P. (2001). Mining Time-Changing Data Streams. In *Proceedings* of the *7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 97–106).

Mladenic D., & Grobelnik M. (1999). Feature Selection for Unbalanced Class Distribution and Naive Bayes, In *Proceeding ICML '99 Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 258–267). ISBN 1-55860-612-2, Morgan Kaufmann.

Nitesh, C., Nathalie, J., & Alek, K. (2004). Special Issue on Learning from Imbalanced Data Sets. *ACM SIGKDD Explorations, 6*(1), 1–6.

Oza N., & Russell S. (2001). Online Bagging and Boosting. In *Artificial Intelligence and Statistics* (pp. 105–112). San Mateo: Morgan Kaufmann.

Pedro D., & Geoff H. (2000). Mining High-Speed Data Streams. In *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 71–80).

Pfahringer B., Holmes G., & Kirkby R. (2007). New Options for Hoeffding Trees. In *Proceedings in Australian Conference on Artificial Intelligence* (pp. 90–99).

Stefan H., Russel P., & Yun S. K. (2009). *CBDT: A Concept Based Approach to Data Stream Mining* (pp. 1006–1012). Lecture Notes in Computer Science, Volume 5476/2009.

Yang H., & Fong S. (2011). Moderated VFDT in Stream Mining Using Adaptive Tie Threshold and Incremental Pruning. In *Proceedings of the 13th International Conference on Data Warehousing And Knowledge Discovery* (pp. 471–483). Berlin/Heidelberg: Springer-Verlag.