Neurocomputing 150 (2015) 458–470

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Online tree-based ensembles and option trees for regression on evolving data streams

Elena Ikonomovska [a,*], João Gama [b], Sašo Džeroski [c]

[a] Turn Inc., 835 Main St, Redwood City, CA, United States
[b] LIAAD-INESC, Rua Dr. Roberto Frias, 378 4200-378 Porto, Portugal
[c] Department of Knowledge Technologies, Jožef Stefan Institute, Jamova cesta 39, SI-1000 Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 1 January 2014
Received in revised form
25 April 2014
Accepted 26 April 2014
Available online 4 November 2014

Keywords:
Online learning
Data streams
Online regression trees
Option trees
Online ensembles
Online random forests

ABSTRACT

The emergence of ubiquitous sources of streaming data has given rise to the popularity of algorithms for online machine learning. In that context, Hoeffding trees represent the state-of-the-art algorithms for online classification. Their popularity stems in large part from their ability to process large quantities of data with a speed that goes beyond the processing power of any other streaming or batch learning algorithm. As a consequence, Hoeffding trees have often been used as base models of many ensemble learning algorithms for online classification. However, despite the existence of many algorithms for online classification, ensemble learning algorithms for online regression do not exist. In particular, the field of online any-time regression analysis seems to have experienced a serious lack of attention. In this paper, we address this issue through a study and an empirical evaluation of a set of online algorithms for regression, which includes the baseline Hoeffding-based regression trees, online option trees, and an online least mean squares filter. We also design, implement and evaluate two novel ensemble learning methods for online regression: online bagging with Hoeffding-based model trees, and an online RandomForest method in which we have used a randomized version of the online model tree learning algorithm as a basic building block. Within the study presented in this paper, we evaluate the proposed algorithms along several dimensions: predictive accuracy and quality of models, time and memory requirements, bias–variance and bias–variance–covariance decomposition of the error, and responsiveness to concept drift.

1. Introduction

Any-time online regression analysis is a research topic which tackles problems such as predicting traffic jams, power consumption, entertainment trends, and even flu outbreaks. These types of problems require online any-time predictive modeling and responsiveness to changes in near real-time. Ensemble learning methods are considered to be one of the most accurate and robust machine learning approaches for solving classification and regression tasks. They have been successfully used in many real-world problems, including the well known Netflix Prize[1] competition, where it was shown that best results can be achieved by combining multiple models and ensembles of models, each of them specializing in different aspects of the problem.

Common knowledge on ensemble learning methods suggests that the base models should be as accurate as possible and should have as diverse as possible distributions of errors. Although this seems as a simple requirement, it is not easily achieved [1]. From the various types of models, decision trees are particularly well suited for this task because they enable a simple and effective way to create an ensemble of diverse yet accurate hypotheses. At the same time, decision and regression trees are easy to interpret and can handle both numeric and nominal attributes. The most typical approach in learning ensembles of trees is to apply basic sampling techniques such as bagging and boosting, or to randomize the learning process.

Hoeffding-based algorithms for learning decision or regression trees [2–6] are one of the most popular methods for learning trees from data streams. Due to their ability to make theoretically supported split selection and stopping decisions in a single pass over the training data, without having to store any of the data instances, they are able to process large quantities of data at a speed that goes beyond the processing abilities of batch learning algorithms. As such, they are an invaluable tool for real-time prediction and analysis of streaming data. A recent new addition to the literature of mining data streams is the paper by Rutkowski et al. [7] in which the McDiarmid's bound is proposed as a substitution and a more suitable choice than the Hoeffding bound. Their proposal is supported by a nice theoretical analysis, however the experimental results presented in the paper are not conclusive.

* Corresponding author.
[1] http://www.netflixprize.com/

http://dx.doi.org/10.1016/j.neucom.2014.04.076
0925-2312/

While online ensembles of decision trees have been extensively studied, algorithms for learning online ensembles of regression trees have not been proposed or empirically evaluated yet. Consequently, in this work, we focus on online tree-based ensembles for any-time regression. Our base models are online Hoeffding-based regression trees, produced by streaming any-time algorithms such as the FIMT-DD algorithm [6] and a randomized version of FIMT-DD, termed R-FIMT-DD, presented in this paper for the first time. We consider two different ensemble learning methods, online bagging of Hoeffding-based trees for regression (OBag) and online random forest for any-time regression (ORF). We analyze and evaluate them in terms of their ability to achieve diversity among the constituent models, improve accuracy, as well as, in terms of their computational complexity and demand for resources. We also provide an extensive empirical comparison of our ensemble methods OBag and ORF to online option trees for regression (ORTO) [8] and show that option trees are an appealing alternative to ensembles of trees in terms of both accuracy and resource allocation.

In sum, we make the following contributions to the area of online learning for any-time regression:

1. We give a systematic overview of existing methods for learning tree-based ensembles for online classification.
2. We implement and empirically evaluate two new methods for learning tree-based ensembles for online regression: online bagging of FIMT-DD trees (OBag) and an online RandomForest method for regression, based on the randomized algorithm R-FIMT-DD (ORF). To the best of our knowledge, there is no other work that studies methods for learning tree-based ensembles for online regression on data streams.
3. We further extend the empirical evaluation by performing a comparison with online option trees for regression and show that option trees achieve better accuracy using fewer resources and less computational power.
4. By using a theoretical analysis of the sources of error we study the ways these different techniques improve the accuracy over the base models and try to correlate the diversity of the ensemble to its generalization power.

The remainder of the paper is organized as follows. In Section 2 we begin with an overview of ensemble methods for online classification. To the best of our knowledge ensembles methods for online regression have not been published yet. Section 3 introduces two novel ensemble methods for online regression on data streams. Section 4 presents an extended version of the work on online option trees for regression [8]. These algorithms are empirically evaluated using the evaluation methodology described in Section 5. In Section 6, we give the results from the empirical evaluation of OBag, ORF, ORTO and Hoeffding-based regression trees in different learning scenarios. Finally, Section 6.4 presents the differences between the two ensemble learning methods OBag and ORF in terms of the source of error and the amount of diversity they were able to introduce in their base models.

2. Online ensembles of decision trees

Although tree-based ensemble methods for online regression have not been studied nor analyzed yet, there is a plethora of tree-based ensemble learning methods for online classification [9–17]. In this section, we present a survey of the existing ensemble learning methods for online classification. In the following subsections, we will discuss *online bagging* and *online boosting* as main representatives of the category of methods which introduce diversity by diversification of the training data. These methods represent the basis for implementing online versions of bagging and boosting for learning various types of ensembles. Separate subsections discuss online RandomForest and stacked generalization with restricted Hoeffding trees.

2.1. Online bagging

The simplest method to introduce diversity among the base models that constitute an ensemble is to generate different samples from the training dataset, i.e., modify the training dataset each time a base model is learned. The batch *bagging* method generates a number of bootstrapped training sets from the original dataset. A bootstrap sample can be obtained by random sampling with replacement according to a Binomial probability distribution. In the next step, a separate model is induced on each training dataset. In the testing phase, the predictions from the base models are aggregated using majority voting. Online versions of sampling-based methods for learning ensembles have been studied and proposed for the first time in the work of [18].

2.1.1. Bootstrap sampling and online bagging

Before looking into the details of online bagging, let us first discuss the details of bootstrap sampling. The main effect of the bootstrap sampling procedure is to simulate repeated observations from an unknown population using the available sample as a basis. A training sample generated with batch bootstrap sampling contains K copies of each of the original training examples, where K is a random variable distributed according to the Binomial distribution. In order to transform the batch procedure into an online one, the number of occurrences of each training example for each training dataset has to be estimated at its first occurrence, without the possibility to examine the rest of the training dataset. Oza and Russel in [18] have observed that when the size of the training dataset tends to infinity; i.e., $N \to \infty$, the Binomial distribution tends to be more and more similar to the Poisson distribution Poisson(1) which is defined as

$$Poisson(\lambda) \sim \frac{e^{-1}}{k!} \quad \text{for } \lambda = 1,$$

where k is the number of occurrences of the discrete stochastic variable K. The formula gives us the means to compute the value of K for a given randomly chosen probability for each training example. At the same time, it applies perfectly to the online learning setup in which the size of the training set is unbounded and tends to infinity. [18] have proven that if the batch and the online bagging algorithms use the same training set that grows to infinity, the distributions over the bootstrapped training sets will converge to the same distribution.

The algorithm receives as input a set of initialized base models $H_M = \{h_1, \ldots, h_M\}$, i.e., one-leaf decision trees. For every training example e, received from the data stream, and for each base model h_i, the number of occurrences K of the example e in the online training set used to update h_i is set to $k = Poisson(1)$. In the next step, the example is given K times to the procedure which refines the corresponding model h_i. This procedure is repeated as long as there are training examples available to update the set of models.

2.2. Online bagging for concept drift management

The *online bagging* meta-algorithm has been used by several authors and mainly for the task of online classification. [19] have proposed two interesting algorithms for learning ensembles of Hoeffding trees for online classification: ADWIN Bagging and Adaptive-Size Hoeffding Tree (ASHT) Bagging. Both ADWIN Bagging and Adaptive-Size Hoeffding Tree (ASHT) Bagging have been developed to address the problem of learning under non-

stationary distributions and perform some form of blind concept drift management.

ADWIN Bagging employs the *online bagging* procedure in order to learn an ensemble of incrementally built Hoeffding trees. The ensemble is equipped with an explicit drift management mechanism that detects the worst performing classifier in the collection. A member whose performance was observed to drop significantly is immediately replaced with a new classifier that would be incrementally built using the new training instances. The name of the algorithm ADWIN proposed by [20] comes from the change detection method used to monitor the evolution of the average loss of each base classifier.

ASHT Bagging, on the other hand builds a set of Hoeffding trees constrained in size by using the same *online bagging* procedure. Each tree is periodically reset (or regrown) using the most recent sample of training examples. The reset rate depends on the size of the tree. Trees limited to size s will be reset twice as often as the trees with a size limited to $2s$. The authors have studied different types of reset operations, varying between replacing over-sized trees with new ones or pruning from the root.

2.3. Online boosting

Like online bagging, the *online boosting* algorithm of [18] aims to transform the batch boosting technique into an online one. It has been designed to approximate the batch learning algorithm AdaBoost by [21]. The *online boosting* procedure is essentially the same as *online bagging*. The main difference is in the dynamic computation of the Poisson parameter λ, whose value in the case of online bagging is kept constant ($\lambda = 1$). In particular, the authors propose to adjust the value of λ according to the performance of the classifier that was updated last. When a base model incorrectly classifies a training example the value of the Poisson parameter λ associated with that example is increased for the next base model. In the case of a correct classification the value of λ will be decreased. Misclassified examples are given half the total weight in the next stage; the correctly classified examples are given the remaining half of the weight.

However, batch boosting is a sequential process that runs multiple times over the same training set, boosting the accuracy of the classifiers in each iteration based on the estimates from the previous run. The problem with online boosting is that the estimate used to determine the probability of presenting an example to the next model is based on the performance given the examples seen so far. This is completely different compared to the batch version, where the estimate is obtained on a fully trained base model. The online boosting algorithm was found to significantly underperform as compared to batch boosting for small training sets, especially for the initial training sequence, when the estimates were fairly biased. However, priming online boosting with an initial batch training was shown to perform comparably to batch boosting over the whole course of learning. In the study of [19], online boosting was also found to yield inferior accuracy compared to online bagging.

A slightly improved version of online boosting, called Online Coordinate Boosting [22], stems from an approximation to the AdaBoost's loss minimization, given a training set that grows one example at a time. While in Oza and Russell's online boosting algorithm the update rule is broken down to two cases, one for the correctly and one for the incorrectly classified examples, in Online Coordinate Boosting a single update rule is used: This rule takes the average of the old weight and the newly updated weight that one would get by using AdaBoost's exponential re-weighting. In an experimental evaluation, it was shown that by greedily minimizing the approximation error of each base classifier, Online Coordinate Boosting is able to approximate the batch AdaBoost performance better than Oza and Russell's online boosting algorithm.

2.4. Online random forest

With the availability of an online bagging method and various online decision tree learning algorithms, a host of methods for learning online RandomForest ensembles have emerged [23–25,17]. The online random forest method of [25] uses online bagging to create an ensemble of *Hoeffding Naïve Bayes* trees. A *Hoeffding Naïve Bayes* tree is a leveraged Hoeffding tree that has Naïve Bayes classifiers in the leaf nodes [4]. When performing a prediction, the leaf will return the Naïve Bayes prediction whenever it has been estimated to be more accurate than the majority vote prediction. In the online random forest method, the trees are grown using a modified version of the algorithm in which splits are allowed only on $\sqrt{m}$ attributes, randomly chosen at each node (here m is the number of attributes). Their experimental findings suggested that the online random forest performed faster, but was evaluated to have 5% lower accuracy than the bagged Hoeffding Naïve Bayes trees.

Another interesting idea in the line of randomizing the learning process is the work of Bifet et al. [25] where they propose using random error-correcting output codes. The way that error-correcting output codes work is as follows. Every class is assigned a random binary string of length n, where each position in the string is learned by a single binary classifier. At prediction time, the testing instance is assigned the class whose code is closest to the combined prediction of the n classifiers. Since every classifier learns and predicts a different function the diversity of the ensemble is expected to increase.

2.5. Stacked generalization with restricted Hoeffding trees

Stacked generalization or stacking [26] learns to combine multiple models by applying a higher level meta-learner in a space whose inputs are the predictions, confidences or distributions of the base models. There are many different ways to implement stacked generalization. The most common implementation combines the predictions of multiple models learned by different methods.

Like bagging, stacking is ideal for parallel computation. The construction of the base models can proceed independently, requiring no communication among the learning processes. While bagging usually requires a considerable number of base models, because it relies on the diversity of the training data used to induce the base models, stacking can work with only two or three base models [27]. Therefore, it seems natural to think of stacking in the context of online learning, if one can determine a suitable online meta-learning algorithm. In this context, [28] proposed a method for learning an ensemble of restricted Hoeffding trees based on stacking. Each decision tree is learned in parallel from a distinct subset of attributes. Their predictions are further combined using perceptrons whose weights can be updated using an online update rule. The authors propose to enumerate all attribute-subsets whose number is predetermined k, and learn a Hoeffding tree from each subset.

The computational complexity of this method depends on the value of k, which defines the size of the ensemble. For a number of m attributes there are $\binom{m}{k}$ possible subsets of attributes. The authors propose $k=2$ for datasets with relatively large number of attributes. Acceptable runtime can be also obtained with $k=4$ for $m=10$. Compared to online bagging of Hoeffding trees, this method was shown to have excellent classification accuracy on most of the datasets used in the experimental evaluation. It was also shown that stacking can improve upon the accuracy of online bagging without increasing the number of trees in the ensemble.

3. Ensembles of Hoeffding-based trees for online regression

In this section, we present two novel ensemble learning methods for online regression. Both of these algorithms use *online bagging* to randomize the training data.

3.1. Online bagging of Hoeffding-based trees for regression

The first ensemble learning method, named *OBag* learns an ensemble of FIMT-DD model trees through the use of the *online bagging* sampling algorithm. FIMT-DD [6] is an online any-time algorithm for learning regression and model trees from evolving data streams. The algorithm is able to incrementally induce model trees by processing each training example only once in the order of its arrival. The incremental process of learning a tree constitutes of making split selection decisions, each on a different sub-sample of the training data. The size of the sample can be determined approximately by applying the Hoeffding bound on the probability to erroneously render a split to be better than its competitor. FIMT-DD is parameterized with a level of confidence δ, used in the formula of the Hoeffding bound ($\epsilon = \sqrt{\ln(1/\delta)/2 \times Seen}$) where *Seen* is the number of processed examples, and a chunk size n_{min}, used for a periodic evaluation of splits.

Like the existing Hoeffding-based algorithms for classification, FIMT-DD starts with an empty leaf and reads examples in the order of their arrival. The algorithm does not require any training examples to be stored. The only memory needed is for storing the sufficient statistics in the leaves and the tree structure itself. FIMT-DD monitors the accuracy of every sub-tree in the induced model tree and alarms when a significant increase in the error is detected by an online change detection test. As a result, the model tree is adapted automatically in real-time according to the most recent changes in the data stream. The details of the drift detection and adaptation methods used in FIMT-DD are presented by [6].

While growing the tree, the algorithm simultaneously computes linear models in the leaves. The linear models are computed using adaptive least mean squares (LMS) filters, where the coefficients of the linear model are equal to the weights of the corresponding LMS filter. This process is of a linear complexity. To allow for mixed types of inputs, the categorical attributes need to be previously transformed into a set of binary attributes. This transformation is necessary only for the linear models.

By plugging these adaptive model trees into the online bagging procedure, we get a dynamic ensemble of model trees which is learned incrementally from a data stream and adapted appropriately when changes in data occur. The pseudo-code of the resulting *OBag* algorithm is given in Algorithm 1. The algorithm starts by initializing a set of model trees each with a LMS adaptive filter, whose weights are set randomly in the range [0,1]. The ensemble is built by online bagging of T_{max} instances of the FIMT-DD algorithm, each initialized with a corresponding one-leaf model tree, as described in Section 2.1. Each model tree provides a single prediction per testing example, which at the end of each training trial is combined with the remaining predictions through non-weighted averaging.

Algorithm 1. The algorithm OBag.

Input: T_{max} – number of base models, δ – confidence parameter, n_{min} – chunk size.
Output: p – prediction from the current OBag ensemble.
for $i = 1 \rightarrow T_{max}$ **do**
  $M_i = InitializeTree()$
end for
for ∞ **do**
  $e \leftarrow ReadNext()$
  for $i = 1 \rightarrow T_{max}$ **do**
    $k \leftarrow Pois(1)$
    for $j = 1 \rightarrow k$ **do**
      $p_i \leftarrow FIMT{-}DD(M_i, e, \delta, n_{min})$
    end for
  end for
  $p \leftarrow AVG(p_1, p_2, \ldots, p_{T_{max}})$ **return** p
end for

3.2. Online random forest, for any-time regression

In this section we present ORF, an Online Random Forest algorithm, which is an online variant of the Random Forest [29] algorithm for the task of regression. As our base learner we use a randomized version of the FIMT-DD algorithm, called R-FIMT-DD. The pseudo-code of R-FIMT-DD is given in Algorithm 2.

Algorithm 2. The algorithm R-FIMT-DD: randomized FIMT-DD.

Input: *RootNode*, e – training instance, δ – confidence parameter, n_{min} – chunk size, (α, λ) – parameters used in the Page–Hinkley test.
Output: p – prediction from the current hypothesis.
$Leaf \leftarrow Traverse(RootNode, e)$
$Counter \leftarrow SeenAt(Leaf)$
if $Counter = 0$ **then**
  $q \leftarrow SelectAttributesAtRandom(Leaf)$
  $InitializePerceptron(Leaf)$
end if
$Counter \leftarrow Counter + 1$
$p \leftarrow GetPrediction(Leaf, e)$
$UpdateStatistics(Leaf)$
$UpdateLMS(Leaf)$
if $Counter \bmod n_{min} = 0$ **then**
  $q \leftarrow SelectAttributesAtRandom(Leaf)$
  for $i = 1 \rightarrow q$ **do**
    $S_i = FindBestSplitPerAttribute(i)$
    $S_a \leftarrow Best(S_1, \ldots, S_q)$
    $S_b \leftarrow SecondBest(S_1, \ldots, S_q)$
    if $S_b/S_a < 1 - \sqrt{\frac{\ln(1/\delta)}{2 \times Counter}}$ **then**
      $MakeSplit(Leaf, S_a)$
    end if
  end for
end if
$t \leftarrow GetTargetValue(e)$
$\Delta \leftarrow ABS(t - p)$
$BackPropagate(\Delta)$
return p

In the randomized FIMT-DD, whenever a new leaf is created, q attributes are chosen at random from the set of d possible attributes. For each of the chosen attributes, the required statistics $\{N, \sum y, \sum y^2\}$ (corresponding to the count of instances processed, the sum of target attribute values and the sum of squared target attribute values per split-side (for all the splits)) will be initialized to zero and incrementally maintained until the moment of split construction. This procedure is based on a special type of a data structure called extended binary search tree (E-BST), described in detail by [6].

Each split evaluation happens after a minimum of n_{min} examples have been observed in a leaf node. We use the same probabilistic estimates based on the Hoeffding bound, with a user-defined confidence level δ, as described by [6]. The main

difference is in the fact that only the previously chosen q attributes will be considered as possible candidates in the split evaluation performed with the *FindBestSplit*(i) procedure, for $i=1,\ldots,q$. In sum, each terminal node will have an independently and randomly chosen set of attributes which define the set of possible directions to search the hypothesis space. Since statistics will be maintained only for these q attributes, where $q=\sqrt{d}$, the memory allocation is significantly reduced. In addition to this, each leaf will contain an adaptive least mean squares filter (LMS) with a number of inputs equal to the number of numerical attributes plus the number of binary attributes resulting from the transformation of the categorical ones into binary.

Each node of the randomized model tree has an online change detection mechanism based on the Page–Hinkley test. The change detection test continuously monitors the accumulated absolute error given every example that goes through the node. This is performed through the *BackPropagate* procedure that takes as input the absolute error $\Delta=ABS(t-p)$ for a given example e, whose true target value is t and predicted value p (the latter obtained with the *GetPrediction*(*Leaf*) procedure). *GetPrediction*(*Leaf*) returns either the estimated mean or the output computed by using the least mean squares filter in the corresponding leaf node. The choice is made given the accuracy of the predictions from the least mean squares filter estimated with a weighted prequential squared error. As each tree is equipped with change detection units, the base models can be kept independently and automatically up-to-date with possible changes in the target function.

The pseudo-code of the final algorithm ORF is almost identical to the OBag algorithm, except for the call to R-FIMT-DD as a base learner (instead of FIMT-DD). Each call of R-FIMT-DD returns a prediction for e. However, only the last one is considered in the aggregation procedure. The final prediction is computed by averaging the individual predictions $\{p_1,p_2,\ldots,p_{T_{max}}\}$ obtained from the constituent model trees. Naturally, each of these processes can be run simultaneously in parallel. Thus, the whole ensemble would represent an autonomous self-sustaining dynamic learning unit with a decentralized concept drift management.

Despite the fact that the ensemble has lower interpretability as compared to a single model, the adaptation to concept drift or changes in the target function is still performed in an informed way. The detected changes in the tree structure can be represented with a set of rules which can be matched for similarities or dissimilarities. We believe that, given the fact that each model can address a different aspect of the data, an ensemble of models has the ability to bring more value and return more information on the nature of changes than a single model tree.

4. Options for speeding-up Hoeffding-based regression trees

Hoeffding-based tree learning algorithms are very sensitive to ambiguity or highly correlated attributes. While some candidate splits exhibit strong advantage over the rest, there are situations in which there is no clear difference in their estimated merits. The Hoeffding bound is ignorant to the value of the mean or the variance of the random variables and does not take into account situations of this type. Therefore, when the value of the estimated mean difference is zero, the algorithm will not be able to make an informed decision even if an infinite number of examples are observed.

This problem has been addressed using a tie-breaking mechanism. The tie-breaking mechanism determines an amount of data which has to be processed before any decision can be reached. As a result, a constant delay will be associated to every ambiguous situation.

Ikonomovska et al. [8] propose an algorithm for learning online option trees (ORTO) that arrises as a natural solution to this problem. The main idea is to use option nodes as an ambiguity resolving mechanism. Option trees replace the standard split selection decision with a multi-split. If we think of tree induction as search through the hypothesis space H, an option node is best understood as a branching point in the search trajectory. Each optional split represents a different direction that can be followed in the exploration of the search space. When multiple splits are allowed there is no need to wait for more statistical evidence to choose between the highest ranked splits. As a result a split selection decision will be reached sooner on the cost of a higher estimation error. The hill-climbing search strategy is therefore replaced with a more robust one, which will enable us to revise the selection decisions when more evidence has been observed.

Being able to explore a larger sub-space of hypotheses H, an option tree is expected to be more stable (in terms of the influence of the choice of training data) and more robust compared to an ordinary decision tree. In the same time an option tree represents a compressed and connected set of multiple regression trees and as such it is very interesting for comparison with ensemble methods.

Similarly to FIMT-DD, ORTO starts with an empty leaf and reads examples from the stream in the order of their arrival. Each example is traversed to a leaf where the necessary statistics (such as N, $\sum y$, $\sum y^2$) are maintained per side for each splitting point. After a minimum of n_{min} examples have been observed in the leafs of the tree, the algorithm examines the splitting criterion. If an ambiguous situation is encountered the algorithm will proceed with creating an option node. Otherwise, a standard splitting node will be introduced and the procedure will be performed recursively for the leaf nodes in which a modulo of n_{min} examples have been observed. The details are given by [8].

The ORTO algorithm employs two different strategies to aggregate multiple predictions per testing example: *non-weighted averaging* and following the *best path*. The *best path* strategy selects a single "best" prediction obtained by determining the best path to a leaf at a given point in time. Details can be found in [8]. To make a distinction between these two different prediction rules within ORTO, we will use the following acronyms: ORTO-A (averaging) and ORTO-BT (following the best path that represents the best tree).

5. Experimental evaluation

5.1. Algorithms and parameter settings

The following section describes the experimental evaluation designed to evaluate the two new ensemble learning methods and compare them to the base-line algorithms discussed previously. Table 1 gives the list of the algorithms with their acronyms and a short description.

For every experiment we set the common parameters to the following values: The confidence parameter is set to $\delta=10^{-6}$, $n_{min}=200$ (the minimal number of examples before a split evaluation is done) and the decay factor *df* for the option trees is set to 0.9 [30]. For the Page-Hinkley change detection test we set the parameters $\alpha=0.005$ and $\lambda=50$. The adaptive LMS filters use a

Table 1
The tree-based algorithms for online regression used in the empirical evaluation.

Algorithm	Description
LMS	Least mean squares adaptive filter
FIMT-DD, [6]	Fast and incremental model tree with drift detection
ORTO-A, [8]	Online option tree with averaging
ORTO-BT, [8]	Online option tree with best model's prediction
OBag	Online Bagging of FIMT-DD trees
ORF	Online Random Forest of FIMT-DD trees

decaying learning rate initialized to 0.1. These are the default values.

The new parameters introduced here are set as follows. The number of randomly selected attributes at each node for the ORF algorithm is the square root of the number of descriptive attributes for the given dataset. A separate set of experiments is performed where some extra attributes are allowed in addition to the square root of the total number of attributes. The maximal number of trees T_{max} is varied in the range between 10 and 30. For OBag and ORF this parameter equals to the number of trees in the ensemble, while for ORTO-A this it the number of different trees represented by the option tree. The maximal depth of each tree l_{max} is limited for all of the algorithms and varied from level 5 to level 7 by increasing the level of maximal depth by one (the root is at level 0).

5.2. Evaluation methodology

To obtain online estimates we opted for a procedure which can be easily parallelized and decouples the testing data set from the training data set. The evaluation procedure consists of ten rounds of sampling with replacement by using the method of [18]. Each round of sampling provides a different training dataset, but the testing datasets remain the same which is important for a fair comparison. If performed in parallel, the system is provided with an online evaluation methodology applicable to the streaming scenarios.

Periodical evaluation is done at pre-defined time intervals using a separate holdout set of unseen testing examples. In particular, the absolute error, the mean squared error and the root relative mean squared error are computed periodically at predefined time steps using a holdout type of sliding window estimation. The holdout evaluation can be implemented by applying a FIFO buffer over the data stream, which will continuously hold out a set of training examples. At the end of every testing phase the training examples will be used to update the model. This approach facilitates comparison with streaming algorithms designed to run over batches of examples.

5.3. Analytical tools

A very useful analytical tool for understanding the source of error is a bias–variance decomposition of the error. Given a training set $\{(\mathbf{x_1},z_1), (\mathbf{x_2},z_2), \ldots, (\mathbf{x}_N,z_N)\}$ the task of regression is to construct an estimator that approximates an unknown target function $z=g(\mathbf{x})$. The learning algorithms greedily (in our case) construct a non-parameterized estimator f (a regression tree) that minimizes the expected mean squared error:

$$e(f) = \int (f(\mathbf{x})-z)^2 p(\mathbf{x}) d(\mathbf{x}). \tag{1}$$

The bias–variance decomposition of the squared error is

$$E\{(f(\mathbf{x})-z)^2\} \tag{2}$$

$$= (E\{f(\mathbf{x})\}-z)^2 + E\{(f(\mathbf{x})-E\{f(\mathbf{x})\})^2\} = bias(f(\mathbf{x}))^2 + variance(f(\mathbf{x})). \tag{3}$$

For best performance, these two components of the error have to be balanced against each other. The *bias* of an estimator reflects its intrinsic ability to model the given unknown function $g(\mathbf{x})$ and does not depend on the choice of the training set. On the other hand, the *variance* component measures the variability of the models produced given different training sets, and does not depend on the true target attribute values.

Ensembles are known to have best performance when ensemble members show disagreement on certain data points. Such disagreement is usually obtained by supporting diversity among ensemble members. In the regression framework, the base models' predictions are usually combined by simple averaging, which allows us to clearly measure the amount of disagreement between the base models.

Let $f_1, f_2, \ldots, f_M$ denote M estimators, where the m-th estimator is trained on a sample training set Z_m, $m=1, \ldots, M$. To obtain a single prediction for a given test instance $\mathbf{x}$, the outputs of the individual estimators are combined by a uniformly weighted average:

$$y_{ens} = f^M_{ens}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} f_i(\mathbf{x}). \tag{4}$$

If the ensemble is treated as a single learning unit, its bias–variance decomposition can be formulated as previously. However, as shown by [31], this decomposition can be extended for the case of ensembles to take into account the correlation among ensemble members: this extension is known as the bias–variance–covariance decomposition of the error. Namely, the generalization error of the ensemble estimator is affected by each factor of interest: bias, variance, covariance, and noise.

The generalization error of the ensemble estimator given by (4) can be decomposed into:

$$E\left\{(y_{ens}-z)^2\right\} = \overline{bias}^2 + \frac{1}{M}\overline{var} + (1-\frac{1}{M})\overline{cov}. \tag{5}$$

This decomposition illustrates that, in addition to the bias and variance of individual estimators, the generalization error of an ensemble also depends on the covariance between the individuals. A negative correlation among two ensemble members contributes to a decrease in the generalization error. Conversely, a positive correlation will increase the generalization error.

The *bias–variance* and the *bias–variance–covariance* decompositions are performed online and periodically using the same holdout testing data set and at the same predefined time steps. In this way, we are able to monitor the evolution of all the components of the squared error over time. Note that, with the described evaluation methods, there is no need to block the data stream or lose any parts of the data. All of the examples used for testing are buffered in the order of their arrival and sequentially dispatched to the learning algorithm for further training.

5.4. Data sets

For the purpose of our study, we decided to use all the available regression datasets of sufficient size, as we are interested in studying how the learning methods behave in different scenarios. From the UCI Machine Learning Repository [32], the Delve Repository[2] and the StatLib System's[3] site, we chose the nine largest available "benchmark" datasets on which no concept drift has been reported. Their characteristics are given in Table 2. We also evaluate the studied approaches on three real-world datasets, described below.

Infobiotics PSP data sets: The PSP repository[4] contains real-world tunable scalability benchmark datasets for protein structure prediction. Each numerical attribute represents one value of the Position-Specific Scoring Matrices (PSSM) profile of a single amino acid from the primary protein sequence. The target attribute is the structural feature of one of the amino acids, positioned at a specified location in the chain sequence. The task is to predict the structural feature of the target from the local context, that is, given the window of neighboring amino acids. For the purpose of

[2] http://www.cs.toronto.edu/~delve/data/datasets.html
[3] http://lib.stat.cmu.edu/index.php
[4] http://icos.cs.nott.ac.uk/datasets/psp_benchmark.html

Table 2
Characteristics of the datasets used for the experimental evaluation.

Dataset name	Size	Number of attributes		Values of the target
		Nominal	Numerical	Mean ± Std.
Abalone	4177	1	8	9.934 ± 3.224
Cal housing	20,640	0	9	206855.817 ± 115395.616
Elevators	16,599	0	19	0.022 ± 0.007
House 8L	22,784	0	9	50074.44 ± 52843.476
House 16H	22,784	0	17	50074.44 ± 52843.476
Mv Delve	40969	3	8	−8.856 ± 10.420
Pol	15,000	0	49	28.945 ± 41.726
Wind	6574	2	13	15.599 ± 6.698
Winequality	4898	0	12	5.878 ± 0.886

our study, we have used the largest and most difficult dataset with 380 numerical attributes, which corresponds to a window of ±9 amino acids, positioned left and right of the target. The size of this dataset is 257760 instances.

City traffic data set: This dataset was generated for the data mining competition that was part of IEEE ICDM 2010.[5] The task is to predict city traffic congestion based on simulated historical measurements. The training dataset was created for the purpose of predicting traffic congestion at road segment 10 in direction 2 and is described by 619 numerical attributes. In particular, we have used the two values measured for a given time point and a given road segment plus the corresponding lagged values measured in the last half an hour, resulting in 30 values per direction. The size of the training dataset is 59952 examples.

Airline data set: The last dataset represents a non-stationary real-world problem and was created using the data from the Data Expo competition (2009).[6] The dataset consists of a large number of records, containing flight arrival and departure details for all the commercial flights within the US, from October 1987 to April 2008. 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 was around 116 million records and 5.76 GB of memory. The target variable is Arrival Delay, given in seconds. Given that plotting and analyzing such a large number of evaluation results is difficult, we decided to use only a portion of the dataset that corresponds to the records in the period from January 2008 to April 2008. The size of this dataset is 5,810,462 examples.

6. Empirical evaluation

This section presents the results of our empirical comparison. We first compare the algorithms in terms of generalization power, using accuracy and tree size as metrics, and using error plots for visualization.

A bias–variance analysis of the errors of the compared methods is further presented. This is followed by a bias–variance–covariance analysis. We then investigate the influence of the parameter values, such as the maximum allowed tree depth and the number of trees in the ensemble. We further measure and compare the cost of using each of the algorithms in terms of memory and learning time. Finally, we evaluate the responsiveness of each learning method to changes in the target function.

[5] http://tunedit.org/challenge/IEEE-ICDM-2010
[6] http://stat-computing.org/dataexpo/2009

6.1. Squared loss and model size

To assess the generalization power of each learning algorithm, we have performed online holdout evaluation, which consists of a periodical evaluation of the model at pre-defined time intervals, using a separate holdout set of testing examples. In Table 3, we give the measured mean squared error in the last hold-out evaluation phase, averaged over 10 rounds of training and testing. The size of the corresponding models in terms of the total number of leaves is shown in Table 4.

Best accuracy overall was achieved by the methods which combine averaging of predictions with an enhanced tree structure, i.e., the option tree with averaging ORTO-A algorithm and the ORF (online random forests) algorithm. However, simple randomization came out to be a suboptimal diversification technique, as it was shown to have best accuracy only for 3 out of 9 datasets. Online bagging came also close to ORTO-A in terms of predictive performance which mainly shows the base model FIMT-DD performs better than its randomized version R-FIMT-DD. Another important advantage of the ORTO-A algorithm over ORF, as well as over OBag is the fact that the resulting models are smaller. The option tree had the smallest number of leaves (predictive rules). This makes it easier for the user to read and understand the model.

However, since the models are learned incrementally from the data stream the predictive performance varies over time and with it the advantage of option trees over the rest of the learning methods. Fig. 1 depicts the mean squared error, computed using sliding window holdout evaluation performed periodically for every 500 learning examples processed, using a test set of size 1000. The figure also shows the evolution of the models, i.e., the curve of models' size. Due to space limitations we show plots only for the Wine Quality dataset.

It is expected that for all of the learning methods the accuracy improves with the growth of the models but the relative improvements can only show the advantage of one algorithm over another over time. From the figure we can see that ORTO-BT and ORTO-A perform best for the first 2500 examples, at which point best performance starts to have OBag. For the House 8L dataset option trees showed best accuracy at any time, while for the rest of the datasets the general trend was that option trees show better performance in the initial phase, thus being more resilient to the cold start problem. Also, option trees use substantially smaller number of rules compared to the ensemble methods. OBag performs comparably, while ORF performs worst. ORF showed significantly worse performance for the Mv Delve dataset.

6.2. Analysis of memory and time requirements

Next, we investigate and compare the space and time requirements. Table 5 gives the averaged learning time over 10 runs (in seconds). ORTO-A is the fastest learning algorithm, excluding the single-tree learning method FIMT-DD. ORTO-BT is slower than ORTO-A because of the need to search for the best path every time a prediction is computed. The ensemble methods are much slower due to the sequential manner of updating the member hypotheses. However, if parallelized, their running time would be approximately 10 times shorter (where 10 is the number of trees in the ensemble).

Table 6 further shows the average amount of allocated memory (in MB). For most datasets ORF algorithm allocates the least amount of memory, except for FIMT-DD which learns a single model and thus allocates less memory. Fig. 2 gives the allocation of memory for each algorithm over time for one of the datasets (Cal Housing) in the left panel, and the corresponding curve showing the learning time in seconds is shown in the right panel. Visibly, the most expensive algorithm is OBag. Due to the fact the memory allocation is mainly dependent on the number of leaves in the tree, if an option tree has many option nodes (nested one below

Table 3
Predictive performance (MSE) of the FIMT-DD algorithm and the ensemble methods for online regression. The averaged mean squared error from 10 runs of sampling for the last holdout evaluation is given. The maximum tree depth is 5, the maximum number of trees is 10. The best results are shown in boldface.

Dataset	FIMT-DD	ORTO-BT	LMS
Abalone	6.69 ± 0.31	6.25 ± 0.43	7.14 ± 0.63
Cal housing	8.70E+9 ± 2.76E+9	5.24E+9 ± 1.24E+9	8.86E+9 ± 3.87E+7
Elevators	2.50E-5 ± 1.57E−6	2.40E-5 ± 1.66E−6	4.60E-5 ± 6.43E−13
House 8L	1.32E+9 ± 5.13E+7	1.18E+9 ± 3.17E+7	2.33E+9 ± 1.86E+6
House 16H	**1.56E+9 ± 5.20E+7**	1.58E+9 ± 5.04E+7	2.33E+9 ± 1.88E+6
Mv Delve	18.04 ± 3.25	18.33 ± 2.99	43.23 ± 3.96
Pol	273.63 ± 22.30	292.57 ± 76.32	1781.87 ± 0.23
Wind	48.89 ± 0.42	48.89 ± 0.42	45.12 ± 0.02
Winequality	**0.52 ± 9.07E-3**	0.53 ± 0.02	0.53 ± 0.02
Infobiotics	29.76 ± 0.46	29.47 ± 0.40	39.60 ± 2.44E−3
Dataset	ORTO-A	OBag	ORF
Abalone	5.91 ± 0.42	6.84 ± 0.08	**5.68 ± 0.35**
Cal Housing	**5.14E+9 ± 0.55E+8**	7.82E+9 ± 0.69E+9	6.51E+9 ± 0.61E+9
Elevators	2.50E-5 ± 1.59E−6	2.30E-5 ± 5.44E−7	**2.20E−5 ± 1.15E−6**
House 8L	**1.11E+9 ± 5.07E+7**	1.30E+9 ± 1.59E+7	1.53E+9 ± 2.97E+7
House 16H	**1.56E+9 ± 4.34E+7**	1.57E+9 ± 6.08E+6	1.72E+9 ± 1.98E+7
Mv Delve	**17.12 ± 2.99**	17.20 ± 0.78	54.38 ± 7.52
Pol	**231.57 ± 16.74**	265.99 ± 7.41	1223.61 ± 123.07
Wind	48.89 ± 0.42	48.65 ± 0.27	**22.04 ± 0.68**
Winequality	**0.52 ± 0.03**	**0.52 ± 7.35E−3**	0.58 ± 0.02
Infobiotics	**28.37 ± 0.19**	29.69 ± 0.25	31.56 ± 0.24

Table 4
The size of the tree models produced by the FIMT-DD algorithm and the ensemble methods for online regression. The averaged number of tree leaves from 10 runs of sampling for the last holdout evaluation is given. The maximum tree depth is 5, and the maximum number of trees is 10. The best results (with smallest number of leaves) among the ensemble methods (i.e., excluding FIMT-DD and ORTO-BT) are shown in boldface.

Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF	LMS
Abalone	6.6 ± 0.9	**42.5 ± 04.9**	10.1 ± 1.4	66.9 ± 0.3	55.6 ± 0.6	1.0 ± 0.0
Cal housing	23.5 ± 1.7	**75.3 ± 10.3**	25.9 ± 0.9	231.9 ± 0.6	212.5 ± 0.8	1.0 ± 0.0
Elevators	16.9 ± 0.9	**93.2 ± 27.5**	16.7 ± 4.2	162.1 ± 0.5	185.6 ± 0.9	1.0 ± 0.0
House 8L	17.6 ± 6.5	**79.9 ± 24.9**	24.9 ± 2.0	145.5 ± 1.5	236.9 ± 1.2	1.0 ± 0.0
House 16H	19.6 ± 1.7	**52.4 ± 08.4**	22.0 ± 0.9	192.7 ± 0.3	209.7 ± 0.5	1.0 ± 0.0
Mv delve	30.9 ± 1.2	**70.7 ± 36.3**	30.9 ± 0.9	313.7 ± 0.3	287.7 ± 0.6	1.0 ± 0.0
Pol	19.8 ± 0.4	**51.3 ± 14.9**	20.3 ± 1.8	199.8 ± 0.3	148.3 ± 0.7	1.0 ± 0.0
Wind	7.6 ± 1.4	**7.6 ± 01.4**	7.6 ± 1.4	81.6 ± 0.33	126.5 ± 0.5	1.0 ± 0.0
Winequality	9.2 ± 0.9	**48.8 ± 05.2**	12.9 ± 2.9	86.8 ± 0.3	118.1 ± 0.4	1.0 ± 0.0
Infobiotics	31.3 ± 0.9	**140.5 ± 30.4**	31.5 ± 0.53	310.3 ± 0.2	302.7 ± 0.5	1.0 ± 0.0

another) then its memory allocation might exceed the memory allocation of an ensemble, especially if the size of the trees in the ensemble is constrained to a pre-specified height.

6.3. Bias–variance analysis

In this section we present the bias–variance decomposition of the error. Table 7 shows the bias component of the error for the last sliding window holdout evaluation, averaged over 10 runs of sampling. The corresponding variance component of the error is given in Table 8.

The algorithms that performed best on individual datasets had in general the smallest bias component of the error. An exception is the ORTO-BT algorithm which has a smaller bias component of the error in general, compared to the winning ORTO-A, but at the cost of a larger variance. The resulting difference in these deltas is such that ORTO-BT was outperformed by ORTO-A. However, this tells us that the option nodes indeed provide a better way to explore the search space and find the best tree overall. The increase in the variance in ORTO-BT is also due to the way the best tree is chosen. Namely, the selection procedure is based on a weighted (faded) mean squared error estimate obtained over the observed training examples at each option node. As such, the diversification in the training data biases the selection. On the other hand, in ORTO-A the predictions are combined using averaging which decreases the variance and finally accounts for best predictive performance.

Online bagging was observed to have the smallest variance, but its bias components are larger. This is something to be expected, since bagging in knows as a method that reduces the variance, however an interesting discovery is that online option trees and online random forest in general reduce the bias component of the error. These methods improve the intrinsic ability of the learning algorithm to model the phenomenon under study. While option trees were able to also reduce the variance the online bagging algorithm was much more successful in this task.

Fig. 3 shows the typical behavior on most of the datasets. The figure shows that the bias component of the error is decreasing over time for all of the learning algorithms. As the bias decreases the variance increase correspondingly. This is natural, because with the growth of the tree its predictions improve up till the point the tree reaches optimal size. The larger the tree, however, the greater is the risk of growing over the optimal size and thus over-fitting. Additional figures showing the evolution of the bias and variance components for the rest of the datasets can be found in [33, (Appendix A.4 Figures 36 (a,b,c,d), 37 (a,b,c,d) and 38 (a,b))].

6.4. Bias–variance–covariance analysis

As described previously, when the loss function is the mean squared error one can successfully use the bias–variance–covariance decomposition for an ensemble of models and examine the source of the error, as well as, align it with the amount of correlation among the models. However, this technique is not applicable for option trees because the decomposition requires a separate prediction from each base member for every example from the hold-out evaluation dataset. The option tree will not necessarily provide multiple predictions for every example. It can return multiple predictions only for those examples that belong to the example space covered by an option node. Further, the number of different predictions for different examples does not need to be equal.

Here we present the results of the bias–variance–covariance decomposition of the mean squared error for the OBag and ORF algorithms. This analysis should provide some more insights into the stability of the models, and the level of correlation between the ensemble members, which can be correlated with the level of diversity. The obtained results, i.e., the bias, variance and covariance components of the error are given in Table 9. As trees are growing larger they become more similar, which is evident from the increase in correlation between the ensemble members. Additional details on the evolution of the three error components can be found in [33, (Appendix A.6 Figures 39 (a, b), 40 (a, b), 41 (a, b), 42 (a, b) and 43 (a, b))].

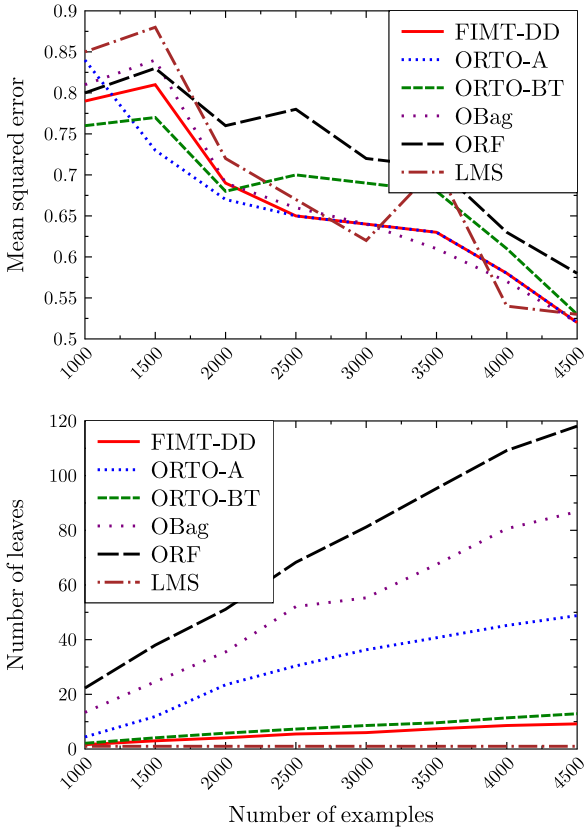


Fig. 1. Mean squared error (upper) and number of leaves (lower) measured periodically using the holdout evaluation method for the Wine Quality dataset.

The ORF algorithm is essentially a randomized version of the OBag algorithm. Thus, a comparison of their error components should give us insights on the effects of randomization. What we have observed is that the randomization method in general reduces the bias component of the error. Few datasets were however exception: the Mv Delve, Pol and Winequality datasets. This is somewhat explainable if taking into account the fact that random forests (and randomized methods in general) are expected to improve performance when the dataset is described with many features (each associated loosely with the target attribute). In addition, best results for these were achieved with ORTO-A which provides a strategy to better search the space of possible solutions. On the other hand, for the Elevators and especially for the Wind dataset introducing options was not able to advance the search of the hypotheses space, mainly because the set of the "best" options did not lead to a set of diverse and accurate hypotheses overall.

Randomizing the learning process naturally increases the variance, while online bagging mainly decreases the variance. In general, the randomization procedure in ORF typically decreases the bias for specific types of datasets, as discussed above, at the price of an increased variance. On the positive side, ORF in many cases also brings a decrease in the covariance, which at the end results in more accurate predictions. This happens for the cases when the bias component is not too high. Since the covariance is a measure of correlation among the predictions of the base ensemble models, if we use correlation as a measure of diversity we can conclude that randomization improves the diversity of the ensemble.

Although randomization might not always improve the accuracy, it still has several advantages over plain online bagging: online RandomForest has smaller memory allocation, a shorter learning time, and accuracy comparable to bagging.

To understand how the parameter k affects the predictive performance of the ensemble, we varied this parameter by systematically increasing the starting value by one. For each dataset, the initial value chosen was the square root of the number of descriptive attributes. What we have observed is that increasing the number of randomly chosen attributes eventually harms the accuracy, because of an increased correlation between the base models' predictions. For some of the datasets, by increasing the number of attributes the bias component of the error was

Table 5
The learning time for the single-tree method FIMT-DD and each of the ensemble methods, given in seconds as the average from 10 runs of sampling. The best results among the ensemble methods (i.e., excluding the single-tree method FIMT-DD and LMS) are shown in boldface.

Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF	LMS
Abalone	0.05 ± 0.01	**0.15 ± 0.01**	0.17 ± 0.02	0.50 ± 5.85E-3	0.20 ± 0.61	0.04 ± 0.01
Cal housing	0.44 ± 0.04	2.67 ± 0.57	2.82 ± 0.81	4.50 ± 0.01	**2.30 ± 0.01**	0.58 ± 0.03
Elevators	0.61 ± 0.06	4.50 ± 1.19	4.40 ± 1.12	6.10 ± 0.02	**3.90 ± 0.01**	0.42 ± 0.05
House 8L	0.52 ± 0.04	**2.53 ± 1.04**	2.55 ± 0.99	5.50 ± 0.02	2.60 ± 0. 02	0.54 ± 0.06
House 16H	1.62 ± 0.14	**5.39 ± 1.88**	5.55 ± 1.63	16.20 ± 0.04	7.30 ± 0.01	1.65 ± 0.09
Mv delve	1.24 ± 0.07	**3.52 ± 1.72**	3.55 ± 1.66	12.20 ± 0.03	5.70 ± 0.03	1.06 ± 0.09
Pol	1.42 ± 0.12	**3.37 ± 0.84**	3.44 ± 0.83	14.00 ± 0.03	8.00 ± 0.02	1.49 ± 0.07
Wind	0.22 ± 0.02	**0.21 ± 0.01**	0.22 ± 0.03	2.20 ± 8.84E-3	1.00 ± 6.76E-3	0.17 ± 0.03
Winequality	0.09 ± 0.02	**0.39 ± 0.16**	0.40 ± 0.18	0.90 ± 6.36E-3	0.50 ± 5.81E-3	0.07 ± 0.01
Infobiotics	2600.84 ± 57.56	13443.29 ± 1910.90	13526.00 ± 1888.21	26508.70 ± 48.07	**3993.50 ± 5.02**	1367.97 ± 8.92

Table 6
The memory used by the single-tree method FIMT-DD and each of the ensemble methods, shown as the average allocated memory in MB over 10 runs of sampling. The best results among the ensemble methods (i.e., excluding the single-tree method FIMT-DD and LMS) are shown in boldface.

Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF	LMS
Abalone	3.26 ± 0.26	**10.69 ± 1.87**	10.69 ± 1.87	34.10 ± 0.09	11.80 ± 0.09	3.79 ± 0.01
Cal housing	26.20 ± 1.08	130.43 ± 13.38	130.29 ± 13.21	267.20 ± 0.23	**78.60 ± 0.12**	17.78 ± 0.04
Elevators	9.45 ± 0.27	67.57 ± 14.28	67.56 ± 14.27	90.70 ± 0.16	**28.30 ± 0.14**	3.42 ± 6.32E-3
House 8L	41.55 ± 2.57	181.29 ± 57.02	181.22 ± 56.94	425.40 ± 0.73	**114.80 ± 0.25**	45.29 ± 0.06
House 16H	77.29 ± 3.39	335.95 ± 97.42	335.95 ± 97.43	779.60 ± 1.31	**235.80 ± 0.32**	100.19 ± 0.34
Mv delve	98.41 ± 1.86	265.88 ± 99.12	265.62 ± 98.21	980.80 ± 0.50	**249.70 ± 1.71**	109.38 ± 9.48E-3
Pol	9.54 ± 0.39	52.29 ± 14.03	52.11 ± 13.74	94.70 ± 0.12	**29.60 ± 0.09**	4.03 ± 5.27E-3
Wind	8.74 ± 0.38	**8.74 ± 0.38**	8.74 ± 0.38	87.30 ± 0.10	26.10 ± 0.11	5.32 ± 0.02
Winequality	3.47 ± 0.56	16.52 ± 6.46	16.52 ± 6.46	31.70 ± 0.16	**16.40 ± 0.05**	2.18 ± 5.16E-3
Infobiotics	158.62 ± 2.80	2178.51 ± 823.98	2192.16 ± 825.31	1581.80 ± 1.09	**271.10 ± 0.38**	29.94 ± 0.00

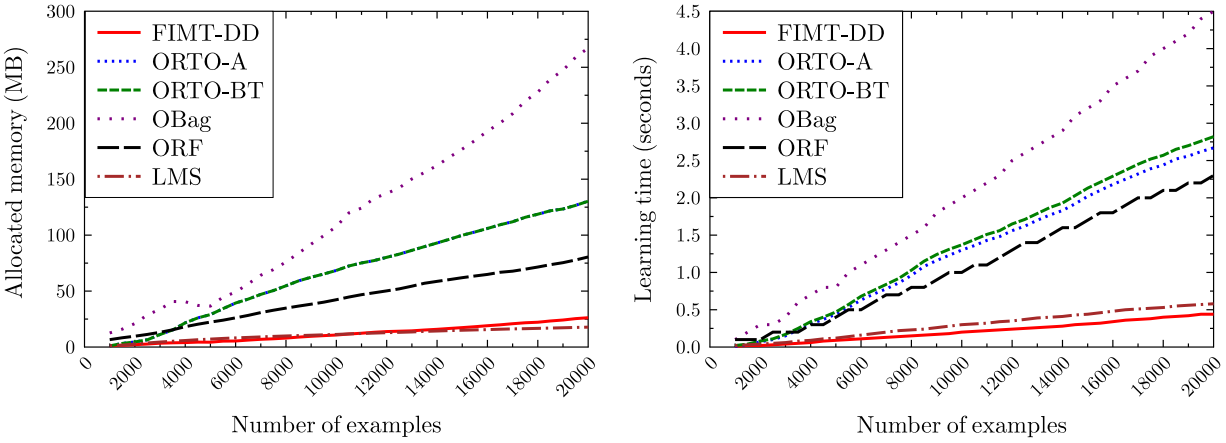


Fig. 2. The allocated memory in MB (left) and learning time in seconds (right) evaluated periodically using a sliding window holdout evaluation method for the Cal Housing dataset. Note that in the left panel, the curves for ORTO-A and ORTO-BT overlap.

significantly increased. This serves as an additional proof that the diversity of the ensemble is a significant factor for improving the performance.

Bifet et al. [25] have also studied the effect of randomizing the input and the output components of the data stream on the accuracy and the diversity of an ensemble of classifiers. Their findings are similar to ours: when applying randomization to the weights of the instances of the input stream one can achieve an increase in accuracy. Another interesting approach in introducing diversity to the ensemble of classifiers is to use trees of controlled depth [19]. Although the main motivation was to address the problem of learning drifting concepts, the method was shown to increase the diversity, which can improve the accuracy even when the concepts are stable.

6.5. Sensitivity analysis of ensemble performance

Here we present the results from the sensitivity analysis performed to assess the influence of the parameters maximum allowed tree depth l_{max} and number of trees in the ensemble T_{max}. Detailed results are given in the Appendix (Tables 34 and 35 in [33]).

When the ensemble size was fixed to $T_{max} = 10$, we observed that varying the maximal tree depth from 5 to 7, which essentially allowed the tree to grow bigger, improved the accuracy but not for all of the datasets. In particular, it was interesting to see that by restricting to a maximum of 6 levels, ORTO-A algorithm managed to show better accuracy than FIRT-DD, especially on the House 16H dataset on which FIRT-DD was initially evaluated as better. For the remaining algorithms, we observed significant increase in accuracy for the Mv Delve and the Pol datasets. On the other hand, for the Cal Housing, House 16H and Winequality datasets, increasing the allowed tree level actually decreased the accuracy of OBag and ORF. In general, for most of the datasets best results were achieved when the maximum level of depth was set to $l_{max} = 6$.

With respect to the second parameter T_{max}, increasing the maximum number of trees of fixed depth resulted in slightly improved accuracy for ORTO-A and ORF. While this trade-off is not so expensive for ORTO-A, for ORF a small increase in accuracy comes at a price of a substantially higher memory allocation. This is one of the advantages of ORTO-A over ORF.

6.6. Responsiveness to concept drift

Finally, we examine the responsiveness of each learning algorithm to changes in the target function. For this task, we used only the last two datasets, i.e., the *City Traffic* and the *Airline* dataset, on which concept drift might be expected. We have chosen to evaluate the responsiveness to changes by observing the trend of the mean squared error, i.e., its variation over time.

Fig. 4 shows (in the top and middle panels) the evolution over time of the mean squared error and the evolution of the total number of leaves for each ensemble method on the City Traffic (left), and the Airline dataset (right). When looking at the evolution of the MSE best performance and thus best responsiveness to the changes in the target function was observed for both FIMT-DD and ORTO-A. ORTO-BT and ORF were observed to behave in a highly unstable fashion, while OBag was somewhere in-between. None of the algorithms detected concept drift on the City Traffic dataset.

Table 7
The bias component in the bias–variance decomposition of the error for the FIMT-DD and LMS algorithms, and the ensemble methods for online regression. Averages are given from 10 runs of sampling for the last holdout .

Algorithm	Abalone	Cal housing	Elevators	House 8L	House 16H	Mv Delve	Pol	Wind	Winequality	Infobiotics
FIMT-DD	6.39	7.25E+9	2.60E-5	1.15E+9	1.39E+9	13.42	235.20	41.35	0.49	28.45
ORTO-A	5.59	4.75E+9	2.40E-5	**1.05E+9**	1.42E+9	13.86	**191.08**	41.35	0.49	**27.87**
ORTO-BT	5.74	**4.47E+9**	2.20E-5	1.07E+9	**1.37E+9**	13.16	207.75	41.35	**0.48**	28.11
OBag	6.53	6.37E+9	2.60E-5	1.18E+9	1.43E+9	**11.96**	227.20	41.02	0.51	28.44
ORF	**4.63**	4.54E+9	**2.10E-5**	1.25E+9	1.42E+9	36.44	927.18	**15.07**	0.52	29.62
LMS	6.61	8.86E+9	4.6E-5	2.33E+9	2.33E+9	41.85	1781.79	45.11	0.50	39.60

Table 8
The variance component in the bias–variance decomposition of the error for the FIMT-DD and LMS algorithms, and the ensemble methods for online regression. Averages are given from 10 runs of sampling for the last holdout.

Algorithm	Abalone	Cal housing	Elevators	House 8L	House 16H	Mv Delve	Pol	Wind	Winequality	Infobiotics
FIMT-DD	0.29	1.45E+9	0.10E-5	0.17E+9	0.16E+9	4.62	38.44	7.54	0.02	1.31
ORTO-A	0.31	0.39E+9	0.30E-5	0.62E+8	0.14E+9	3.26	40.49	7.54	0.03	0.50
ORTO-BT	0.50	0.77E+9	0.40E-5	0.11E+9	0.21E+9	5.17	84.82	7.54	0.06	1.36
OBag	**0.02**	0.13E+9	**0.00**	0.12E+8	0.15E+8	**0.38**	3.87	0.59	**0.00**	0.11
ORF	0.09	0.17E+9	**0.00**	0.28E+8	0.28E+8	2.04	28.24	0.59	0.01	0.17
LMS	0.52	**0.82E+5**	**0.00**	**0.92E+5**	**0.12E+6**	1.38	**0.08**	**0.98E-3**	0.02	**1.30E-5**

For the Airline dataset on the other hand all of the algorithms detected large number of changes, which resulted in many updates to the models. This can be seen in Fig. 4, in the middle panel, where we can see the number of leaves for each method measured in a periodical fashion. It is also visible that there is no clear winner, however ORF performed worse than the other algorithms. The memory allocation on these two datasets is given at the bottom of Fig. 4.

7. Conclusions and future work

Noticing the apparent lack of algorithms for online learning of ensembles for any-time regression, in this work we presented a survey on the state-of-the-art methods for online learning of tree-based ensembles for classification. We further designed, implemented and empirically evaluated two new online ensemble learning algorithms for any-time online regression: OBag and ORF. The first method is an online bagging procedure of adaptive online model trees built using the FIMT-DD algorithm. The second method is an online random forest of randomized online adaptive model trees built using the R-FIMT-DD algorithm presented here for the first time.

We performed an extensive experimental comparison of the two new ensemble learning methods to a baseline linear regression approach (adaptive LMS filter); online model trees induced with the FIMT-DD algorithm; and online option trees with two different prediction rules where ORTO-A uses non-weighted averaging and ORTO-BT uses adaptively the best model's predictions. We compared these methods on a variety of datasets along several dimensions: predictive accuracy and quality of models, time and memory requirements, bias–variance and bias–variance–covariance decomposition of the error, sensitivity to parameter values and responsiveness to concept drift. By using a bias–variance–covariance decomposition of the error we investigated and presented theoretically supported method to measure the diversity of an ensemble of models.

The results from the empirical analysis showed that online option trees with averaging were the most stable and accurate method for the majority of the datasets. Having the lowest bias component of the error option trees were indeed shown to provide a better way to explore the search space and find the best possible tree. Online bagging of adaptive model trees as expected was observed to reduce the variance component of the error. Online random forests of randomized FIMT-DD trees on the other hand improve the diversity of the ensemble, as well as the bias component of the error.

The online option trees besides being most accurate were also the fastest learning method, excluding the FIMT-DD and R-FIMT-DD algorithms, which can only produce one model tree. The ensemble methods are the slowest ones, although, if

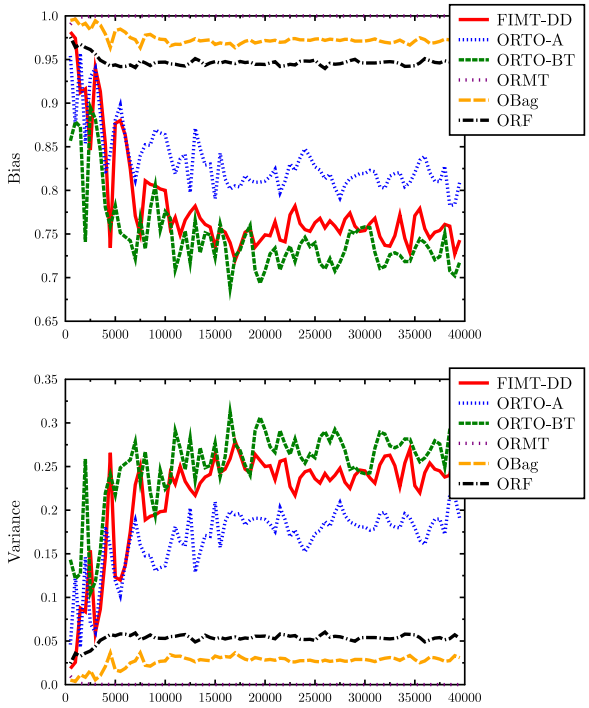


Fig. 3. Relative bias and variance error components of the single-tree method FIMT-DD and the different ensemble methods computed using a sliding window bias–variance decomposition on the Mv Delve dataset.

Table 9
The bias, variance and covariance components of the error from the bias–variance–covariance decomposition averaged across 10 runs of sampling for the last holdout on the UCI and Infobiotics datasets.

Dataset	Bias		Variance		Covariance	
	OBag	ORF	OBag	ORF	OBag	ORF
Abalone	168.86E+00	**121.04E+00**	**194.13E+00**	307.81E+00	8.95E+03	**5.43E+03**
Cal Housing	1.58E+11	**1.29E+11**	6.16E+11	6.16E+11	1.88E+13	**1.29E+13**
Elevators	6.94E-04	**5.79E−04**	**1.00E−03**	2.00E-03	**0.07E+00**	0.08E+00
House 8L	**2.37E+10**	2.51E+10	**8.49E+10**	1.08E+11	3.37E+12	**2.14E+12**
House 16H	**2.89E+10**	3.02E+10	**7.24E+10**	9.29E+10	1.98E+12	**1.21E+12**
Mv Delve	**374.59E+00**	1004.13E+00	**5.09E+03**	5.81E+03	2.28E+05	**1.11E+05**
Pol	**5.74E+03**	2.33E+04	**7.17E+04**	8.11E+04	3.55E+06	**5.89E+05**
Wind	9.31E+02	**4.67E+02**	5.04E+03	**2.54E+03**	2.30E+05	**6.61E+04**
Winequality	**1.18E+01**	1.28E+01	**1.57E+01**	2.71E+01	4.72E+02	**3.83E+2**
Infobiotics	**7.73E+2**	8.27E+02	7.49E+02	**7.30E+02**	2.89E+04	**1.84E+04**

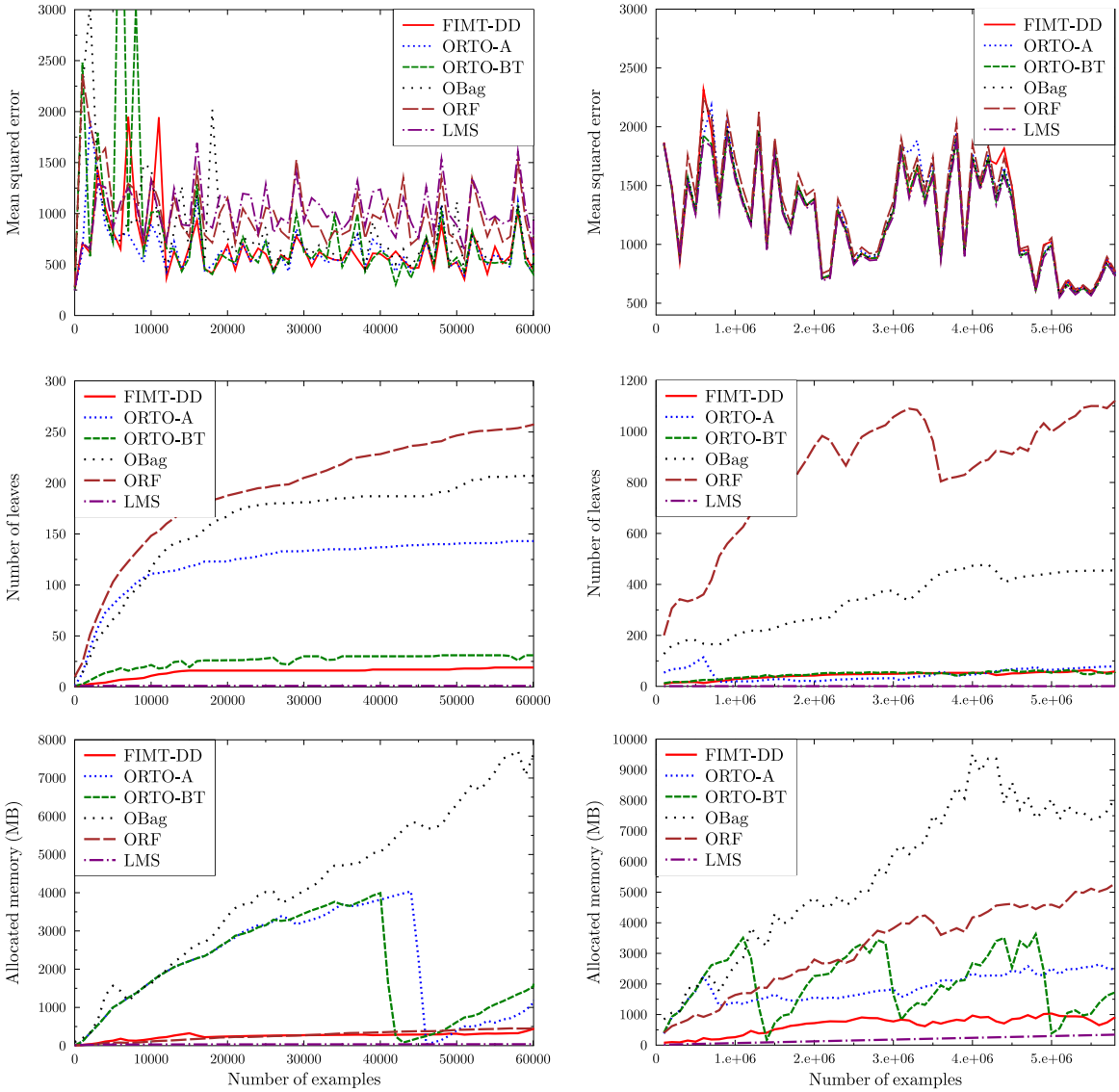


Fig. 4. Mean squared error (top), number of leaves (middle) and allocated memory in MB (bottom) for the single-tree method FIMT-DD, LMS, and the different ensemble methods. Measured periodically using a sliding window holdout evaluation method for the City Traffic (left) and Airline (right) datasets.

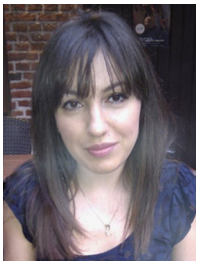
implemented through parallelization the running time would be significantly reduced. In terms of the memory allocation, ORTO-A is eligible for affordable trade-offs in terms of accuracy. For ORF however, a small increase in accuracy obtained by increasing the size of the trees comes at a price of substantially higher memory allocation.

Acknowledgments

This work was supported in part by the Slovenian Research Agency (Grant Knowledge Technologies, P2-0103), the grant 'Knowledge Discovery from Ubiquitous Data Streams (PTDC/EIA/098355/2008) and the European Commission (Grants SUMO – Super Modeling by combining imperfect models; ICT-2010-266722 and MAESTRA – Learning from Massive, Incompletely annotated, and Structured Data; ICT-2013-612944).

References

[1] L.I. Kuncheva, C.J. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Machine Learning 51 (2) (2003) 181–207.
[2] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2000, pp. 71–80.
[3] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 97–106.
[4] J. Gama, R. Rocha, P. Medas, Accurate decision trees for mining high-speed data streams, in: Proceedings of 9th International Conference on Knowledge Discovery and Data Mining, ACM, 2003, pp. 523–528.
[5] J. Gama, P. Medas, R. Rocha, Forest trees for on-line data, in: Proceedings of 19th ACM Symposium on Applied Computing, ACM, 2004, pp. 632–636.
[6] E. Ikonomovska, J. Gama, S. Džeroski, Learning model trees from evolving data streams, Data Mining and Knowledge Discovery (2010) 1–41.
[7] L. Rutkowski, L. Pietruczuk, P. Duda, M. Jaworski, Decision trees for mining data streams based on the McDiarmid's bound, IEEE Transactions of Knowledge and Data Engineering 25 (6) (2013) 1272–1279.
[8] E. Ikonomovska, J. Gama, B. Ženko, S. Džeroski, Speeding-up Hoeffding-based regression trees with options, in: Proceedings of 28th International Conference on Machine Learning, ACM, 2011, pp. 537–544.
[9] W.N. Street, Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 377–382.
[10] H. Wang, W. Fan, P.S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2003, pp. 226–235.
[11] J. Gama, P. Medas, Learning decision trees from dynamic data streams, Universal Computer Science 11 (8) (2005) 1353–1366.
[12] Y. Zhang, X. Jin, An automatic construction and organization strategy for ensemble learning on data streams, ACM SIGMOD Record 35 (3) (2006) 28–33.
[13] M. Scholz, R. Klinkenberg, Boosting classifiers for drifting concepts, Intelligent Data Analysis 11 (1) (2007) 3–28.
[14] J.Z. Kolter, M.A. Maloof, Dynamic weighted majority: an ensemble method for drifting concepts, Machine Learning Research 8 (2007) 2755–2790.
[15] P. Zhang, X. Zhu, Y. Shi, Categorizing and mining concept drifting data streams, in: Proceedings of 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 812–820.
[16] A. Tsymbal, M. Pechenizkiy, P. Cunningham, S. Puuronen, Dynamic integration of classifiers for handling concept drift, Information Fusion 9 (1) (2008) 56–68.
[17] P. Li, X. Wu, X. Hu, Q. Liang, Y. Gao, A random decision tree ensemble for mining concept drifts from noisy data streams, Applied Artificial Intelligence 24 (7) (2010) 680–710.
[18] N.C. Oza, S.J. Russell, Experimental comparisons of online and batch versions of bagging and boosting, in: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 359–364.
[19] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavaldà, New ensemble methods for evolving data streams, in: Proceedings of 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009, pp. 139–148.
[20] A. Bifet, R. Gavaldà, Learning from time-changing data with adaptive windowing, in: Proceedings of 7th SIAM International Conference on Data Mining, SIAM, 2007.
[21] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Computer and System sciences 55 (1) (1997) 119–139.
[22] R. Pelossof, M. Jones, I. Vovsha, C. Rudin, Online coordinate boosting, in: Proceedings of 12th International Conference on Computer Vision Workshops, IEEE Computer Society, 2009, pp. 1354–1361. URL ⟨http://arxiv.org/abs/0810.4553v1⟩.
[23] H. Abdulsalam, D.B. Skillicorn, P. Martin, Streaming random forests, in: Proceedings of 11th International Database Engineering and Applications Symposium, IEEE Computer Society, 2007, pp. 225–232.
[24] H. Abdulsalam, D. B. Skillicorn, P. Martin, Classifying evolving data streams using dynamic streaming random forests, in: Proceedings of 19th International Conference on Database and Expert Systems Applications, Springer-Verlag, 2008, pp. 643–651.
[25] A. Bifet, G. Holmes, B. Pfahringer, Leveraging bagging for evolving data streams, in: Proceedings of the European Conference on Machine learning and Knowledge Discovery in Databases, Springer-Verlag, 2010, pp. 135–150.
[26] D.H. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241–259.
[27] K.M. Ting, I.H. Witten, Issues in stacked generalization, Artificial Intelligence Research 10 (1) (1999) 271–289.
[28] A. Bifet, E. Frank, G. Holmes, B. Pfahringer, Ensembles of restricted Hoeffding trees, ACM Transactions on Intelligent Systems and Technology 3 (2) (2012) 30:1–30:20.
[29] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32.
[30] R. Kohavi, C. Kunz, Option decision trees with majority votes, in: Proceedings of 14th International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 1997, pp. 161–169.
[31] N. Ueda, R. Nakano, Generalization error of ensemble estimators, in:Proceedings of 1996 IEEE International Conference on Neural Networks, vol. 1, IEEE Computer Society, 1996, pp. 90–95.
[32] A. Frank, A. Asuncion, UCI machine learning repository (2010). URL ⟨http://archive.ics.uci.edu/ml⟩.
[33] E. Ikonomovska, Algorithms for learning regression trees and ensembles on evolving data streams. (PhD Thesis), 2012.



Elena Ikonomovska received her Ph.D. in 2012 from the Jožef Stefan International Post-graduate School in Ljubljana. She is a research engineer in applied science at Turn Inc., Redwood City, California, performing original scientific research in the field of large-scale data mining technologies for optimal ad selection in real-time display and social ad-serving platforms. Her research interests focus on analyzing digital behavioral data or streams of multi-dimensional sensory information, online learning from evolving concepts, as well as, multi-relational data streams, and learning from multiple information sources.

João Gama received his Ph.D. degree in Computer Science at the University of Porto, Portugal. He is a researcher at LIAAD, the Laboratory of Artificial Intelligence and Decision Support at INESC TEC. His main research interest is Data Mining focusing in Learning from Data Streams. He recently authored a book on Knowledge Discovery from Data Streams.

Sašo Džeroski received his Ph.D. degree in Computer Science from the Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Slovenia, in 1995. He is a scientific councilor at the Department of Knowledge Technologies, Jozef Stefan Institute and the Centre of Excellence for Integrated Approaches in Chemistry and Biology of Proteins, both in Ljubljana. He is also a professor at the Jozef Stefan International Postgraduate School in Ljubljana. His research interests include data mining and machine learning and their applications in environmental sciences (ecoinformatics and systems ecology) and life sciences (bioinformatics and systems biology). Since 2008, he has the title ECCAI fellow, awarded by the European Coordinating Committee for Artificial Intelligence for pioneering research and organizational work in artificial intelligence.