Data Min Knowl Disc (2011) 23:128–168
DOI 10.1007/s10618-010-0201-y

Learning model trees from evolving data streams

Elena Ikonomovska · João Gama · Sašo Džeroski

Received: 27 January 2010 / Accepted: 9 September 2010 / Published online: 15 October 2010

Abstract The problem of real-time extraction of meaningful patterns from time-changing data streams is of increasing importance for the machine learning and data mining communities. Regression in time-changing data streams is a relatively unexplored topic, despite the apparent applications. This paper proposes an efficient and incremental stream mining algorithm which is able to learn regression and model trees

This paper has its origins in two conference papers that propose and partly evaluate a new algorithm for learning model trees from stationary data streams (Ikonomovska and Gama 2008) and an improvement of this algorithm for learning from non-stationary data streams (Ikonomovska et al. 2009). However, this paper significantly extends and upgrades the work presented there, both on the algorithmic design and experimental evaluation fronts. Concerning the algorithm: We consider a new approach for the split selection criteria; We include a memory saving method for disabling bad split points; We improve the adaptation mechanism by using the Q statistic with a fading factor (Gama et al. 2009); We discuss and propose several memory management methods that enable most effective learning with constrained resources (deactivating and reactivating non-problematic leaves, removing non-promising alternate trees). We also perform a much more extensive and in-depth experimental evaluation: We consider a larger collection of real-world datasets; we use a more carefully designed experimental methodology (sliding window prequential and holdout evaluation among others); we provide a much more comprehensive discussion of the experimental results.

Responsible editor: Eamonn Keogh.

E. Ikonomovska (✉) · S. Džeroski
Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
e-mail: elena.ikonomovska@ijs.si

J. Gama
LIAAD/INESC, University of Porto, Rua de Ceuta, 118-6, 4050-190 Porto, Portugal

J. Gama
Faculty of Economics, University of Porto, Rua Roberto Frias, 4200 Porto, Portugal

E. Ikonomovska
Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University, Karpos II bb, 1000 Skopje, Macedonia

from possibly unbounded, high-speed and time-changing data streams. The algorithm is evaluated extensively in a variety of settings involving artificial and real data. To the best of our knowledge there is no other general purpose algorithm for incremental learning regression/model trees able to perform explicit change detection and informed adaptation. The algorithm performs online and in real-time, observes each example only once at the speed of arrival, and maintains at any-time a ready-to-use model tree. The tree leaves contain linear models induced online from the examples assigned to them, a process with low complexity. The algorithm has mechanisms for drift detection and model adaptation, which enable it to maintain accurate and updated regression models at any time. The drift detection mechanism exploits the structure of the tree in the process of local change detection. As a response to local drift, the algorithm is able to update the tree structure only locally. This approach improves the any-time performance and greatly reduces the costs of adaptation.

Keywords Non-stationary data streams · Stream data mining · Regression trees · Model trees · Incremental algorithms · On-line learning · Concept drift · On-line change detection

1 Introduction

In the last decade, data streams (Aggarwal 2006) have been receiving growing attention in many research communities, due to the wide recognition of applications emerging in many areas of everyday activities. Examples include financial applications (stock exchange transactions), telecommunication data management (call records), web applications (customers click stream data), surveillance (audio/video data), bank-system management (credit card/ATM transactions, etc.), monitoring patient health, and many others. Such data are typically represented as evolving time series of data items, arriving continuously at high speeds, and having dynamically changing distributions. Modeling and predicting the temporal behavior of streaming data can provide valuable information towards the success of a time-critical operation.

The task of regression analysis is one of the most commonly addressed topics in the areas of machine learning and statistics. Regression and model trees are often used for this task due to their interpretability and good predictive performance. However, regression on time-changing data streams is a relatively unexplored and typically non-trivial problem. Fast and continuous data feeds as well as the time-changing distributions make traditional regression tree learning algorithms unsuitable for data streams. This paper proposes an efficient and incremental algorithm for learning regression and model trees from possibly unbounded, high-speed, and time-changing data streams. Our contributions are fourfold: an efficient splitting-attribute selection in the incremental growing process; an effective approach for computing the linear models in the leaves; an efficient method for handling numerical attributes; and change detection and adaptation mechanisms embedded in the learning algorithm.

The remainder of this paper is organized as follows. Section 2 reviews related work, starting with batch and incremental algorithms for inducing model trees, continuing with algorithms for learning classification trees from data streams and concluding with

techniques for change detection. In Sect. 3, we present our algorithm for incremental learning of regression and model trees from time-changing data streams and discuss the design choices. The evaluation methodology is described in Sect. 4, while the results of the evaluation are presented in Sect. 5. The evaluation includes experiments on both artificial and real-world datasets. Conclusions are given in Sect. 6, where we also discuss possible directions for immediate further work.

2 Related work

As related work, we first consider batch and incremental learning of model trees. We then turn our attention to related methods for learning from stationary data streams: learning decision/classification trees and linear regression/neural networks. Finally, we take a look at methods for on-line change detection and management of concept drift.

2.1 Batch learning of model trees

Regression and model trees are known to provide efficient solutions to complex non-linear regression problems due to the divide-and-conquer approach applied on the instance-space. Their main strength in solving a complex problem is by recursively fitting different models in each subspace of the instance-space. Regression trees use the mean for the target variable as the prediction for each sub-space. Model trees improve upon the accuracy of regression trees by using more complex models in the leaf nodes.

The splitting of the instance-space is performed recursively by choosing a split that maximizes some error reduction measure, with respect to the examples that are assigned to the current region (node). In one category are falling algorithms like M5 (Quinlan 1992), CART (Breiman et al. 1998), HTL (Torgo 1997). These use variants of the variance reduction measure used in CART, like standard deviation reduction or the fifth root of the variance in M5′ the WEKA (WEKA 3 2005) implementation of the algorithm M5. Another category of algorithms are those aiming to find better globally optimal partitions by using more complex error reduction methods, on the cost of increased computational complexity. Here fall RETIS (Karalic 1992), SUPPORT (Chaudhuri et al. 1994), SECRET (Dobra and Gherke 2002), GUIDE (Loh 2002), SMOTI (Malerba et al. 2002) and LLRT (Vogel et al. 2007).

All existing batch approaches to building regression/model trees assume that the training set is finite and stationary. For this reason, they require all the data for training to be available on the disk or in main memory before the learning process begin. When given very large training sets, batch algorithms have shown to be prohibitively expensive both in memory and time. In this spirit, several efforts have been made in speeding up learning on large datasets. Notable examples are SECRET (Dobra and Gherke 2002) and the LLRT algorithm (Vogel et al. 2007). Their main weakness is that as rest of the batch algorithms, they require storing of all the examples in main memory. This becomes a major problem when our datasets are larger than the main memory of the system. In such situations, users are forced to do sub-sampling or other data reduction methods. This is a nontrivial task, because of the danger of underfitting. Another characteristic of large data is that, it is typically collected over a long time

period or generated rapidly by a continuous, possibly distributed sources of data. In both of these scenarios, there is a high probability of non-stationary relations, which in learning problems takes the form of concept drift. We have noted that none of the existing batch algorithms for learning regression trees is able to deal with concept drift.

As a conclusion, traditional batch methods are not suitable for dealing with continuous feeds of data, because the required memory and processing power might easily outgrow the available system resources and the given response period for which the answers are still relevant. This motivates the need of incremental algorithms with fast execution and response time that need a sub-linear amount of memory for the learning task, and are able to detect changes in the concept and adapt their models correspondingly.

2.2 Incremental learning of model trees

Mining data streams raises many new problems previously not encountered in data mining. One crucial issue is the real-time response requirement, which severely constrains the use of complex data mining algorithms that perform multiple passes over the data. Although regression and model trees are an interesting and efficient class of learners, little research has been done in the area of incremental regression or model tree induction.

To the best of our knowledge, there is only one paper (Potts and Sammut 2005) addressing the problem of incremental learning of model trees. The authors follow the method proposed by Siciliano and Mola (1994), applying it in an incremental way. They have proposed two different splitting rules (RD and RA), both of which considered in a batch (BatchRD and BatchRA) and in an incremental version (OnlineRD and OnlineRA).

The splitting decision is formulated as hypothesis testing. For the RD splitting rule, the null hypothesis is that the examples associated with a node are generated by a single linear model. If the null hypothesis can be rejected with the desired degree of confidence, a split should be performed. The statistical test computes the F statistic which is distributed according to Fisher's distribution. The split least likely to occur under the null hypothesis is the best one.

The F statistic is based on the residual sum of squares (RSS), which can be computed incrementally by using the recursive least squares algorithm. However, the RSS values must be computed from the linear models for every possible splitting point and for all the attributes. Each iteration of this process has a complexity of $O(d^2)$. In the case of continuous numeric attributes, the task becomes highly computationally expensive. In order to make this approach scalable, only a constant number k of candidate splits is considered for each attribute. Thus, the training phase complexity becomes $O(Nkd^3)$. This is not the most appropriate complexity for learning from fast data streams, especially when much better solutions have been proposed in the area of incremental decision tree learning (Domingos and Hulten 2000).

As an alternative, a more efficient version of the same algorithm is proposed by using the RA splitting rule. The RA splitting rule comes from SUPPORT (Chaudhuri

et al. 1994). It requires the computation of the residuals from a linear model and the distributions of the regressor values from the two sub-samples associated with the positive and negative residuals. The null hypothesis is thus the following: if the function being approximated is almost linear in the region of the node, then the positive and the negative residuals should be distributed evenly. The statistics used are differences in means and in variances, which are assumed to be distributed according to the Student's t distribution. These statistics cannot be applied to splits on categorical attributes, which are therefore ignored.

Both algorithms use incremental pruning, which can be considered as a strategy for adapting to possible concept drift. The pruning takes place if the prediction accuracy of an internal node is estimated to be not worse than its corresponding sub-tree. The accuracy is estimated using the observed error on the training set. A requirement is that a linear model has to be maintained at each internal node.

2.3 Learning decision trees from stationary data streams

The problem of incremental decision tree induction has fortunately received appropriate attention within the data mining community. There is a large literature on incremental decision tree learning, but our focus of interest is on the line of research initiated by Musick et al. (1993), which motivates sampling strategies for speeding up the learning process. They note that only a small sample from the distribution might be enough to confidently determine the best splitting attribute. Example algorithms from this line of research are the Sequential ID3 (Gratch 1996), VFDT (Domingos and Hulten 2000), UFFT (Gama et al. 2003), and the NIP-H and NIP-N algorithms (Jin and Agrawal 2003).

The Sequential ID3 algorithm is a sequential decision tree induction method, which guarantees similarity of the incrementally learned tree to the batch tree based on a statistical procedure called the sequential probability ratio test. Its guarantee however, is much looser than the one of the *Hoeffding tree* algorithm, the basis of VFDT. The VFDT system is considered as state-of-the-art and represents one of the best known algorithms for classifying streams of examples.

VFDT gives a strong guarantee that the produced tree will be asymptotically arbitrarily close to the batch tree, given that enough examples are seen (Domingos and Hulten 2000). This is achieved by using the *Hoeffding* probability bound (Hoeffding 1963), which bounds the probability of a mistake when choosing the best attribute to split. This method has proven very successful in deciding when and how (on which attribute) to expand the tree and has been followed by all the above mentioned algorithms, except the NIP-H and NIP-N algorithms (Jin and Agrawal 2003). In the NIP algorithms, the authors use the properties of the normal distribution in the estimation of the evaluation measures (entropy, Gini index) and claim that this enables them to achieve the same accuracy and guarantees using fewer examples.

The VFDT system has a memory management feature which will deactivate the least promising leaves when system RAM is nearly all consumed. Reactivation is possible when the resources are available again. Some memory can be freed also by deactivating poor attributes early in the process of split evaluation. Our algorithm was

implemented using an adapted version of the VFML library of Domingos (VFML 2003), and therefore adopts and re-implements many of the proposed memory management features of VFDT.

2.4 Other regression methods in stream mining

One of the most notable and successful examples of regression on data streams is the multi-dimensional linear regression analysis of time-series data streams (Chen et al. 2002). It is based on the OLAP technology for streaming data. This system enables an online computation of linear regression over multiple dimensions and tracking unusual changes of trends according to users' interest.

Some attempts have been also made in applying artificial neural networks over streaming data. In Rajaraman and Tan (2001), the authors address the problems of topic detection, tracking and trend analysis over streaming data. The incoming stream of documents is analyzed by using Adaptive Resonance Theory (ART) networks.

2.5 On-line change detection and management of concept drift

The nature of change is diverse. Changes may occur in the context of learning due to changes in hidden variables or changes in the intrinsic properties of the observed variables. Often these changes make the model built on old data inconsistent with new data, and regular updating of the model is necessary.

As Gao et al. (2007) have noted, the joint probability, which represents the data distribution $P(\mathrm{x}, y) = P(y|\mathrm{x}) \cdot P(\mathrm{x})$, can evolve over time in three different ways: (1) changes in $P(\mathrm{x})$ known as *virtual concept drift* (*sampling shift*); (2) changes in the conditional probability $P(y|\mathrm{x})$; and (3) changes in both $P(\mathrm{x})$ and $P(y|\mathrm{x})$. We are in particular interested in detecting changes in the conditional probability, which in the literature is usually referred to as *concept drift*. Further, a change can occur *abruptly* or *gradually*, leading to abrupt or gradual concept drift.

With respect to the region of the instance space affected by a change, concept drift can be categorized as *local* or *global*. In the case of local concept drift, the distribution changes only over a constrained region of the *instance space* (set of ranges for the measured attributes). In the case of global concept drift, the distribution changes over the whole region of the instance space, that is, for all the possible values of the target/class and the attributes.

The book of Nikiforov (Basseville and Nikiforov 1993) introduces a number of off-line and on-line methods for detection of abrupt changes. In the context of regression, change detection methods are discussed for additive changes in a linear regression model, as well as additive and non-additive changes in a linear ARMA model (Basseville and Nikiforov 1993). Relevant work in modeling time-changing continuous signals is presented by Pang and Ting (2005), who address the problem of structural break detection in forecasting non-stationary time series. However, the authors use an off-line change detection algorithm, while we are only interested in on-line change detection methods.

An interesting and relevant review of the types of concept drift and existing approaches to the problem is also given by Gama and Castillo (2004). Most existing

work falls in the area of classification. We distinguish two main approaches for managing concept drift: methods for blind adaptation, based on blind data management (instance weighting and selection) or blind model management (ensembles of models), and methods based on explicit detection of change and informed adaptation. The first category of methods is not very relevant to our work and will not be further discussed. In the second category fall methods that detect the point of change, or determine a time-window in which change has occurred. The adaptation is initiated by the event of change detection and is therefore referred to as informed. The change detection methods may follow two different approaches: monitoring the evolution of some performance indicators, or monitoring the data distribution over two different time-windows.

Examples of the first approach are the algorithms in the FLORA family (Widmer and Kubat 1996), and the work of Klinkenberg and Renz (1998). In the context of information filtering, the authors in Klinkenberg and Renz (1998) propose to monitor the values of three performance indicators: accuracy, recall and precision over time. Klinkenberg and Joachims (2000) present a theoretically well-founded method to recognize and handle concept changes using the properties of Support Vector Machines.

A representative example of the second approach is the work of Kifer et al. (2004). The idea here is to examine samples drawn from two probability distributions (from different time-windows) and decide whether these distributions are statistically different. They exploit order statistics of the data, and define generalizations of the Wilcoxon and Kolmogorov–Smirnoff test in order to define distance between two distributions. Another representative algorithm is VFDTc (Gama et al. 2003), which also performs continuous monitoring of the differences in the class-distributions of the examples from two time-windows. The change detection method in VFDTc evaluates past splitting decisions in an incrementally built classification tree and detects splits that have become invalid due to concept drift. Subramaniam et al. (2006) propose a distributed streaming algorithm for outlier and change detection that uses kernel methods in the context of sensor networks. Given a baseline data set and a set of newly observed data, Song et al. (2007) define a test statistic called the density test based on kernel estimation to decide if the observed data is sampled from the baseline distribution. A more recent method for detection of changes and correlations in data is given in Sebastiao et al. (2009): It is based on the Kullback–Leibler Divergence measure, which is a popular way to measure the distance between two probability distributions. A somewhat similar approach is used by Dasu et al. (2009) where instead of a summary of the data, they maintain a sample from the stream inside the windows used to estimate the distributions.

In this category also falls the CVFDT (Hulten et al. 2001) algorithm, which is very relevant to our work. The CVFDT algorithm performs regular periodic validation of its splitting decisions by maintaining the necessary statistics at each node over a window of examples. For each split found to be invalid, it starts growing a new decision tree rooted at the corresponding node. The new sub-trees grown in parallel are intended to replace the old ones, since they are generated using data which corresponds to the new concepts. To smooth the process of adaptation, CVFDT keeps the old subtrees rooted at the influenced node, until one of the newly grown trees becomes more accurate.

```
Input: δ user-defined probability and N_min
Output: Model tree
Begin with empty Leaf (Root)
For each example in the stream
        Read next example
        Traverse the example through the tree to a Leaf
        Update change detection tests on the path
        If (Change_is_detected)
            Adapt the model tree
        Else
            Update statistics in the Leaf
            Every N_min examples seen in the Leaf
               Find best split per attribute
               Rank attributes using the same evaluation measure
               If (Splitting_criterion_is_satisfied)
                  Make a split on the best attribute
                  Make two new branches leading to (empty) Leaves
```

Fig. 1 The incremental FIMT-DD algorithm

3 The FIMT-DD algorithm

The problem of learning model trees from data streams raises several important issues typical for the streaming scenario. First, the dataset is no longer finite and available prior to learning. As a result, it is impossible to store all data in memory and learn from them as a whole. Second, multiple sequential scans over the training data are not allowed. An algorithm must therefore collect the relevant information at the speed it arrives and incrementally decide about splitting decisions. Third, the training dataset may consist of data from several different distributions. Thus the model needs continuous monitoring and updating whenever a change is detected. We have developed an incremental algorithm for learning model trees to address these issues, named fast incremental model trees with drift detection (FIMT-DD). The pseudo code of the algorithm is given in Fig. 1.

The algorithm starts with an empty leaf and reads examples in the order of arrival. Each example is traversed to a leaf where the necessary statistics are updated. Given the first portion of instances,[1] the algorithm finds the best split for each attribute, and then ranks the attributes according to some evaluation measure. If the splitting criterion is satisfied it makes a spit on the best attribute, creating two new leafs, one for each branch of the split. Upon arrival of new instances to a recently created split, they are passed down along the branches corresponding to the outcome of the test in the split for their values. Change detection tests are updated with every example from the

[1] Since a major difference is not expected to be observed after each consecutive example, attributes are evaluated and compared after consecutive chunks of $N_{\min}$ data points (e. g., chunks of 200 examples).

stream. If a change is detected, an adaptation of the tree structure will be performed. Details of the algorithm are given in the next sections.

3.1 Splitting criterion

In the literature, several authors have studied the problem of efficient feature, attribute or model selection over large databases. The idea was first introduced by Musick et al. (1993) under the name of decision theoretic sub-sampling, with an immediate application to speed up the basic decision tree induction algorithm. One of the solutions they propose, which is relevant for this work, is the utilization of the *Hoeffding bound* (Hoeffding 1963) in the attribute selection process in order to decide whether the best attribute can be confidently chosen on a given subsample. Before introducing the details on how the bound is used in FIMT-DD, we will first discuss the evaluation measure used in the attribute selection process.

To calculate the merit of all possible splitting tests, the algorithm must maintain the required statistics in the leaves. Due to the specific memory and time constraints when processing data streams, we propose the Standard Deviation Reduction (SDR) measure which can be efficiently computed in an incremental manner. Given a leaf where a sample of the dataset S of size N has been observed, a hypothetical binary split h_A over attribute A would divide the examples in S in two disjoint subsets S_L and S_R, with sizes N_L and N_R correspondingly ($S = S_L U S_R$; $N = N_L + N_R$). The formula for the SDR measure of the split h_A is given below:

$$\mathrm{SDR}(hA) = sd(S) - \frac{NL}{N} sd(S_L) - \frac{NR}{N} sd(S_R) \tag{1}$$

$$sd(S) = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N}(yi - \bar{y})^2\right)} = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} yi^2 - \frac{1}{N}\left(\sum_{i=1}^{N} yi\right)^2\right)} \tag{2}$$

From formula (2), it becomes clear that it suffices to maintain the sums of y values (values of the predicted attribute) and squared y values, as well as the number of examples that have passed through that leaf node.

Let h_A be the best split (over attribute A) and h_B the second best split (attribute B). Let us further consider the ratio of the SDR values for the best two splits (h_A and h_B) as a real-valued random variable r. That is:

$$r = \mathrm{SDR}(h_B)/\mathrm{SDR}(h_A) \tag{3}$$

The values of the variable r therefore vary between 0 and 1. Let us further observe the ratios between the best two splits after each consecutive example from the stream. Each observed value can be considered as real-valued random variable $r_1, r_2, \ldots, r_N$.

Having a predefined range for the values of the random variables, the *Hoeffding* probability bound (Hoeffding 1963) can be used to obtain high confidence intervals for the true average of the sequence of random variables. The probability bound enables us to state with confidence $1 - \delta$ that the sample average for N random i.i.d. variables

with values in the range R is within distance ε of the true mean. The value of ε is computed using the formula:

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2N}} \tag{4}$$

The value of ε monotonically decreases with the number of observations N, giving the flavor of the central limit theorem to the approach. By observing more and more values, the sampled mean approaches the true mean.

The main benefit of using the *Hoeffding bound* is that, it gives an exponential decay of the probability for the sum of random variables to deviate from its expected value, as moving away from the mean. This is because the bound is based on the theory of large deviations. Let us denote the upper and lower bounds on the estimated sample average as:

$$\bar{r}^+ = \bar{r} + \varepsilon \quad \text{and} \quad \bar{r}^- = \bar{r} - \varepsilon \quad \text{and} \quad \bar{r}^- \leq r_{\text{true}} \leq \bar{r}^+ \tag{5}$$

If the upper bound of the sample average is below 1 then the true mean r_{true} is also below 1, so at the moment of observation, the best observed attribute over a portion of the data is really the best over the whole distribution. Therefore with confidence $1 - \delta$ the split h_A is deemed as the best one. In this case, the splitting criterion is satisfied and the split h_A can be applied.

However, very often we have a situation when two or more splits are very similar or even identical with respect to their SDR values. In such situations, if the confidence intervals determined by ε have shrunk substantially and still one cannot make a difference between the best splits, choosing any of them would be equally satisfying. The stopping rule can be then defined as a threshold τ for ε with which the user is satisfied. If the error becomes smaller than τ (e.g. $\tau = 0.05$) and the splitting criterion is still not satisfied, it is a tie situation. Since both of the competing attributes are equally good, the split is made on the one with a higher SDR value.

3.2 Numerical attributes

The efficiency of the split selection procedure is highly dependent on the number of possible split points. For numerical attributes with a large number of distinct values, both memory and computational costs can be very high. The common approach in the batch setting is to perform a preprocessing phase, typically partitioning the range of numerical attributes (discretization). This requires an initial pass of the data prior to learning, as well as sorting operations.

Preprocessing is not an option with streaming data and sorting can be very expensive. The range of possible values for numerical attributes is also unknown and can vary in case of sampling shift. For classification tasks on data streams, a number of interesting solutions have been proposed: on-line discretization (with a pre-specified number of bins) (Domingos and Hulten 2000), Gaussian-based methods for two-class problems (Gama et al. 2004) and an equi-width adaptation to multi-class

problems (Pfahringer et al. 2008) and an exhaustive method based on binary search trees (Gama et al. 2003). They are either sensitive to skewed distributions or are appropriate only for classification problems. We have developed a time-efficient method for handling numerical attributes based on a E-BST structure, which is an adaptation of the exhaustive method proposed in Gama et al. (2003), tailored for regression trees.

The basis of our approach is to maintain an extended binary search tree structure (E-BST) for every numerical attribute in the leaves of the regression tree. The E-BST holds the necessary statistics and enables us to sort values on the fly. Each node in the E-BST has a test on a value from the attribute range (in the form: $\leq key$). The *keys* are the unique values from the domain of the corresponding numerical attribute. Further, each node of the E-BST structure stores two arrays of three elements. The first array maintains statistics for the instances reaching the node with attribute values less than or equal to the *key*: (1) a counter of the number of instances that have reached the node, (2) the sum of their target attribute values $\sum y$ and (3) the sum of the squared target attribute values $\sum y^2$. The second array maintains these statistics for the instances reaching the node with attribute values greater than the *key*.

The E-BST structures are incrementally updated with every new example that is reaching the leaf of the regression tree where they are stored. The process starts with an empty E-BST. Unseen values from the domain of the attribute are inserted in the structure as in a binary search tree. The main difference is that while traversing the E-BST, the counts in all the nodes on the way are additionally updated.

As an example, consider the E-BST structure in Fig. 2. It is constructed from the following sequence of (numerical attribute, target attribute) pairs of values: {(0.95, 0.71), (0.43, 0.56), (7.44, 2.3), (1.03, 0.18), (8.25, 3.45)}. The root holds the key 0.95 since it is the first from the sequence. The first column which corresponds to the case $value \leq key$ is initialized with the values $(0.71, 0.71^2, 1)$. The next value 0.43 is a new key inserted to the left of the root. The first column of the new node is initialized with $(0.56, 0.56^2, 1)$ and the first column of the root is also updated with the values $(0.71 + 0.56, 0.71^2 + 0.56^2, 2)$. This procedure continues for the remaining pairs of values. The last value inserted in the tree will update the second column of the root node (0.95) and the node (7.44).

To evaluate the possible splits, the whole tree must be traversed in an in-order fashion (in order to obtain a sorted list of distinct values). The computation of the SDR measure for each possible splitting point is performed incrementally. This is possible because all the nodes in the structure have the information about how many examples have passed that node with attribute values less or equal and greater than the key, as well as the sums of their target attribute values.

The algorithm for finding the best splitting point is given in Fig. 3. *FindBestSplit* is a recursive algorithm which takes as input a pointer to the E-BST. As it moves through the tree in in-order it maintains the sum of all target attribute values for instances falling to the left or on the split point (*sumLeft*), the sum of all target attribute values for instances falling to the right of the split point (*sumRight*), the corresponding sums of squared target attribute values (*sumSqLeft* and *sumSqRight*), the number of instances falling to the right of the split point (*rightTotal*), and the total number of instances observed in the E-BST structure (*total*). From these, the SDR statistics for each split

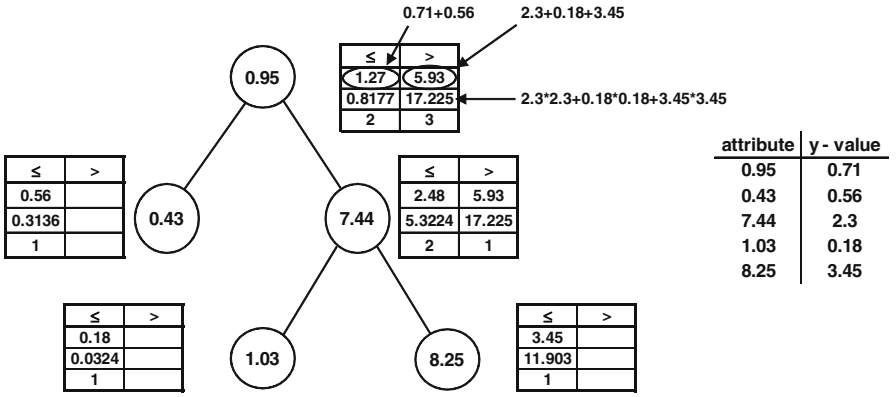

Fig. 2 Illustrative example of a E-BST constructed from the given table of pair of values (*right*)

point is calculated. When returning from the left or the right child node, these counts must be returned as they were at the parent node.

Insertion of a new value in the E-BST structure has time complexity $O(\log(n))$ on average and $O(n)$ in the worst case, where n is the number of unique values seen thus far in the leaf. In the case of a balanced E-BST (AVLTree) the worst insertion time is $O(\log(n))$. Determining the best split point has a time complexity of $O(n)$.

While the tree provides time efficiency, its memory consumption in some situations can be a concern. For reducing the memory requirements, several strategies can be employed. For example, if we do not maintain a balanced tree, the second array for the instances falling to the right of the split point can be removed, and this information can be kept only at the root node. Another option is reducing the precision of the real values stored as unique keys. Rounding to three decimal places can substantially reduce the amount of memory allocated.

To further reduce the memory requirements, we propose a method for disabling bad split points that will result in dropping parts of the E-BST structure. The procedure is initiated whenever the splitting criterion is not satisfied. Let $\mathrm{SDR}(h_1)$ be the greatest reduction in the standard deviation measured at the last evaluation and $\mathrm{SDR}(h_2)$ the second highest reduction over a different attribute. Let $r = \mathrm{SDR}(h_2)/\mathrm{SDR}(h_1)$ be the corresponding ratio. Every split point with $\mathrm{SDR}(h_i)/\mathrm{SDR}(h_1) < r - 2\varepsilon$ is considered bad.

The rationale of this method is as follows: If the upper bound of the ratio (for any split point) is smaller than the lower bound of r, then the true reduction of that split relative to the best one is definitely smaller. Therefore, every other split point with a ratio below this value is not a competing point and can be safely removed. The algorithm starts from the leftmost node of the tree structure that has been updated with at least 5% of the examples observed in the whole structure.

The SDR values of the split points are computed the same way as previously, using the algorithm *FindBestSplit*. If the children of a node that has to be removed are good split points, or only the right child remained, its key will be replaced with the key of the in-order successor. With this action the arrays of the successor and all the predecessor

```
Global variables: //intialization
     sumTotalLeft = 0
     sumTotalRight = sumLeft + sumRight
     sumSqTotalLeft = 0
     sumSqTotalRight = sumSqLeft + sumSqRight
     rightTotal = total = left + right

Algorithm _ComputeSDR()
Input: // Global variables
Output: SDR computed using formula (1) where N_L = total − rightTotal and N_R =
rightTotal and the corresponding sums for the left subset (sumTotalLeft and
sumSqTotalLeft) and for the right subset (sumTotalRight and sumSqTotalRight)

Algorithm FindBestSplit(Btree)
Input: root
Output:
     maxSDR //standard deviation reduction of the best split
     splitPoint //value of the best split
Begin
     If LeftChild exists
               Call FindBestSplit(LeftChild)
     //update the sums and counts for computing the SDR of the split
     sumTotalLeft ← sumTotalLeft + sumLeft
     sumTotalRight ← sumTotalRight - sumLeft
     sumSqTotalLeft ← sumSqTotalLeft + sumSqLeft
     sumSqTotalRight ← sumSqTotalRight - sumSqLeft
     rightTotal ← rightTotal - left
     If maxSDR < _ComputeSDR(current node)
               Update maxSDR
               splitPoint gets the key from the current node
     If RightChild exists
               Call FindBestSplit(RightChild)
     //update the sums and counts for returning to the parent node
     sumTotalLeft ← sumTotalLeft - sumLeft
     sumTotalRight ← sumTotalRight + sumLeft
     sumSqTotalLeft ← sumSqTotalLeft - sumSqLeft
     sumSqTotalRight ← sumSqTotalRight + sumSqLeft
     rightTotal ← rightTotal + left
End
```

Fig. 3 Pseudo code for the FindBestSplit algorithm for finding the best split point

nodes have to be updated correspondingly. If the node that has to be removed does not have an in-order successor, then we use the key from the in-order predecessor: In this case, the updating strategy is more complicated, because all the predecessor nodes on the path to the node (being removed) have to be updated.

When a node is removed from the tree no information is lost because its sums and counts are held by upper nodes. This method enables us to save a lot of memory without losing any relevant information. A simpler and less costly procedure is to prune only the nodes whose children are all bad split points. In this case no updating is necessary. Figure 4 gives an illustrative example of an E-BST with split points disabled by using the simpler procedure. The threshold is 1.6 and the colored (grey) keys (30, 34, 36, 37, 56, 67, and 69) are removed from the tree, leaving as possible split points the uncolored keys (25, 29, 38, 40, 45, 48, and 50).

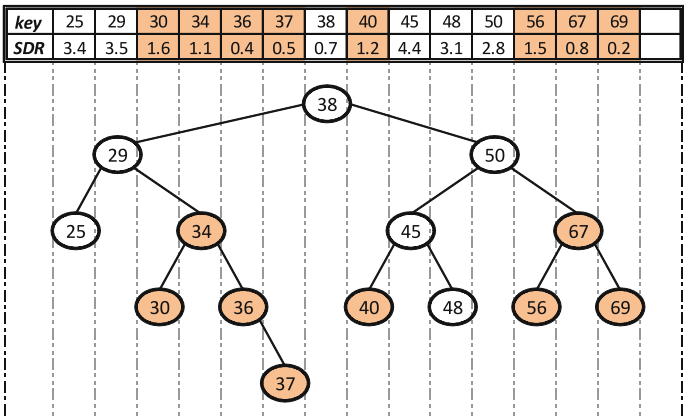

Fig. 4 Illustrative example of an E-BST and disabled regions with bad split points

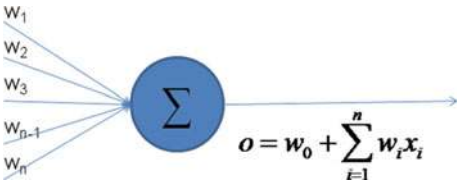

Fig. 5 Illustrative example of a perceptron without activation function

3.3 Linear models in leaves

In this section, we discuss the process of inducing linear models in the leaves. Existing batch approaches compute the linear models ether in the pruning phase or in the growing phase. In the later approach, the algorithms need to perform heavy computations necessary for maintaining the pre-computed linear models for every possible split point. While efforts have been made in reducing the computational complexity, we observe that none of the proposed methods would be applicable when dealing with high speed data streams, which are described by many numerical attributes having large domains of unique values. For this reason, we propose the most lightweight method for inducing linear models, based on the idea of on-line training of perceptrons. The trained perceptrons will represent the linear models fitted separately in each sub-space of the instance-space.

An important difference between our proposed method and the batch ones is that the process of learning linear models in the leaves will not explicitly reduce the size of the regression tree. The split selection process is invariant to the existence of linear models in the leaves. However, if the linear model fits well the examples assigned to the leaf, no further splitting would be necessary and pre-pruning can be applied.

The basic idea is to train perceptrons in the leaves of the tree by updating the weights after each consecutive example. We use the simplest approach: no attribute selection is performed. All the numerical attributes are included in the regression equation, which is represented by a perceptron without an activation function. The weights of the links are the parameters of the linear equation. This is illustrated in Fig. 5.

For training the perceptrons, we propose the incremental (stochastic) gradient descent method, using the same error as in the *Perceptron rule*, i.e. the mean squared error. The main difference is that the weights are updated with every example arriving in the leaf instead of using the whole set. For updating the weights, we use the *Delta rule* (also known as *Widrow–Hoff rule*), for which it is known that the learning process converges even in the case when the examples do not follow a linear model.

When a new example arrives, the output is computed using the current weights. Each weight (except w_0) is next updated with the product of: the difference between the output (o) and the real value (y), the normalized value of the corresponding attribute (x_i) and the learning rate (η).

$$w_i \leftarrow w_i + \eta(o - y)x_i, \quad i \neq 0. \tag{6}$$

The weights are initially set for the root node to random real valued numbers in the range $[-1, 1]$. The learning rate can be kept constant or it can decrease with the number of examples seen during the process of learning. Details on this process are given in Appendix B. If we would like the learning rate to be constant, it is usually set to some small value (e.g., 0.01). The perceptrons can handle only numerical attributes. Categorical variables can be transformed into a set of binary (numerical) variables before learning. The values of the attributes should be normalized (standardized) before the process of learning, so each of them will have the same influence during the process of training. The normalization can be performed incrementally by maintaining the sums of target values and their squares from the beginning of learning. We use a variant of the studentized residual equation, where instead of dividing with one standard deviation (σ) we divide by three standard deviations. The equation is given in formula (7).

$$x_i' = \frac{x_i - \bar{x}}{3\sigma}. \tag{7}$$

The learning phase of each perceptron is in parallel with the process of growing a node, and ends with a split or with transforming the growing node into a leaf. If a split was performed, the linear model in the splitting node is passed down to the child nodes, avoiding the need of training from scratch: The learning continues in each of the new growing nodes, independently and according to the examples that will be assigned. This can be seen as fine tuning of the linear model in the corresponding sub-region of the instance-space.

3.4 Change detection

When local concept drifts occur, most of the existing methods discard the whole model simply because its accuracy on the current data drops. Despite the drop in accuracy, parts of the model can still be good for the regions not affected by the drift. In such situations, we propose to update only the affected parts of the model. An example of a system that possesses this capability is the CVFDT system (Hulten et al. 2001). In CVFDT, splitting decisions are repeatedly re-evaluated over a window of most recent

examples. This approach has a major drawback: maintaining the necessary counts for class distributions at each node requires a significant amount of additional memory and computation (especially when the tree becomes large). We address this problem by using a lightweight on-line change detection test for continuous signals. Discussed ideas are applicable for both of the options: with or without linear models in the leaves.

3.4.1 Detection

The change detection mechanism that we propose is on-line and enables local change detection. The basic idea is to monitor the evolution of the error at every region of the instance space, and inform about significant changes that have affected the model tree locally or globally. The detection and adaptation to changes can be done simultaneously and independently in different parts of the instance space.

Each node of the tree is associated with a change detection test. The intuition behind this is simple: each node with the sub-tree below covers a region (a hyper-rectangle) of the instance space. When concept drift occurs in some part of the instance space, the model will not correspond to reality and the error over that part of the instance space will increase.

This suggests a simple method to detect changes: monitor the evolution of the error at each node. If it starts increasing, this may be a sign of change. To confidently tell that there is a significant increase in the error due to concept drift, we propose to use the Page–Hinckley (PH) change detection test (Mouss et al. 2004).

The PH test is a sequential adaptation of the detection of an abrupt change in the average of a Gaussian signal. This test considers two variables: a cumulative variable m_T, and its minimum value M_T after seeing T examples. The first variable, m_T, is defined as the sum of differences between the observed values and their mean accumulated until the current moment, where T is the number of all examples seen so far and x_t is the variable's value at time t:

$$m_T = \sum_{t=1}^{T} (x_t - \bar{x}_t - \alpha) \tag{8}$$

$$\bar{x}_t = \frac{1}{t} \sum_{l=1}^{t} x_l \tag{9}$$

$$M_T = \min\{m_T, t = 1, \ldots, T\}. \tag{10}$$

At every moment, the PH test monitors the difference between the minimum M_T and m_T:

$$PH_T = m_T - M_T. \tag{11}$$

When this difference is greater than a given threshold (λ), it triggers an alarm about a change in the distribution (i.e., $PH_T > \lambda$). The threshold parameter λ depends on the admissible false alarm rate. Increasing λ entails fewer false alarms, but might miss some changes.

The parameter α corresponds to the minimal absolute value of the amplitude of the jump to be detected. It should be adjusted according to the standard deviation of the target attribute. Larger deviations will require larger values for α. On the other hand, large values can produce larger delays in detection.

Our experiments have confirmed that 10% of the standard deviation gives best results, but the optimal case is achieved in combination with the value of λ. As a general guideline, when α has a smaller value λ should have a larger value: This is to reduce the number of false alarms. On the other hand, when α has a larger value λ should have a smaller value: This is to shorten the delays of change detection.

To standardize these parameters for any problem we normalize the value of the target and the prediction from the node before computing the absolute error. Additional experiments have shown that equally good results are achieved for the values of $\alpha = 0.005$ and $\lambda = 50$ for all the real-world and simulated non-stationary datasets. If we would like to tolerate larger variations, λ can be set to 100. For more sensible change detection if the user is willing to accept the risk of increased false alarms, the value of α can be set to 0.001.

The variable x_t is in our case the absolute error at a given node observed at time t (i.e., $|y_t - p_t|$ where p_t is the prediction at time t). For computing the absolute loss we have considered two possible alternatives: (1) using the prediction from the node where the PH test is performed, or (2) using the prediction from the leaf node where the example will be assigned. As a consequence of this we propose two methods for change detection:

1. *Top-Down (TD) method*: If the error is computed using the prediction from the current node, the computation can be performed while the example is passing the node on its path to the leaf. Therefore, the loss will be monitored in the direction from the top towards the "bottom" of the tree.
2. *Bottom-Up (BU) method*: If the error is computed using the prediction from the leaf node, the example must first reach the leaf. The computed difference at the leaf will be then back-propagated to the root node. While back-propagating the error (re-tracing the path the example took to reach the leaf) the PH tests located in the internal nodes will monitor the evolution.

The idea for the BU method came from the following observation: When moving towards the "bottom" of the tree, predictions in the internal nodes become more accurate (as a consequence of splitting). Using more accurate predictions will emphasize the error in case of drift, shorten the delays and reduce the number of false alarms. This was investigated and confirmed by the empirical evaluation, which showed that with the TD method most of the false alarms were triggered at the root node and its immediate descendants.

3.4.2 Adaptation

There are three possible adaptation strategies for model trees:

1. The first adaptation strategy consists of further splitting and growing new sub-trees to the old structure. Although the predictions might improve, the structure of the tree would likely be wrong and misleading.

2. The second adaptation strategy is to prune the parts of the tree where concept drift was detected. In case the change was properly detected, the actual model will be most consistent with the data and the current concept. If a change has been detected relatively high in the tree, pruning the sub-tree will decrease significantly the number of leaves which will lead to unavoidable and drastic short-time degradation in accuracy. In these circumstances, an outdated sub-tree may still give better predictions than a single leaf.
3. The third adaptation strategy is to build an alternate sub-tree for the region where drift is detected. When a change is detected in a given region, the node will be marked for re-growing. The re-growing process will initiate a new alternate tree rooted at the node, which will grow in parallel with the old one. Every example that will reach a marked node will be used for growing both of the trees. The old tree will be kept and grown in parallel until the new alternate tree becomes more accurate. The nodes in the alternate tree will not perform change detection until the moment the alternate tree replaces the original one.

3.4.3 Discussion on the alternate trees adaptation strategy

The alternate trees will not always replace the original trees. If the detected change is a false alarm or the localization of the drift is improper, the alternate tree might never achieve better accuracy. In order to deal with these situations, we propose to monitor the difference in the accuracy of the alternate tree (*AltTree*) and the original sub-tree (*OrgTree*). This is performed by monitoring the log ratio of the accumulated prequential mean squared error computed for every example observed at the node where both of the trees are rooted.

Let $S_i^{AltTree}$ be the sequence of the prequential accumulated loss (prequential mean squared error) for the alternate tree and $S_i^{OrgTree}$ the corresponding sequence for the original tree. The prequential accumulated loss is the sum of losses accumulated for a sequence of examples. For each example, the model first makes a prediction based on the example's attribute values, and then the example is used to update the actual model. The statistic used to compare these errors is the Q statistic and is computed as $Q_i(OrgTree, AltTree) = log(S_i^{OrgTree}/S_i^{AltTree})$. The sign of Q indicates the relative performance of both models, while its value shows the strength of the difference.

However, it is well known that the prequential error is pessimistic, because it contains all the errors from the beginning of the learning, when the model is still not fully induced. These long term influences hinder the detection of recent improvement. Gama et al. (2009) propose a formula for using fading factors with the Q_i statistic

$$Q_i^{\alpha}(OrgTree, AltTree) = \log\left(\frac{L_i(OrgTree) + fS_{i-1}^{OrgTree}}{L_i(AltTree) + fS_{i-1}^{AltTree}}\right), \qquad (12)$$

where L_i is the current loss (squared error) computed from the last example. The prequential error will be updated after computing the Q_i statistic. The fading factor f

can be set to a value close to 1, for example 0.995. If the Q_i statistic has values greater than zero, the original tree has bigger error than the alternate tree.

The Q_i statistic can be evaluated on a user predetermined evaluation interval $T_{\min}$ (e.g., $T_{\min} = 100$ examples). If it is positive, the new alternate tree will replace the old one. The old tree will be removed, or can be stored for possible reuse in case of reoccurring concepts. If the detected change was a false alarm or it was not well localized, the Q_i statistic will increase slowly, and possibly will never reach the value of zero. For this reason, we need to monitor its average. If instead of moving towards zero it starts to decrease, the alternate tree is not promising any improvement to the current model and should be removed.

In addition, a separate parameter can be used to specify a time period in which a tree is allowed to grow. When the time period for growing an alternate tree has passed or the average has started to decrease, the alternate tree will be removed from the node and the allocated memory will be released. In order to prevent premature discarding of the alternate tree, we start to monitor the average of the Q_i statistic after $10n_{\min}$ examples have been used for growing and evaluation.

3.5 Discussion of algorithm design choices

The FIMT-DD algorithm[2] is based on a compromise between the accuracy achieved by and the time required to learn a model tree. It therefore offers approximate solutions in real-time. For making splitting decisions, any method can be used that has high confidence in choosing the best attribute, given the observed data. The Hoeffding bound was chosen due to its nice statistical properties and the independence of the underlying data distribution. The growing process is stable because the splitting decisions are supported statistically, so the risk of overfitting is low. This is an advantage over batch algorithms, where splits in the lower levels of the tree are chosen using smaller subsets of the data.

To ensure the any-time property of the model tree, we chose perceptrons as linear models in the leaves. This approach does not reduce the size of the model tree, but improves its accuracy by reducing the bias as well as the variance component of the error.

The choice of the change detection mechanism was supported by three arguments: The method is computationally inexpensive, performs explicit change detection, and enables local granular adaptation. Change detection requires the setting of several parameters, which enable the user to tune the level of sensitivity to changes and the robustness.

4 Experimental evaluation

The experimental evaluation was designed to answer several questions about the algorithm's general performance:

[2] The software is available for download at:http://kt.ijs.si/elena_ikonomovska/.

1. What is the quality of models produced by FIMT-DD as compared to models produced by batch algorithms?
2. What is the advantage/impact of using linear models in the leaves?
3. How stable are the incrementally learnt trees?
4. What is the bias-variance profile of the learned models?
5. How robust is the algorithm to noise?
6. How the learned models evolve in time?
7. What is the impact of constrained memory on the model's accuracy?
8. Is FIMT-DD resilient to type-II errors?
9. How fast can the algorithm detect change?
10. How well can it adapt the tree after concept drift?

4.1 Experimental setup

4.1.1 Parameter settings

For all the experiments, we set the input parameters to: $\delta = 0.01$, $\tau = 0.05$ and $N_{\min} = 200$. All algorithms were run on an Intel Core2 Duo/2.4 GHz CPU with 4 GB of RAM. All of the results in the tables are averaged over 10 runs. For all the simulations of drift and for the first real dataset, we use the values $\alpha = 0.005$ and $\lambda = 50$ and $T_{\min} = 150$. For the second real dataset we set the values $\alpha = 0.01$ and $\lambda = 100$ for change detection and $T_{\min} = 100$.

4.1.2 Error metrics

The accuracy is measured using the following metrics: mean error (ME), mean squared error (MSE), relative error (RE), root relative squared error (RRSE), and the correlation coefficient (CC). We further measure the total number of nodes, and the number of growing nodes. When learning under concept drift, we measure the number of false alarms, miss-detections and the delays.

4.1.3 Evaluation methodology

Traditional evaluation methodologies used for batch learning are not adequate for the streaming setting, because they don't take into consideration the evolution of the model in time. Several evaluation methodologies for data streams have been proposed in the literature. Two major approaches are the use of a holdout evaluation and the use of a prequential evaluation.

The first is a natural extension of batch holdout evaluation. To track model performance over time, the model can be evaluated periodically, using a separate holdout test set. When no concept drift is assumed, the test set can be static and independently generated from the same data distribution. In scenarios with concept drift, the test set should consist of new examples from the stream that have not yet been used to train the learning algorithm. This is the most preferable method for comparison with other algorithms for which there is no other incremental evaluation

method available, because the learner is tested in the same way. The strategy is to use a sliding window with a 'look ahead' procedure to collect examples from the stream for use as test examples. After testing is complete, the examples are given to the algorithm for additional training according to the size of the sliding step.

The second approach is the use of predictive sequential or *prequential* error (Dawid 1984). The prequential approach has been used in Hulten et al. (2001). It consists of an interleaved test-than-train methodology for evaluation. Its main disadvantage is that it accumulates mistakes over the entire period of learning. Due to this, the learning curve is pessimistic and current improvements cannot be easily seen due to early mistakes. To reduce the influence of early errors, Gama et al. (2009) use a *fading factor*. The fading-factor method is very efficient since it requires setting up of only one parameter. Another alternative is to use a sliding window over the prequential error, which will give a clear picture on the performance of the algorithms over the latest examples used for training.

According to the above discussion, we propose two incremental evaluation methods: (1) *prequential-window* and (2) *holdout-window*. In the first case, the prequential error metrics are computed over a window of N most recent examples. In the second case, the error metrics are computed over a window of examples which have not yet been used for training. The size of the sliding window determines the level of aggregation and it can be adjusted to the size the user is willing to have. The sliding step determines the level of details or the smoothness of the learning curve. The holdout-window methodology provides the least biased evaluation, but can be less practical for real-life situations.

4.2 Datasets

4.2.1 Artificial data

To evaluate the algorithm on large enough datasets, the only feasible solution is to use artificial data. We use three well known artificial datasets: the Fried dataset used by Friedman (1991) and the Losc and Lexp datasets proposed by Karalic (1992). The Fried dataset contains 10 continuous predictor attributes with independent values uniformly distributed in the interval [0, 1], from which only five attributes are relevant and the rest are redundant. The Losc and Lexp datasets contain five predictor attributes, one discrete and four continuous, all relevant. These datasets are used in the first part of the evaluation on stationary streams.

4.2.1.1 Noise. The noise introduced in the artificial datasets is a perturbation factor which shifts attributes from their true value, adding an offset drawn randomly from a uniform distribution, the range of which is a specified percentage of the total value range.

4.2.1.2 Concept drift. The Fried dataset is used also in the second part of the evaluation, for simulating different types of drift. The simulations allow us to control the relevant parameters and to evaluate the drift detection and adaptation mechanisms. Using the Fried data set, we simulate and study three scenarios for concept drift:

1. *Local expanding abrupt drift*: The first type of simulated drift is local and abrupt. The concept drift appears in two distinct regions of the instance space. There are three points of abrupt change in the training dataset, the first one at 1/4 of the examples, the second one at 1/2 of the examples and the third at 3/4 of the examples. At every consecutive change the region of drift is expanded.
2. *Global reoccurring abrupt drift*: The second type of simulated drift is global and abrupt. The concept drift appears over the whole instance space. There are two points of concept drift, the first of which occurs at 1/2 of the examples and the second at 3/4 of the examples. At the second point of drift the old concept reoccurs.
3. *Global and slow gradual drift*: The third type of simulated drift is global and gradual. The first occurrence of the gradual concept drift is at 1/2 of examples of the training data. Starting from this point, examples from the new concept are being gradually introduced among the examples from the old concept. After 100k examples the data contains only the new concept. At 3/4 of examples a second gradual concept drift is initiated on the same way. Examples from the new concept will gradually replace the ones from the last concept like before. The drift ends again after 100k examples.

For all these scenarios, details of the functions and the regions can be found in Appendix C.

4.2.2 Real data

The lack of real-world datasets for regression large enough to test FIMT-DD's performance has restricted us to only 10 datasets from the UCI Machine Learning Repository (Blake et al. 1999) and other sources. The smallest dataset is of size 4,177 instances, while the largest of size 40,769 instances. The datasets are multivariate with nominal and numeric attributes. All of these datasets were used for the evaluation on stationary problems, because none of them has been collected with the aim of an online learning or an incremental analysis.

For the evaluation of learning on non-stationary streaming real-world problems, we have prepared a dataset by using the data from the Data Expo competition (2009). The dataset consists of a large amount of records, containing flight arrival and departure details for all the commercial flights within the USA, from October 1987 to April 2008. This is a large dataset with nearly 120 million records (11.5 GB memory size). The dataset was cleaned and records were sorted according to the arrival/departure date (year, month, and day) and time of flight. Its final size is around 116 million records and 5.76 GB of memory.

We use this dataset for the first time as a supervised machine learning (prediction) problem, therefore we make a choice of attributes on the basis of the Data Expo experience with this data. It has been reported that the arrival delay is dependent on the day of the week, the month, the time of the day and the carrier, the origin and the destination, among other factors. The 13 attributes are therefore the following: *Year*,*Month*, *Day of Month*, *Day of Week*, *CRS Departure Time*, *CRS Arrival Time*, *Unique Carrier*, *Flight Number*, *Actual Elapsed Time*, *Origin*, *Destination*, *Distance*, and *Diverted*. Details

on their meaning can be found in Data Expo (2009). The target variable is the *Arrival Delay*, given in seconds.

5 Results

In this section, we present the results of the evaluation and sensitivity analysis of the FIMT-DD algorithm. The results come in three parts. The first part is the evaluation over stationary data streams and the second part over non-stationary data streams. The last part is the analysis over the real-world non-stationary dataset, which is suspected to contain concept drift and noise.

5.1 Results on stationary streams

On the stationary data, we compare our algorithm's performance, stability and robustness to several state-of-the-art batch algorithms for learning regression/model trees.

5.1.1 Quality of models

Since the 10 real datasets used are relatively small, the evaluation was performed using the standard method of ten-fold cross-validation, using the same folds for all the algorithms included. To make a fair analysis and evaluate the impact of linear models in the leaves, two separate comparisons were performed.

Four basic versions of our algorithm are compared among themselves and to other algorithms: the basic version FIMT-DD includes both linear models in the leaves and change detection, FIRT-DD has no linear models in the leaves, FIMT has no change detection and FIRT has no linear models in the leaves and no change detection. First, FIRT was compared to CART (Breiman et al. 1998) from R (default settings), and M5′ (Quinlan 1992) from WEKA (default settings) with the regression-tree option (M5′RT). Second, FIMT was compared to M5′ from WEKA (default settings) with the model-tree option (M5′MT), linear regression method LR from WEKA (default settings), the commercial algorithm CUBIST (2009) (default settings) from Quinlan/RuleQuest, and the four algorithms for learning model trees proposed in Potts and Sammut (2005): two versions of batch learners (BatchRD and BatchRA) and two of incremental learners (OnineRD and OnlineRA). We also used two versions of FIMT: FIMT_Const with a constant learning rate and FIMT_Decay with a decaying learning rate (as described in Appendix B).

The performance measures for CART, M5′ and FIRT averaged across the 10 datasets are given in Table 1. The detailed (per dataset) results are given in Table 9 (Appendix A). The comparison of regression tree learners shows that FIRT has similar accuracy (RRSE, CC) to CART and lower than M5′RT. These results were somewhat expected, since these datasets are relatively small for the incremental algorithm. FIRT produces medium sized models (on average 35 leaves), which are larger than CART trees (on average four leaves), but significantly smaller than the M5′RT trees (on average 245 leaves). The Nemenyi test confirmed that there is no statistically significant differ-

Table 1 Results from tenfold cross-validation averaged over the real datasets for regression tree learning algorithms

Algorithm	RE%	RRSE%	Leaves	Time (s)	CC
M5′RT	47.30	51.72	244.69	29.28	0.82
CART	75.63	62.77	4.18	9.67	0.75
FIRT	55.38	60.11	35.54	0.33	0.75

Table 2 Results from tenfold cross-validation averaged over the real datasets for model tree learning algorithms

Algorithm	RE%	RRSE%	Leaves	Time (s)	CC
M5′MT	42.10	46.09	76.71	29.54	0.85
LR	60.48	63.01	1.00	2.59	0.74
CUBIST	48.75	–	20.67	1.02	0.75
FIMT_Const	60.21	104.01	35.54	0.42	0.70
FIMT_Decay	59.40	74.11	35.54	0.42	0.73
BatchRD	41.84	45.27	55.58	6.13	0.85
BatchRA	50.22	53.39	22.85	2.24	0.81
OnlineRD	49.77	53.26	9.98	32.67	0.80
OnlineRA	48.18	52.03	12.78	3.08	0.82

ence (in terms of the correlation coefficient) between FIRT and CART: this was also confirmed with a separate analysis using the Wilcoxon Signed-Ranks test.

The performance results of model-tree learners averaged across the 10 datasets are given in Table 2, while detailed (per dataset) results can be found in Table 10 for the relative error (RE%), Table 11 for the size of the models (leaves) and Table 12 for the learning time (Appendix A). The comparison of model tree learners shows that FIMT_Decay and FIMT_Const have similar accuracy to LR, while the rest of the learners are better (BatchRD being the best). The analysis on the statistical significance of results (Friedman test for comparing multiple classifiers) followed by the Nemenyi post-hoc test confirmed this, additionally showing that there is no significant difference between FIMT and OnlineRA, OnlineRD and BatchRA methods. There is a significant difference in accuracy between M5′MT, CUBIST and BatchRD over FIMT and LR, while no significant difference over BatchRA, OnlineRA and OnlineRD. It was also observed that linear models in the leaves rarely improve the accuracy of the incremental regression tree on these small real datasets. This could be due to the fact that the regression surface is not smooth for these datasets. The biggest models on average are produced by M5′MT and BatchRD, however in this comparison CUBIST is a special case because it has a special mechanism for creating rules from the tree. Online learners have lower accuracy than their batch equivalents. FIRT-DD has a significant advantage in speed over all of the learners. CUBIST is in the second place. Another advantage of FIMT-DD is that it can learn with a limited amount of memory (as allowed by the user), while the other algorithms do not offer this option.

Table 3 Results from holdout evaluation averaged over the artificial datasets 1000k/300k

Algorithm	RE%	RRSE%	Leaves	Time (s)	CC
FIRT	0.16	0.19	2452.23	24.75	0.98
CUBIST	0.13	–	37.50	104.79	0.98
FIMT_Const	0.11	0.14	2452.23	27.11	0.98
FIMT_Decay	0.11	0.14	2452.23	26.93	0.98
LR	0.46	0.51	1.00	2468.61	0.83
BatchRD	0.08	0.10	27286.30	5234.85	0.99
BatchRA	0.71	0.69	56.97	2316.03	0.51
OnlineRD	0.10	0.13	6579.50	3099.82	0.98
OnlineRA	0.70	0.68	57.77	2360.56	0.53

For the artificial problems (Fried, Lexp and Losc), we generated 10 random datasets, each of size 1 million examples, and one separate test set of size 300k examples. All the algorithms were trained and tested using identical folds. The results in Table 3 show that the incremental algorithms are able to achieve better accuracy than the batch algorithms CUBIST, LR and BatchRA. Detailed (per dataset) results can be found in Table 13 (Appendix A). The best algorithm is BatchRD whose models are much larger. BatchRA and OnlineRA give the worst results. The linear models in the leaves improve the accuracy of FIRT because now we are modeling smooth surfaces. The incremental model trees of FIMT_Const and FIMT_Decay have similar accuracy as OnlineRD, but are smaller in size and the time of learning is at least 100 times smaller.

5.1.2 Bias-variance profile

A very useful analytical tool for evaluating the quality of models is the bias-variance decomposition of the mean squared error (Breiman 1998). The bias component of the error is an indication of the intrinsic capability of the method to model the phenomenon under study and is independent of the training set. The variance component of the error is independent of the true value of the predicted variable and measures the variability of the predictions given different training sets. Experiments were performed for all the real datasets over the same folds used in the cross-validation and for the artificial ones over the same training sets and the same test set (as described above in Sect. 5.1.1). The experimental methodology is the following: We train on the 10 independent training sets and log the predictions of the corresponding models over the test set. Those predictions are then used to compute the bias and the variance using the derived formula for squared loss in Geman et al. (1992).

Table 4 gives the bias-variance profile of the models learned by the algorithms FIRT, FIMT_Const, OnlineRD and CUBIST. The results show that the incremental algorithms (FIRT and OnlineRD) have the smallest values for the variance component of the error, which accounts for smaller variability in predictions, given different training sets. This suggests that incrementally built trees are more stable, and are less dependent on the choice of training data.

Table 4 Bias-variance decomposition of the mean squared error for real and artificial datasets

Dataset	FIRT		FIMT_Const		OnlineRD		CUBIST	
	Bias	Variance	Bias	Variance	Bias	Variance	Bias	Variance
Abalone	1.19E+01	0.41E+01	1.22E+01	**0.39E+01**	1.17E+01	0.53E+01	*1.16E+01*	0.49E+01
Ailerons	0.00E+00	0.00E+00	1.00E-06	4.00E-06	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Mv_delve	9.80E+01	**9.68E+01**	9.83E+01	9.83E+01	*9.79E+01*	9.71E+01	*9.79E+01*	9.76E+01
Wind	4.26E+01	**2.64E+01**	4.29E+01	3.59E+01	4.20E+01	3.20E+01	*4.15E+01*	3.18E+01
Cal_housing	1.27E+10	**7.51E+09**	1.25E+10	7.75E+09	1.25E+10	7.93E+09	*1.24E+10*	9.37E+09
House_8L	2.58E+09	1.42E+09	2.59E+09	1.55E+09	2.59E+09	**1.35E+09**	*2.57E+09*	1.39E+09
House_16H	2.61E+09	1.09E+09	2.61E+09	1.21E+09	2.61E+09	**0.93E+09**	*2.59E+09*	1.14E+09
Elevators	0.38E-04	**0.19E-04**	0.59E-04	1.98E-04	*0.37E-04*	0.36E-04	*0.37E-04*	0.26E-04
Pol	1.58E+03	1.42E+03	1.58E+03	1.50E+03	1.60E+03	**0.96E+03**	*1.56E+03*	1.5E+03
Winequality	0.72E+00	**0.21E+00**	0.71E+00	0.31E+00	*0.69E+00*	0.24E+00	*0.69E+00*	0.29E+00
Fried	1.62E+00	9.20E-01	1.40E+00	4.30E-01	**9.80E-01**	**1.30E-02**	1.50E+00	9.00E-02
Lexp	2.75E-02	4.88E-02	**5.57E-04**	1.90E-03	1.25E-03	**1.05E-03**	5.91E-02	1.25E-02
Losc	1.02E-01	2.18E-02	9.61E-02	1.95E-02	1.90E-01	**1.20E-05**	**1.33E-02**	4.78E-03

The best variance value is in bold and best bias value is in italics

On the real datasets, CUBIST has the smallest values for the bias component of the error and FIRT has the smallest values for the variance. The bias of the incremental learners is slightly higher, but this was somewhat expected given the relatively small size of the datasets. For the artificial data, incremental learners have again low variance, and for some of the problems also lower bias. For the Fried dataset, the algorithm OnlineRD is a winner. The Lexp dataset is modeled best by FIMT_Const with a significant difference on the bias component of the error. The Losc dataset is modeled best by CUBIST. On the artificial data most stable is the OnlineRD algorithm.

5.1.3 Sensitivity analysis

5.1.3.1 Stability. We have analyzed the effect of different orderings of examples on the performance of incremental learners, both on the real and artificial datasets. In our experiments, the artificial test and training datasets are of size 300 and 1000k respectively.

We shuffle the training set 10 times and measure the error, the learning time, the model size as well as the variable (attribute) chosen at the root node of the learned tree. The results show that the variance of the root relative squared error over all the real datasets is in the range $[1.3 \times 10^{-5}, 5.6 \times 10^{-4}]$, with a mean value of 0.000215. The root split attribute for most of the datasets and for all the permutations of training examples was same. More precisely, the percentage of cases with a different root split attribute averaged over all 10 datasets is 14%.

5.1.3.2 Robustness to noise. We also studied the sensitivity of the learners to noise in the data. The test set was fixed to 300k examples with 10% of noise. The training set was fixed to 1000k examples and the level of noise was systematically increased from

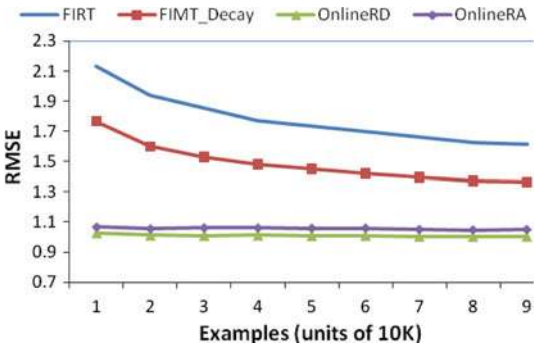

Fig. 6 Comparison of the RMSE evaluated by using the window-holdout method on the Fried dataset

0 to 75%. We measured the accuracy and the model size. From the results, we conclude that the incremental learners respond to noise similarly to the batch learner CUBIST. As the level of noise rises, accuracy drops. The regression trees without linear models in the leaves give best results on the Fried dataset. On the other datasets, there is no significant difference among the algorithms.

5.1.3.3 The Anytime Property. Trees are known for their excellent speed in predicting, despite the size of the model. The traversal time for an example from the root to the leaves of the tree is extremely small. The model induced by FIMT-DD is available for predicting immediately after an initial sequence of examples needed to grow a sufficiently accurate tree. For visualizing the on-time performance of the regression/model trees we use the learning curves obtained with the previously described evaluation methods. The prequential-window and the holdout-window method both converge fast to the holdout method.

In Fig. 6, we give a comparison of the accuracy (RMSE) of FIRT, FIMT_Decay, OnlineRA and OnlineRD algorithms on the Fried dataset by using the holdout-window evaluation. Using the holdout window method, all of the models are tested in the same way, by computing the error only over examples which have not been used for training. In each consecutive testing, more data for learning has been provided. As a response, FIRT and FIMT_Decay increase the complexity of the models, which improves the accuracy. This is not the case for the models induced by OnlineRA and OnlineRD whose accuracy does not improve in time. Both, FIRT and FIMT_Decay show no signs of overfitting, which is not clear for OnlineRA and OnlineRD.

5.1.3.4 Dependency on memory constraints. FIMT-DD algorithm is able to learn with a small constrained amount of available memory: This is achieved with similar memory management utilities as in the VFDT algorithm. When almost all the available memory is allocated, the algorithm starts to search for the leaves with the smallest error estimated on-line and starts to deactivate some of them: In this way, advantage is given to the leaves with the highest error. Ignoring bad split points also saves some memory, as well as ignoring bad attributes (which can produce less accurate models).

To evaluate the algorithm's dependency on memory constraints, we have measured the accuracy and model size at the end of learning for different quantities of available memory. Figure 7 depicts the dependency of accuracy on the available memory for the Fried dataset (the most complex one): The quality of the regression trees does

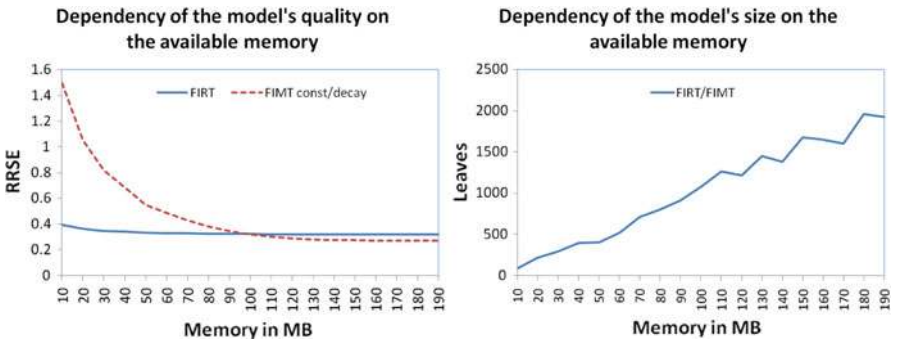

Fig. 7 Illustration of the dependency of the model's quality on the size of the available memory

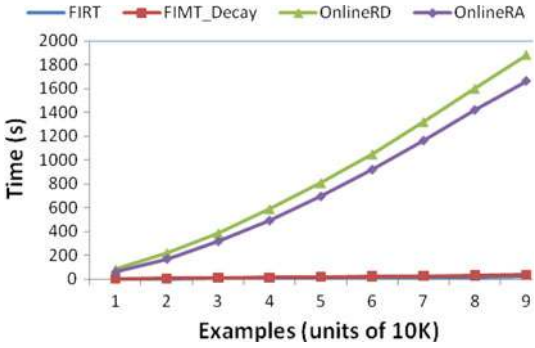

Fig. 8 Comparison on the learning time on the Fried dataset

not significantly depend on the available memory. The models have the same quality with 20 and 200 MB of memory. For the additional option of training perceptrons in the leaves, more memory is required: FIMT achieves the same accuracy as FIRT at ∼100 MB of allocated memory. In conclusion, the incremental algorithms can induce good models using only a small amount of memory.

5.1.3.5 Processing speed. One of the most important characteristics of the proposed algorithm, along with the anytime property and the ability of learning without storing any examples, is its processing speed. The processing speed of FIRT-DD is dependent mostly on the number of attributes, the number of distinct values of each attribute and the frequency of change. However, as a rough indication of the processing abilities of the incremental algorithm we give the processing times for the simulated and the real datasets.

In Fig. 8, we give a comparison of the learning times of FIRT and FIMT_Decay, with those of the algorithms OnlineRD and OnlineRA. We can see that for OnlineRD and OnlineRA, the time of learning increases almost exponentially with the number of examples. On the other hand, FIRT and FIMT_Decay manage to maintain almost constant processing time with an average speed of approximately 24,000 examples per second (e/s). For the real-world datasets, the average processing speed is approximately 19,000 examples per second, without the strategy for growing alternate trees

Table 5 Averaged results from the evaluation of change detection with the pruning adaptation strategy over 10 experiments

			Change point 250k		Change point 500k		Change point 750k	
		#FA's	Global delay	PH delay	Global delay	PH delay	Global delay	PH delay
Local abrupt drift	TD	0.0	12071.0	4006.9	5029.3	1458.2	8631.5	894.1
	BU	0.0	30403.0	1542.1	6967.7	550.3	9519.0	601.9
Global abrupt drift	TD	0.0	NA	NA	1995.6	273.3	7002.7	676.9
	BU	0.0	NA	NA	374.1	374.1	412.5	412.5
Global gradual drift	TD	0.7	NA	NA	21673.8	3473.5	17076.0	6200.6
	BU	0.0	NA	NA	18934.6	18934.6	19294.7	2850.0

enabled. This option would decrease the speed due to an increased number of E-BST structures which need to be maintained for further growing.

5.2 Results on time-changing streams

5.2.1 Change detection

The second part of the experimental evaluation analyzes and evaluates the effectiveness of the change detection methods proposed. The comparison is performed only on the artificial datasets (each with size of 1 million examples), where the drift is known and controllable. This allows us to make precise measurement of delays, false alarms and miss-detections.

Tables 5 and 6 give averaged results over 10 experiments for each of the artificial datasets. We measure the number of false alarms for each point of drift, the Page–Hinckley test delay (number of examples monitored by the PH test before the detection) and the global delay (number of examples processed in the tree from the point of the concept drift until the moment of the detection). The delay of the Page–Hinckley test gives an indication of how fast the algorithm will be able to start the adaptation strategy at the local level, while the global delay shows how fast the change was detected globally. Table 5 contains results on change detection when the pruning strategy is used, while Table 6 contains results when the strategy of growing alternate trees is used.

The general conclusions are that both of the proposed methods are quite robust, and rarely trigger false alarms. An exception is the dataset with gradual concept drift for which the TD method triggers one false alarm in seven of 10 experiments. Both of the methods detect all the simulated changes for all the types of drift. For the local abrupt concept drift, the BU method has larger global delay but smaller PH delay. This indicates that the TD method can enable faster adaptation if localization of drift is proper. The situation is opposite for the global abrupt drift. For the global gradual drift the situation cannot be resolved by only looking at the delays. Therefore, further discussion is provided in the following sub-sections. Experiments were performed

Table 6 Averaged results from the evaluation of change detection with the alternate trees adaptation strategy over 10 experiments

		#FA's	#Adapt.	Change point 250k		Change point 500k		Change point 750k	
				Global delay	PH delay	Global delay	PH delay	Global delay	PH delay
Local abrupt drift	TD	0.0	20.8	12071.0	4006.9	4282.7	768.4	5438.7	548.2
	BU	0.0	16.3	30403.0	1542.1	6582.6	457.3	6863.7	1399.2
Global abrupt drift	TD	0.0	27.4	NA	NA	1995.6	273.3	5692.3	451.7
	BU	0.0	3.8	NA	NA	374.1	374.1	410.5	410.5
Global gradual drift	TD	0.0	43.7	NA	NA	21673.8	3473.5	12389.8	3157.9
	BU	0.0	52.8	NA	NA	18934.6	18934.6	16684.5	13186.1

also on datasets without drift, where the change detection methods did not trigger any false alarms. The delays and the number of adaptations are discussed in Sect. 5.2.2.

5.2.1.1 Resilience to type-I and type-II errors. To examine FIMT-DD's resilience to type-I (false positive) and type-II errors (false negative), we perform a set of experiments over all the datasets (stationary and non-stationary) with the change detection option activated. FIMT-DD does not generate any false positives on the stationary datasets or false negatives on the non-stationary datasets.

5.2.2 Change adaptation

To evaluate the change adaptation mechanism, we compare the change detection methods with two possible adaptation strategies to the "No Detection" option and the incremental algorithms OnlineRD and OnlineRA. Table 7 gives the results over the last 10,000 examples of the stream, using the holdout-window evaluation method. The discussion is structured by the different scenarios of simulated drift.

5.2.2.1 Conclusions on local abrupt drift. The results for the *Local abrupt drift* dataset show that the algorithms OnlineRD and OnlineRA have poor abilities to adapt to local abrupt drift. The error of OnlineRD and OnlineRA is higher than the "No Detection" option. Best results are achieved with the AltTree strategy. A more detailed analysis has shown that the TD method mostly detects changes in the higher nodes of the tree. This can be seen in Fig. 9 where for the third point of change we observe an additional peak due to the pruning of a significant number of nodes, which might be more than necessary. The BU method, on the other hand, detects changes typically at the lowest node whose region is closest to the region affected by the drift. This is a sign of a good localization of the concept drift. On the same figure we observe that although at the end the results are similar, the BU change detection method enables faster adaptation to the new concept which is due to proper localization.

Figure 10 compares the learning curves for the two different change detection methods (BU and TD) and the AltTree strategy obtained by using the sliding window evaluation on the local abrupt concept drift. In the comparison, we also include the situation when no change detection is performed (solid line). The AltTree strategy

Table 7 Performance results over the last 10000 examples of the data stream (never used for learning) averaged over 10 experiments for each type of simulated drift

Dataset	Measures	No det.	OnlineRD	OnlineRA	Top-Down (TD)		Bottom-Up (BU)	
					Prune	AltTree	Prune	AltTree
Local abrupt drift	MSE/SD	4.23 ± 0.11	5.95 ± 1.09	9.12 ± 0.09	4.619 ± 0.093	3.17 ± 0.07	3.88 ± 0.08	3.19 ± 0.07
	RRSE	0.38	0.45	0.56	0.396	0.33	0.37	0.33
	Leaves	2489.00	173.6	109.3	391.6	1840.69	583.20	1879.30
	Nodes	4977.00	–	–	782.2	3680.39	1165.40	3757.60
Global abrupt drift	MSE/SD	3.81 ± 0.07	1.21 ± 0.002	2.71 ± 0.01	4.051 ± 0.073	3.58 ± 0.06	3.67 ± 0.07	3.66 ± 0.07
	RRSE	0.39	0.22	0.33	0.398	0.38	0.38	0.38
	Leaves	2576.70	177.9	101.9	430.9	629.20	572.10	572.10
	Nodes	5152.39	–	–	860.79	1257.40	1143.19	1143.19
Global gradual drift	MSE/SD	9.41 ± 0.19	4.35 ± 3.53	17.88 ± 2.0	3.249 ± 0.058	3.56 ± 0.06	3.38 ± 0.06	3.95 ± 0.07
	RRSE	0.56	0.37	0.76	0.326	0.34	0.33	0.36
	Leaves	2616.70	241.8	115.2	323.6	528.30	500.20	635.10
	Nodes	5232.39	–	–	646.20	1055.59	999.40	1269.19

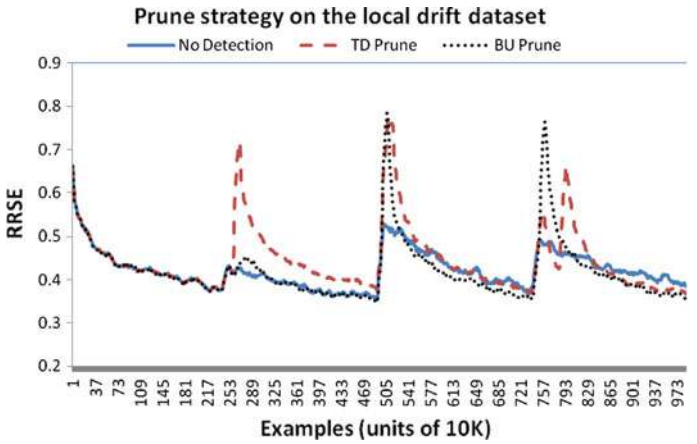

Fig. 9 Learning curves for the Prune adaptation strategy to local abrupt drift in the Fried dataset, using holdout sliding window evaluation over a window of size 10k and a sliding step 1k

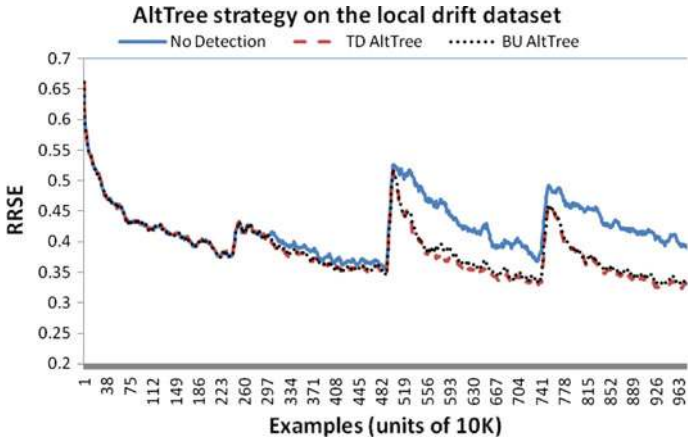

Fig. 10 Learning curves for the AltTree adaptation approach to local abrupt drift in the Fried dataset, using holdout sliding window evaluation over a window of size 10k and a sliding step 1k

prevents reactions to false alarms. Drastic drop of accuracy is avoided. The models have similar errors with both of the change detection methods proposed.

5.2.2.2 Conclusions on global abrupt drift. The analysis on the *Global abrupt drift* dataset yields different results. Best accuracy is achieved by the OnlineRD and OnlineRA algorithms. They are able to completely rebuild the tree at each point of change better than FIMT-DD. Since the drift is global, Prune and AltTree do the same thing, re-growing the whole tree after the change, thus the results are similar. The BU method performs better due to proper detection of drift. The TD approach does not detect the change at the root node, but somewhere in the lower nodes, which can be seen on Fig. 11. This is evident also from the number of adaptations given in Table 6, which

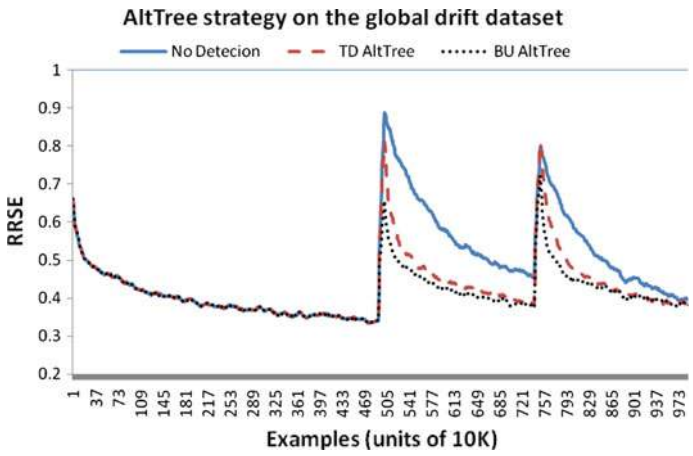

Fig. 11 Learning curves for the AltTree adaptation approach to global abrupt drift in the Fried dataset, using holdout sliding window evaluation over a window of size 10k and a sliding step 1k

is almost seven times higher than for BU. Because of that, neither of the adaptation strategies used with TD will enable proper correction of the structure. If "No detection" is used the size of the tree becomes around five times larger as compared to the other strategies.

5.2.2.3 Conclusions on global gradual drift. For the *Global gradual drift* dataset best results are achieved using the Prune strategy. The results are in favor of the TD method. The trees obtained are smaller and more accurate because of the on-time adaptation. For the same reasons, pruning is a better strategy in this case than growing alternate trees. The BU method performs worse than TD because once a change is detected the adaptation of the model hinders the detection of additional changes although the drift might still be present and increasing.

5.2.3 Comparison when rebuilding

Figure 12 ilustrates the ability of the AltTree adaptation strategy to learn the new phenomenon. As a reference strategy, we rebuild the tree by using examples from the new concept only (option Rebuild, segmented line). The conclusions are as follows: detection of local concept drift enables best adaptation when the tree does not need to be completely rebuilt. The proposed change detection methods are able to localize the drift properly and adapt the tree with minimal loss of accuracy. In Fig. 12 we can see that TD/BU AltTree method provides smallest error at any-time, thus the adaptation can be considered as appropriate.

As a general conclusion, when no detection is performed the tree improves its accuracy due to consecutive splitting, but becomes much larger than necessary. The structure would likely be misleading due to improper hierarchy of splits. For different types of drift different algorithms perform best and there is no generally applicable method.

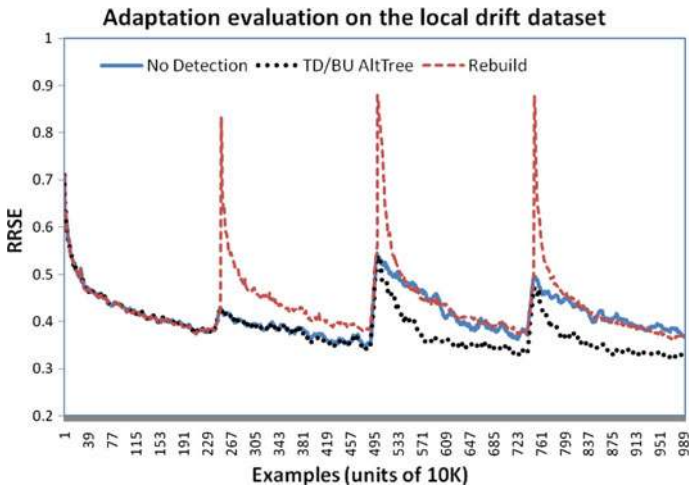

Fig. 12 A comparison of learning curves for the TD/BU AltTree adaptation strategy, on the local abrupt gradual drift in the Fried dataset, obtained by using holdout-window evaluation with window size 10k and sliding step 1k

Table 8 Performance results over the last 100k examples of the data stream on the Airline dataset

Dataset	Measures	No detection	Top-Down (TD)		Bottom-Up (BU)	
			Prune	AltTree	Prune	AltTree
Airline	AE	17.99	14.91	14.80	14.85	14.79
	MSE/SD	862.54 ± 8.56	741.90 ± 8.69	731.73 ± 8.62	733.49 ± 8.59	728.56 ± 8.58
	RE	0.99	0.82	0.81	0.81	0.80
	RRSE	1.02	0.94	0.93	0.93	0.93
	Nodes	86509.00	31.00	143.00	101.00	41.00
	Growing nodes	7558.00	16.00	72.00	51.00	21.00

5.3 Results on real-world data

Figure 13 gives a comparison of the different adaptation strategies (both in terms of accuracy and model size) on the Airline dataset from the DataExpo09 competition. The reference case is "No detection". The AltTree strategy improves greatly the accuracy at any time, while providing significantly smaller models. When no change detection is performed the tree contains nearly 50,000 leaves, of which only 7500 can grow further.

The learning curves confirm the existence of many concept drifts, which are occurring irregularly at very high rates. The changes are of both local and global type, resulting in partial or complete restructuring of the tree. The results in Table 8 obtained for the last 100k examples seen, show that the AltTree strategy achieves continuously smaller error and performs equally well with both of the change detection methods (TD or BU).

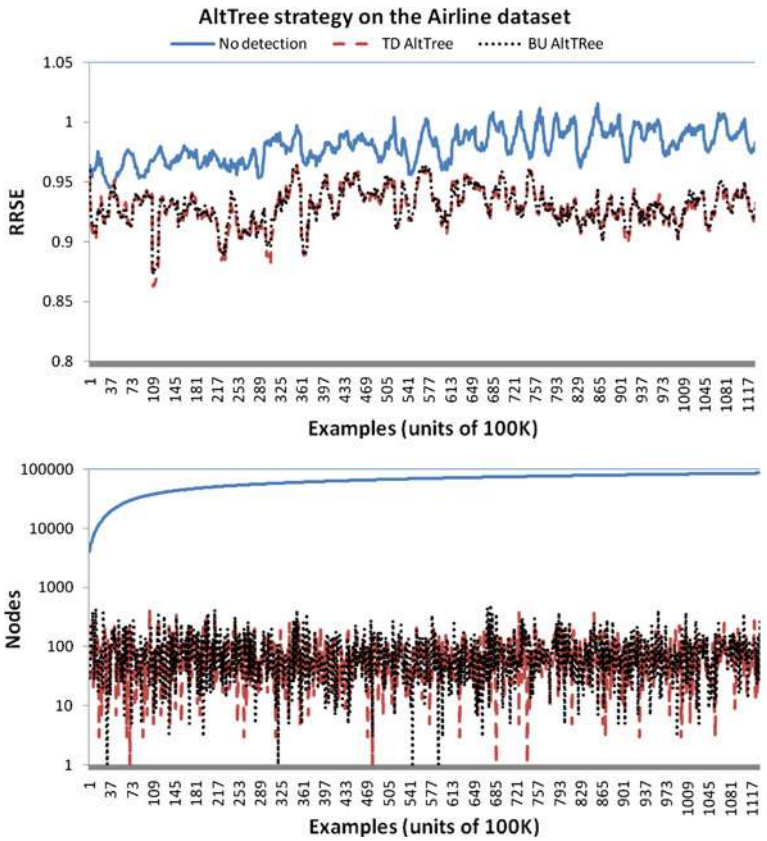

Fig. 13 Learning curves for the Data Expo Airline dataset using sliding prequential-window evaluation with size 1 M and sliding step 100k

The most influential attributes were, as suspected, the *Origin*, the *Destination*, the *Flight time*, the *Day of week*, the *Day of month* and the *Departure* and *Arrival* time. Most of the changes were detected for splits on the attributes *Origin* and *Destination*, moderately on the attribute *Unique Carrier* and only few on the date and the time of flying. This suggests that, the delays were most of the time related to the flying destinations and the carriers.

A change in the frequency of the flights from or to a certain destination, the closing of an airport, an airline entering a crisis and similar events can change the relations between the target attribute (the delay) and the input attributes. Due to the explicit change detection mechanism of FIRT-DD, the regression tree can be used to explain and describe such changes which occurred at irregular periods over the years. A more detailed case study might reveal more interesting facts on the flight delays and their connection to the airlines or the flying destinations, but this is out of the scope of this paper.

6 Conclusions and further work

In this paper, we presented an algorithm for learning model trees from time-changing data streams. To the best of our knowledge, FIMT-DD is the first algorithm for learning model trees from time-changing data streams with explicit drift detection. The algorithm is able to learn very fast (in a very small time per example) and the only memory it requires is for storing sufficient statistics at tree leaves. The model tree is available for use at any time in the course of learning, offering an excellent processing and prediction time per example.

In terms of accuracy, the FIMT-DD is competitive with batch algorithms even for medium sized datasets and has smaller values for the variance component of the error. It effectively maintains an up-to-date model even in the presence of different types of concept drifts. The algorithm enables local change detection and adaptation, avoiding the costs of re-growing the whole tree when only local changes are necessary.

One direction for further research, emerging from the current state of the art in the area of regression analysis on data streams, is the problem of on-line computation of a confidence in individual predictions. There is relevant work in batch learning scenarios (Gammerman and Vovk 2002, 2007) as the most notable, but little attention has been given to incremental algorithms. One example is the work of Rodrigues et al. (2008).

Another promising direction is to extend the proposed algorithms for learning option trees (for regression) from time-changing streaming data. Option trees can be seen as compact and easily interpretable models, which can improve the limited look-ahead and instability of regular regression (decision) trees. Their contribution can also be seen in enhancing the learning process for more complex tasks, such as multi-target regression in a streaming setting.

Acknowledgments We thank Claude Sammut and Duncan Potts for providing the code of their algorithm, and to Raquel Sebastião and Dejan Gjorgjevik who coauthored the paper (Ikonomovska et al. 2009). Elena Ikonomovska is supported by a young researcher grant from the Slovenian Research Agency. Sašo Džeroski is supported by the grants "Advanced Machine Learning Methods for Automated Modeling of Dynamic Systems", "Data Mining for Integrative Analysis in Systems Biology" and "Knowledge Technologies" from the Slovenian Research Agency and the grant PHAGOSYS (Systems Biology of Phagosome Formation and Maturation – Modulation by Intracellular Pathogens) by the European Commission. João Gama is supported by the grant "Knowledge Discovery from Ubiquitous Data Streams" (PTDC/EIA/098355/2008).

Appendix

A Detailed results on stationary streams

See Appendix Tables 9,10,11,12, and 13.

B Details on FIMT_Decay option

To enable a smooth decay of the learning rate, we propose a simple formula (12) for decreasing the value of the learning rate with the number of instances used for training

Table 9 Results of tenfold cross-validation for regression tree learners on the real datasets

Dataset	Size	CART			M5′RT			FIRT		
		RRSE(%)	Leaves	CC	RRSE(%)	Leaves	CC	RRSE(%)	Leaves	CC
Abalone	4,177	74.68	10.6	0.67	70.26	41.8	0.71	77.31	8.5	0.63
Ailerons	13,750	58.67	9.6	0.81	46.31	246.7	0.89	56.73	23.4	0.82
Mv_delve	40,769	27.51	8.0	0.96	4.76	312.1	0.99	8.97	98.8	0.99
Wind	6,574	59.56	7.9	0.80	53.28	107.4	0.85	60.68	11.2	0.79
Cal_housing	20,640	67.67	12.0	0.74	51.79	439.7	0.86	61.46	49.6	0.79
House_8L	22,784	69.14	10.2	0.72	62.45	251.9	0.78	66.99	50.1	0.74
House_16H	22,784	79.38	12.1	0.61	70.80	347.9	0.71	78.28	42.8	0.63
Elevators	16,599	68.09	12.8	0.73	52.14	382.6	0.86	74.86	31.9	0.66
Pol	15,000	36.76	7.0	0.93	22.57	264.9	0.97	29.18	27.1	0.96
Winequality	4,898	86.27	6.5	0.51	82.85	51.9	0.56	86.62	12.0	0.50

Table 10 Results of tenfold cross-validation (RE%) for model tree learners on the real datasets

Dataset	CUBIST	M5′ MT	FIMT_Const	FIMT_Decay	LR	Batch RD	Batch RA	Online RD	Online RA
Abalone	63.20	63.97	74.10	74.18	66.96	63.91	66.41	66.45	66.95
Ailerons	100.00	37.67	102.35	85.53	41.05	37.54	39.32	38.36	40.24
Mv_delve	0.70	0.54	4.83	8.49	38.32	0.06	1.19	0.40	2.05
Wind	44.60	45.98	53.77	60.48	46.90	43.53	44.02	44.48	44.32
Cal_housing	39.10	39.80	48.90	49.24	55.73	40.63	45.27	48.77	48.64
House_8L	50.30	51.39	65.37	69.61	75.10	51.86	55.48	62.31	56.89
House_16H	52.50	56.13	67.91	63.72	78.49	57.07	62.20	71.61	66.58
Elevators	49.90	34.78	82.50	80.35	43.52	34.25	39.01	36.87	39.76
Pol	6.80	6.83	16.20	18.09	71.50	7.06	63.34	43.99	30.38
Winequality	80.40	83.81	86.21	84.26	87.30	82.49	85.97	84.48	85.96

the perceptron:

$$\eta = \frac{\eta_0}{(1 + s\eta_d)} \tag{13}$$

The initial learning rate $\eta_0$and the learning rate decay parameter η_d should be set to appropriately low values (e.g., $\eta_0 = 0.1$ and $\eta_d = 0.005$).

C Details on the simulations of different types of concept drift

The original function for the Fried dataset is: $y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0, 1)$,where $\sigma(0,1)$ is a random number generated from a normal distribution with mean 0 and variance 1.

Table 11 Results of tenfold cross-validation (learning time in seconds) for model tree learners on the real datasets

Dataset	CUBIST	$M5'MT$	FIMT_Const	FIMT_Decay	LR	Batch RD	Batch RA	Online RD	Online RA
Abalone	0.190	2.692	0.041	0.043	0.324	0.589	0.270	0.869	0.325
Ailerons	0.320	14.727	0.661	0.656	1.068	11.865	0.885	148.517	2.355
Mv_delve	1.290	135.125	0.907	0.911	15.449	12.968	11.799	13.974	12.526
Wind	0.450	8.179	0.138	0.135	0.663	1.797	0.657	4.114	0.744
Cal_housing	1.330	48.183	0.356	0.358	3.102	4.998	2.911	5.504	2.930
House_8L	1.50	29.931	0.357	0.352	1.671	4.907	1.921	4.099	1.962
House_16H	2.780	34.515	0.705	0.704	1.757	11.359	2.156	15.667	2.658
Elevators	0.540	11.1	0.367	0.366	0.736	4.905	0.732	14.41	1.203
Pol	1.480	8.24	0.606	0.610	0.771	7.125	0.820	117.468	5.737
Winequality	0.270	2.745	0.067	0.071	0.331	0.841	0.291	2.117	0.398

Table 12 Results of tenfold cross-validation (size) for model tree learners on the real datasets

Dataset	CUBIST	M5′MT	FIMT_Const	FIMT_Decay	Batch RD	Batch RA	Online RD	Online RA
Abalone	11.5	8.4	8.5	8.5	7.1	2.0	2.7	2.0
Ailerons	1.0	5.8	23.4	23.4	13.5	4.7	3.9	2.5
Mv_delve	8.6	9.9	98.8	98.8	166.4	117.2	49.5	54.2
Wind	15.4	7.2	11.2	11.2	10.8	4.1	4.0	4.1
Cal_housing	41.8	214.0	49.6	49.6	120.3	24.8	7.1	9.8
House_8L	21.9	117.2	50.1	50.1	60.7	29.5	8.7	19.3
House_16H	29.6	171.3	42.8	42.8	81.4	33.8	4.8	18.9
Elevators	18.8	39.6	31.9	31.9	37.0	8.2	11.9	6.3
Pol	25.1	159.2	27.1	27.1	46.8	2.0	3.9	8.3
Winequality	33.0	34.5	12.0	12.0	11.8	2.2	3.3	2.4

Local expanding abrupt drift. The first region with drift is defined by the following inequalities: $R1 \equiv x_2 d < 0.3 \,\&\, x_3 < 0.3 \,\&\, x_4 > 0.7 \,\&\, x_5 < 0.3$. The new function for this region is: $y_{1R1} = 10x_1x_2 + 20(x_3 - 0.5) + 10x_4 + 5x_5 + \sigma(0, 1)$. The second region is defined by the following inequalities: $R2 \equiv x_2 > 0.7 \,\&\, x_3 > 0.7 \,\&\, x_4 < 0.3 \,\&\, x_5 > 0.7$. The new function for this region is: $y_{1R2} = 10cos(x_1x_2) + 20(x_3 - 0.5) + e^{x4} + 5x_5^2 + \sigma(0, 1)$. At the second point of change $R1$ and $R2$ are expanded by removing the last inequality ($x_5 < 0.3$ and $x_5 > 0.7$, respectively). At the third point of change, the regions are expanded again by removing the last inequalities ($x_4 > 0.7$ and $x_4 < 0.3$, respectively).

Global reoccurring abrupt drift. The change is performed by misplacing the variables from their original position in the above given function. After the first change the new function is: $y_2 = 10\sin(\pi x_4x_5) + 20(x_2 - 0.5)^2 + 10x_1 + 5x_3 + \sigma(0, 1)$. For the second change, the old concept reoccurs.

Table 13 Results on the artificial datasets: The algorithms were run for 10 randomly generated pairs of train/test sets (1000k/300k), averages over the 10 runs are given here

Algorithm	Dataset	RE	RRSE	Leaves	Time (s)	CC
FIRT	Fried	0.31	0.32	2406.6	42.26	0.95
CUBIST		0.24	NA	58.2	192.47	0.97
FIMT_Const		0.26	0.27	2406.6	46.72	0.96
FIMT_Decay		0.26	0.27	2406.6	46.93	0.96
LR		0.50	0.52	1.0	2509.18	0.85
BatchRD		0.19	0.19	289.9	2341.83	0.98
BatchRA		0.20	0.20	160.8	1946.87	0.98
OnlineRD		0.19	0.20	120.7	2253.09	0.98
OnlineRA		0.20	0.21	143.8	2013.05	0.98
FIRT	Lexp	0.09	0.11	2328.8	15.82	0.99
CUBIST		0.06	NA	27.5	59.19	0.99
FIMT_Const		0.01	0.02	2328.8	19.81	1.00
FIMT_Decay		0.01	0.02	2328.8	19.07	1.00
LR		0.67	0.75	1.0	2785.64	0.67
BatchRD		0.01	0.04	43969.6	6411.54	0.99
BatchRA		0.92	0.88	9.1	2549.39	0.47
OnlineRD		0.004	0.02	13360.1	4039.49	0.99
OnlineRA		0.90	0.85	28.5	2645.06	0.53
FIRT	Losc	0.09	0.14	2621.3	16.16	0.99
CUBIST		0.09	NA	26.8	62.71	0.99
FIMT_Const		0.07	0.14	2621.3	14.81	0.99
FIMT_Decay		0.07	0.14	2621.3	14.80	0.99
LR		0.22	0.26	1.0	2111.03	0.96
BatchRD		0.02	0.07	37599.5	6951.18	0.99
BatchRA		1.00	0.99	1.0	2451.83	0.08
OnlineRD		0.11	0.17	6257.7	3006.89	0.98
OnlineRA		1.00	0.99	1.0	2423.58	0.08

Global slow gradual drift. At the first point of change, examples are generated using the old and the new function: $y_3 = 10\sin(\pi x_4 x_5) + 20(x_2 - 0.5)^2 + 10x_1 + 5x_3 + \sigma(0, 1)$. At the second point of change, the situation is similar and the new function is: $y_4 = 10\sin(\pi x_2 x_5) + 20(x_4 - 0.5)^2 + 10x_3 + 5x_1 + \sigma(0, 1)$.

References

Aggarwal CC (2006) Data streams: models and algorithms. Springer, New York

Basseville M, Nikiforov I (1993) Detection of abrupt changes: theory and applications. Prentice-Hall, Englewood Cliffs, NJ

Blake C, Keogh E, Merz C (1999) UCI repository of machine learning databases. http://archive.ics.uci.edu/ml. Accessed 19 Jan 2010

Breiman L (1998) Arcing classifiers. Ann Stat 26(3):801–824
Breiman L, Friedman JH, Olshen RA, Stone CJ (1998) Classification and regression trees. CRC Press, Boca Raton, FL
Chaudhuri P, Huang M, Loh W, Yao R (1994) Piecewise polynomial regression trees. Stat Sin 4:143–167
Chen Y, Dong G, Han J, Wah BW, Wang J (2002) Multi-dimensional regression analysis of time-series data streams. In: Proc the 28th int conf on very large databases. Morgan Kaufmann, San Francisco, pp 323–334
CUBIST (2009) RuleQuest research. http://www.rulequest.com/cubist-info.html. Accessed 19 Jan 2010
Dasu T, Krishnan S, Lin D, Venkatasubramanian S, Yi K (2009) Change (detection) you can believe in: finding distributional shifts in data streams. In: Proc IDA'09. Springer, Berlin, pp 21–34
Data Expo (2009) ASA sections on statistical computing and statistical graphics. http://stat-computing.org/dataexpo/2009. Accessed 19 Jan 2010
Dawid AP (1984) Statistical theory: the prequential approach. J R Stat Soc A 147:278–292
Dobra A, Gherke J (2002) SECRET: a scalable linear regression tree algorithm. In: Proc 8th ACM SIGKDD int conf on knowledge discovery and data mining. ACM Press, New York, pp 481–487
Domingos P, Hulten G (2000) Mining high speed data streams. In: Proc 6th ACM SIGKDD int conf on knowledge discovery and data mining. ACM Press, New York, pp 71–80
Friedman JH (1991) Multivariate adaptive regression splines. J Ann Stat 19(1):1–67. doi:10.1214/aos/1176347963
Gama J, Castillo G (2004) Learning with local drift detection. In: Proc 2nd int conf on advanced data mining and applications, LNCS, vol 4093. Springer, Berlin, pp 42–55
Gama J, Rocha R, Medas P (2003) Accurate decision trees for mining high-speed data streams. In: Proc 9th ACM SIGKDD int conf on knowledge discovery and data mining. ACM Press, New York, pp 523–528
Gama J, Medas P, Rocha R (2004) Forest trees for on-line data. In: Proc 2004 ACM symposium on applied computing. ACM Press, New York, pp 632–636
Gama J, Sebastiao R, Rodrigues PP (2009) Issues in evaluation of stream learning algorithms. In: Proc 16th ACM SIGKDD int conf on knowledge discovery and data mining. ACM Press, New York, pp 329–338
Gammerman A, Vovk V (2002) Prediction algorithms and confidence measures based on algorithmic randomness theory. J Theor Comput Sci 287:209–217
Gammerman A, Vovk V (2007) Hedging predictions in machine learning. Comput J 50:151–163
Gao J, Fan W, Han J, Yu PS (2007) A general framework for mining concept-drifting data streams with skewed distributions. In: Proc 7th int conf on data mining, SIAM, Philadelphia, PA
Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. J Neural Comput 4:1–58
Gratch J (1996) Sequential inductive learning. In: Proc 13th natl conf on artificial intelligence and 8th innovative applications of artificial intelligence conf, vol 1. AAAI Press, Menlo Park, CA, pp 779–786
Hoeffding W (1963) Probability for sums of bounded random variables. J Am Stat Assoc 58:13–30
Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proc 7th ACM SIGKDD int conf on knowledge discovery and data mining. ACM Press, New York, pp 97–106
Ikonomovska E, Gama J (2008) Learning model trees from data streams. In: Proc 11th int conf on discovery science, LNAI, vol. 5255. Springer, Berlin, pp 52–63
Ikonomovska E, Gama J, Sebastião R, Gjorgjevik D (2009) Regression trees from data streams with drift detection. In: Proc 11th int conf on discovery science, LNAI, vol 5808. Springer, Berlin, pp 121–135
Jin R, Agrawal G (2003) Efficient decision tree construction on streaming data. In: Proc 9th ACM SIGKDD int conf on knowledge discovery and data mining. ACM Press, New York, pp 571–576
Karalic A (1992) Employing linear regression in regression tree leaves. In: Proc 10th European conf on artificial intelligence. Wiley, New York, pp 440–441
Kifer D, Ben-David S, Gehrke J (2004) Detecting change in data streams. In: Proc 30th int conf on very large data bases. Morgan Kaufmann, San Francisco, pp 180–191
Klinkenberg R, Joachims T (2000) Detecting concept drift with support vector machines. In: Proc 17th int conf on machine learning. Morgan Kaufmann, San Francisco 487–494
Klinkenberg R, Renz I (1998) Adaptive information filtering: learning in the presence of concept drifts. In: Proc AAAI98/ICML-98 wshp on learning for text categorization. AAAI Press, Menlo Park, pp 33–40

Loh W (2002) Regression trees with unbiased variable selection and interaction detection (2002). Stat Sin 12:361–386

Malerba D, Appice A, Ceci M, Monopoli M (2002) Trading-off local versus global effects of regression nodes in model trees. In: Proc 13th int symposium on foundations of intelligent systems, LNCS, vol 2366. Springer, Berlin, pp 393–402

Mouss H, Mouss D, Mouss N, Sefouhi L (2004) Test of Page–Hinckley, an approach for fault detection in an agro-alimentary production system. In: Proc 5th Asian control conference, vol 2. IEEE Computer Society, Los Alamitos, CA, pp 815–818

Musick R, Catlett J, Russell S (1993) Decision theoretic sub-sampling for induction on large databases. In: Proc 10th int conf on machine learning. Morgan Kaufmann, San Francisco, pp 212–219

Pang KP, Ting KM (2005) Improving the centered CUSUMS statistic for structural break detection in time series. In: Proc 17th Australian joint conf on artificial intelligence, LNCS, vol 3339. Springer, Berlin, pp 402–413

Pfahringer B, Holmes G, Kirkby R (2008) Handling numeric attributes in Hoeffding trees. In: Proc 12th Pacific-Asian conf on knowledge discovery and data mining, LNCS, vol 5012. Springer, Berlin, pp 296–307

Potts D, Sammut C (2005) Incremental learning of linear model trees. J Mach Learn 61:5–48. doi:10.1007/s10994-005-1121-8

Quinlan JR (1992) Learning with continuous classes. In: Proc 5th Australian joint conf on artificial intelligence. World Scientific, Singapore, pp 343–348

Rajaraman K, Tan (2001) A topic detection, tracking, and trend analysis using self-organizing neural networks. In: Proc 5th Pacific-Asian conf on knowledge discovery and data mining, LNCS, vol 2035. Springer, Berlin, pp 102–107

Rodrigues PP, Gama J, Bosnic Z (2008) Online reliability estimates for individual predictions in data streams. In: Proc IEEE int conf on data mining workshops. IEEE Computer Society, Los Alamitos, CA, pp 36–45

Sebastiao R, Rodrigues PP, Gama J (2009) Change detection in climate data over the Iberian peninsula. In: Proc IEEE int conf on data mining workshops. IEEE Computer Society, Los Alamitos, CA, pp 248–253

Siciliano R, Mola F (1994) Modeling for recursive partitioning and variable selection. In: Proc int conf on computational statistics. Physica Verlag, Heidelberg, pp 172–177

Song X, Wu M, Jermaine C, Ranka S (2007) Statistical change detection for multidimensional data. In: Proc 13th ACM SIGKDD conf on knowledge discovery and data mining, pp 667–676

Subramaniam S, Palpanas T, Papadopulous D, Kalogeraki V, Ginopulos D (2006) Online outlier detection in sensor data using non-parametric methods. In: Proc 32nd int conf on very large databases, ACM, New York, pp 187–198

Torgo L (1997) Functional models for regression tree leaves. In: Proc 14th int conf on machine learning. Morgan Kaufmann, San Francisco, pp 385-393

VFML (2003) A toolkit for mining high-speed time-changing data streams. http://www.cs.washington.edu/dm/vfml. Accessed 19 Jan 2010

Vogel DS, Asparouhov O, Scheffer T (2007) Scalable look-ahead linear regression trees. In: Berkhin P, Caruana R, Wu X (eds) Proc 13th ACM SIGKDD int conf on knowledge discovery and data mining, KDD. ACMK, San Jose, CA, pp 757–764

WEKA 3 (2005) Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka. Accessed 19 Jan 2010

Widmer G, Kubat M (1996) Learning in the presence of concept drifts and hidden contexts. J Mach Learn 23:69–101. doi:10.1007/BF00116900