Machine Learning, 61, 5–48, 2005
2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
DOI: 10.1007/s10994-005-1121-8

Incremental Learning of Linear Model Trees

DUNCAN POTTS duncanp@cse.unsw.edu.au
CLAUDE SAMMUT claude@cse.unsw.edu.au
School of Computer Science and Engineering and ARC Centre of Excellence for Autonomous Systems, University of New South Wales, Sydney, NSW 2052, Australia

Editor: Johannes Fürnkranz
Published online: 09 June 2005

Abstract. A linear model tree is a decision tree with a linear functional model in each leaf. Previous model tree induction algorithms have been batch techniques that operate on the entire training set. However there are many situations when an incremental learner is advantageous. In this article a new batch model tree learner is described with two alternative splitting rules and a stopping rule. An incremental algorithm is then developed that has many similarities with the batch version but is able to process examples one at a time. An online pruning rule is also developed. The incremental training time for an example is shown to only depend on the height of the tree induced so far, and not on the number of previous examples. The algorithms are evaluated empirically on a number of standard datasets, a simple test function and three dynamic domains ranging from a simple pendulum to a complex 13 dimensional flight simulator. The new batch algorithm is compared with the most recent batch model tree algorithms and is seen to perform favourably overall. The new incremental model tree learner compares well with an alternative online function approximator. In addition it can sometimes perform almost as well as the batch model tree algorithms, highlighting the effectiveness of the incremental implementation.

Keywords: model trees, linear regression trees, online learning, incremental learning

1. Introduction

There are many situations when incremental learning is more suitable than a batch processing technique. If the input is a continuous stream of data it may not be tractable to record all of its history and execute a batch algorithm each time an output is required. For example an agent operating in a real-time environment may need to constantly process the latest sensor information to determine the next action. A large processing delay may be unacceptable, and some form of incremental learning algorithm is required that scales linearly with the incoming data.

This article focuses on the problem of inducing a model of the environment to be used for control, however the methods developed have a potentially much wider applicability. In order to control an agent it is necessary to understand how the world evolves, and how the agent's actions affect this evolution. This knowledge is often obtained by constructing a model of the environment that can be analysed to make predictions. In control theory the model (or a parameterised family of models in adaptive control) is generally specified in advance. However we are interested in developing an entirely autonomous technique that requires minimal prior knowledge.

Linear models have been intensively studied in both control theory and regression analysis, and many stability proofs and convergence theorems exist for these systems. However most interesting real-world domains exhibit some degree of non-linearity, and both learning and control become significantly harder. A non-linear model of a continuous dynamic environment can be formulated in continuous time as

$$\dot{\mathbf{z}} = f(\mathbf{z}, \mathbf{u}) \tag{1}$$

where $\mathbf{z}$ is an n dimensional state vector, $\mathbf{u}$ is an m dimensional input, $\dot{\mathbf{z}}$ is the rate of change of $\mathbf{z}$ with respect to time, and the model f is assumed to be time-invariant (Slotine & Li, 1991). For example the dynamics of an aircraft can be formulated in this manner. The learning problem is to incrementally induce a model of these dynamics using real-time observations of the measured state variables and the actions taken by the pilot.

Common classical methods of system identification include analysing either the frequency response of the system to sine waves of different frequencies or the time-domain response to impulse or step inputs. These techniques only apply to linear systems and are not suited to the multiple-input multiple-output (MIMO) domains in which we are interested. The identification of non-linear MIMO models usually involves a transformation of the inputs such that the model f is linear in the new feature space, and standard online learning techniques can be applied (Ljung, 1987). This transformation, however, requires detailed prior knowledge of the types of non-linearity present.

Instance-based methods that store all training examples and form predictions when required can be effective. Nearest neighbour, kernel regression (Hastie & Loader, 1993) and locally weighted regression (LWR) (Atkeson, Moore, & Schaal, 1997) are increasingly sophisticated methods for making the predictions. Recently Šuc, Vladušič, and Bratko (2004) have applied LWR at the leaves of a tree of qualitative constraints to give qualitatively faithful quantitative predictions. With all these techniques, as more examples arrive making predictions becomes slower or examples must be selectively discarded. This article concentrates on alternative algorithms that maintain some type of internal model instead of storing examples, and can therefore always make fast predictions.

Standard parametric machine learning methods for incrementally inducing an unknown function f include neural networks and locally or globally optimised radial basis functions (RBFs) (Nelles, 2001). The training of neural networks can be slow, is prone to local minima, and convergence is not guaranteed. It is also difficult to estimate in advance the required network structure to achieve a pre-specified degree of accuracy. RBFs comprise a set of models that are spatially localised in the input space by fixed weighting functions. Global optimisation takes place over all local models while local optimisation divides up the problem by optimising each model separately, thus avoiding a very high-dimensional minimisation problem. If the number of receptive fields are distributed uniformly over the input domain then the number of RBFs scales exponentially with the number of dimensions, and even covering a 4-dimensional space becomes difficult. In addition the input range and approximate curvature of the function being approximated must be known in advance to specify how many receptive fields are required and how large they should be.

When there is little prior knowledge it may be more beneficial to consider non-parametric alternatives where the number of learning parameters is adjusted by the algorithm. Schaal

and Atkeson (1998) have developed an RBF-based algorithm that not only dynamically allocates models as required, but also adjusts the shape of each local weighting function. Enhanced dimensionality reduction has also been incorporated (Vijayakumar & Schaal, 2000). These algorithms perform well for data with a low intrinsic dimensionality, even if the input data itself has a high number of dimensions. However there are several parameters that are hard to specify without trial and error, and the range of inputs and a metric over the input space must be defined in advance.

A decision tree with a linear model in each leaf (a linear model tree) can also approximate a non-linear function. The induction of such trees in a batch manner has received significant attention in the literature, however the incremental induction of model trees has only recently been addressed (Potts, 2004a, b). This article expands and builds upon this work, bringing together the batch and incremental versions of the two splitting rules under a unified framework, and extending the empirical evaluation. In the framework model trees are build from the top down, using one of two statistical tests to determine both the split point and whether to carry on splitting. One statistical test compares the error in the leaf to the sum of errors on each side of a candidate split, and the other analyses the distributions of positive and negative errors. Both test the hypothesis that all the examples observed in the leaf were generated from a single linear model, and splitting occurs when this hypothesis is rejected. A stopping rule is used to limit the asymptotic tree size, and in the incremental algorithms the tree can be pruned back if an early split point is deemed to be incorrect later.

Section 2 gives a brief overview of linear model trees and the online linear regression techniques used in this article. Section 3 surveys related work on both the incremental induction of classification trees and the batch induction of linear model trees. Sections 4 and 5 describe the batch and incremental versions of the algorithm, and Section 6 details the method for smoothing predictions. Section 7 presents the experimental results, the implications are discussed in Section 8 and finally we conclude and suggest areas for future work.

2. Linear model trees

The problem of learning the n dimensional function in (1) can be divided into the n smaller problems of separately learning each dimension, or component, of $\dot{\mathbf{z}}$. Each of these tasks can be formulated as a regression problem

$$y = f(\mathbf{z}, \mathbf{u}) + \epsilon$$

where $f(\mathbf{z}, \mathbf{u})$ is the corresponding component of $\dot{\mathbf{z}}$. The observed values y are corrupted by zero-mean noise ϵ with unknown variance σ^2. The aim of a regression analysis is to find an approximation $\hat{f}$ to f that minimises some cost function (e.g. sum of squared errors) over a set of training examples.

For a linear model tree the estimate $\hat{f}$ is a binary decision tree with a linear model in each leaf (see Figure 1). Each internal node in the tree contains a splitting decision based on the regressor variables that partitions the data into two subsets corresponding to the left and right sub-trees. Often the splits are restricted to being axis-parallel and the decision tree

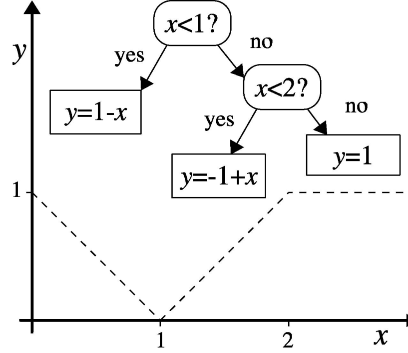


Figure 1. Example of a simple linear model tree with one regressor x.

partitions the input space into multi-dimensional rectangles. In these rectangles the estimate is linear with respect to the regressors and standard linear optimisation techniques can be applied. Within each leaf

$$\hat{f}(\mathbf{z}, \mathbf{u}) = \hat{f}(\mathbf{x}) = \mathbf{x}^T \hat{\boldsymbol{\theta}} \tag{2}$$

where $\mathbf{x}$ is a d dimensional column vector constructed from the $d - 1$ numeric predictors and a constant (included to simplify the notation), and $\hat{\boldsymbol{\theta}}$ is a column vector of d parameters. For each example i the difference between the observed value and the model prediction is

$$e_i = y_i - \hat{f}(\mathbf{x}_i) \tag{3}$$

and the linear least squares estimate $\hat{\boldsymbol{\theta}}_{\mathrm{LS}}$ of the function f is the value of $\hat{\boldsymbol{\theta}}$ that minimises the sum of these squared errors

$$J = \sum_{i=1}^{N} e_i^2$$

over the N training examples $\langle \mathbf{x}_i, y_i \rangle$ in the leaf. An analytical solution to this minimisation can be obtained by stacking the N equations on top of each other. If we define the $N \times 1$ vector $\mathbf{y}$, the $N \times d$ matrix $\mathbf{X}$ and the $N \times 1$ vector $\mathbf{e}$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \qquad \mathbf{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix}$$

then (3) becomes

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\theta}}$$

Table 1. Recursive least squares (RLS) algorithm.

Initialise $\mathbf{P}_N = [\mathbf{X}^T\mathbf{X}]^{-1}$, $\hat{\boldsymbol{\theta}}_N$ and RSS_N from an initial collection of N examples using (4) and (5).

For each subsequent example $\langle \mathbf{x}_i, y_i \rangle$ compute

$$\hat{\boldsymbol{\theta}}_i = \hat{\boldsymbol{\theta}}_{i-1} + \frac{\mathbf{P}_{i-1}\mathbf{x}_i\left(y_i - \mathbf{x}_i^T\hat{\boldsymbol{\theta}}_{i-1}\right)}{1 + \mathbf{x}_i^T\mathbf{P}_{i-1}\mathbf{x}_i}$$

$$\mathbf{P}_i = \mathbf{P}_{i-1} - \frac{\mathbf{P}_{i-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{P}_{i-1}}{1 + \mathbf{x}_i^T\mathbf{P}_{i-1}\mathbf{x}_i}$$

$$\text{RSS}_i = \text{RSS}_{i-1} + \left(y_i - \mathbf{x}_i^T\hat{\boldsymbol{\theta}}_{i-1}\right)\left(y_i - \mathbf{x}_i^T\hat{\boldsymbol{\theta}}_i\right)$$

and the quadratic $J = \mathbf{e}^T\mathbf{e}$ can be minimised with respect to $\hat{\boldsymbol{\theta}}$. This gives the linear least squares estimate

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = [\mathbf{X}^T\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{y} \tag{4}$$

The residual for each example is the difference between the value y_i and the least squares prediction $\mathbf{x}_i^T\hat{\boldsymbol{\theta}}_{\text{LS}}$, and the residual sum of squares (RSS) is the minimum value of J

$$\text{RSS} = \sum_{i=1}^{N}\left(y_i - \mathbf{x}_i^T\hat{\boldsymbol{\theta}}_{\text{LS}}\right)^2 \tag{5}$$

If the true function f is indeed linear over the area approximated by the decision tree leaf and the noise is assumed to be independent and Gaussian, then it can be shown that RSS/σ^2 is χ^2-distributed with $N-d$ degrees of freedom (Ljung, 1987).

The great advantage of the least squares estimate in an incremental setting is that a simple online update can calculate each successive estimate and RSS value. Given the previous estimate and RSS value at time $i-1$ and the next example at time i, the recursive least squares (RLS) algorithm shown in Table 1 determines the next estimate and RSS value (Haykin, 2002; Ljung, 1987)

$$\hat{\boldsymbol{\theta}}_{i-1}, \text{RSS}_{i-1}, \langle \mathbf{x}_i, y_i \rangle \xrightarrow{\text{RLS}} \hat{\boldsymbol{\theta}}_i, \text{RSS}_i \tag{6}$$

The algorithm can be started by collecting d or more examples and solving (4) and (5) to obtain initial values for $\hat{\boldsymbol{\theta}}$ and RSS. Each online update takes time $O(d^2)$.

It is therefore straightforward to maintain the linear least squares estimate in each leaf of a model tree in an incremental manner. The difficulty lies in defining the tree structure itself.

3. Related work

In this section we survey previous work on the incremental induction of trees and show that existing techniques do not fulfil our requirement of a fully online algorithm that is

guaranteed to be fast. The literature on constructing linear model trees in a batch setting is also examined, giving insights into how an incremental method can be developed.

3.1. Incremental classification tree induction

Schlimmer and Fisher (1986) proposed an incremental version of Quinlan's ID3 classification tree learner that maintains summary statistics at each node. The information measure that determines splits is re-calculated from these statistics after every example. This may lead to a leaf node being split, or a revision of a splitting rule higher up the tree and the discarding of the sub-tree below the new rule. Although the algorithm may require more examples to learn a given concept, no examples are stored in the tree and a strict upper bound can be placed on the processing time for a single example. The same principles of incremental tree induction and pruning (but no restructuring) are followed in this paper.

Utgoff, Berkman, and Clouse (1997) also consider the incremental induction of classification trees. One of their aims is that the online algorithm generates an identical tree to one built in a batch manner, and therefore that the resultant tree is invariant to the order in which the examples are processed. It is quite possible that a particular order of training data may result in an incorrect tree initially which must later be restructured, and unfortunately it proves necessary to store all training examples to guarantee that the restructuring meets their aims. Although the average incremental update is fast, in some cases it can take longer than re-building the entire tree.

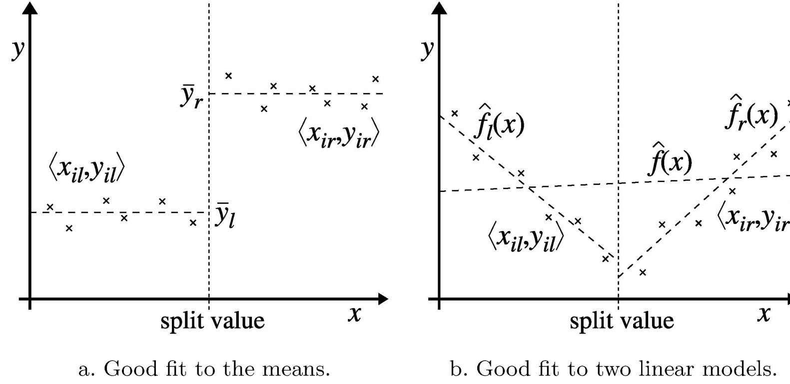
The parti-game algorithm (Moore & Atkeson, 1995) uses an incrementally grown decision tree to form a variable resolution partition of the state space. However a leaf is split on the basis of how well the control policy can be expressed, and not on the characteristics of the examples within the leaf. This technique and later work (Munos & Moore, 2002) therefore finds a good solution to the problem at hand, but does not seek to learn a general model that can be re-used for alternative tasks.

Last (2002) periodically re-learns the classifier with a batch algorithm from a moving window of training examples. The size of the window is dynamically adjusted according to how much concept drift is observed in a non-stationary environment. Here, however, the target concept is assumed to be stationary and the internal model is updated incrementally after *every* training example.

Statistical tests are well suited to incremental algorithms because the test statistics can often be easily updated. Such a test is used by Gama, Rocha, and Medas (2003) to decide whether to split a classification tree leaf, although they do not consider the possible restructuring required due to a shifting input distribution.

3.2. Batch induction of linear model trees

The most common approach to building trees from a training set is to start at the root and perform top down induction. At each node the training set is recursively partitioned using a splitting rule until the tree is sufficiently accurate. A number of alternative rules have been proposed for the induction of linear model trees. Denote the N examples at a particular



a. Good fit to the means. b. Good fit to two linear models.

Figure 2. Splitting the data into two subsets.

node as $\langle \mathbf{x}_i, y_i \rangle$. A potential split divides these examples into two subsets. Denote the N_l examples in the subset to the left of the split $\langle \mathbf{x}_{il}, y_{il} \rangle$, and the N_r in the right subset $\langle \mathbf{x}_{ir}, y_{ir} \rangle$.

Figure 2(a) illustrates the situation when a constant is assigned to each leaf. The total error (sum of the distances from each example to the constant leaf value) is clearly minimised when the leaves are assigned the means $\bar{y}_l$ and $\bar{y}_r$. The original work by Breiman et al. (1984) chooses the split that minimises either a measure of the variance or the absolute deviation of the y values. M5 (Quinlan, 1993b) and M5′ (Frank et al., 1998; Wang & Witten, 1997) choose the split that minimises a measure of the standard deviation and HTL (Torgo, 1997) also minimises the variance. All these techniques measure error from the average y value, even though linear models will be fitted each side of the split. This discrepancy, or incoherence (Malerba et al., 2004), between the split evaluation function and the model tree may result in poor splits.

Figure 2(b) on the other hand shows fitted linear models $\hat{f}_l$ and $\hat{f}_r$ constructed from the examples on either side of the candidate split. If the leaf is to contain a linear model then clearly distance to the mean value is an inappropriate measure of error, and distance to the linear regression plane should be used instead. This more suitable measure is used by RETIS (Karalič, 1992) which selects the split that minimises the total RSS over the two subsets

$$\mathrm{RSS}_l + \mathrm{RSS}_r = \sum_{i=1}^{N_l} (y_{il} - \hat{f}_l(\mathbf{x}_{il}))^2 + \sum_{i=1}^{N_r} (y_{ir} - \hat{f}_r(\mathbf{x}_{ir}))^2 \tag{7}$$

The minimisation of (7) requires the calculation of the linear least squares estimate $\hat{\boldsymbol{\theta}}_{\mathrm{LS}}$ using (4) for the two subsets of examples on each side of every candidate split. The number of potential split points increases with the number of examples (assuming the regressors are drawn from a continuous set), and it quickly becomes intractable to test all possible splits, even using the computationally efficient method of Torgo (2002).

Alexander and Grimshaw (1996) reduce the complexity by only considering simple linear models (with a single regressor) in each leaf, but this limits the representation to surfaces

with axis-orthogonal slopes. Malerba et al. (2004) build up a multivariate linear model using a sequence of simple linear regressions hence simplifying the split selection. This introduces a bias towards models containing both global and local contributions, and the split values near the root are only selected on the basis of a few regressors. Empirically their algorithm works well on real-world datasets that do contain both global and local effects. SECRET (Dobra & Gehrke, 2002) separates the data into two Gaussian clusters with the EM algorithm and forms a split between them, enabling oblique splits to be found. Although the EM technique is fast, it cannot be implemented online. Li, Lue, and Chen (2000) find oblique splits using principal Hessian directions, however the procedure is also iterative and requires interaction from the user. A unified framework is proposed by Gama (2004) that covers classification and regression trees, and both axis-orthogonal and oblique splits. The intractability of selecting from all possible split points for large datasets, however, is not addressed.

The above methods are effective in a batch setting where typically an overly large tree is grown initially, and a pruning process is later applied to try and optimise the prediction capability on unseen examples. In an online algorithm, however, it is desirable to limit the growth of the tree in the first place and avoid any complex post-pruning procedure. The algorithm must therefore determine not only *where* to make a split but also *when*, and the splitting rules considered so far do not help.

Fortunately this problem has also received attention in the statistics community, and the splitting rules in the next section allow us to determine the likelihood of the examples occurring under the hypothesis that they were generated from a single linear model. The best split is the one with the lowest probability under this hypothesis. However if this probability is not small enough, then no splitting should occur until further evidence is accumulated.

4. Batch induction algorithm

This section describes a new batch algorithm upon which the incremental algorithm detailed in Section 5 is based. The batch algorithm performs top-down induction starting at the root, and uses a statistical test at each leaf to determine both whether a split should be made and the position of the split. The tree size can also be limited by an optional stopping rule. The rest of the algorithm is very simple; there is no post-pruning stage and the predictions are obtained by fitting a least squares multivariate linear model to the examples in each leaf using Eq. (4). There is no attribute selection, therefore all numeric attributes are used in this regression. Only axis-orthogonal splits are considered. Table 2 defines the high level steps in the algorithm.

Two types of statistical test are considered. The first is based on the difference in residual sums of squares (RD), and the second analyses the distributions of positive and negative residuals (RA).

4.1. RD splitting rule (Batch-RD)

The question of whether the two linear models on each side of a potential split give a better estimation of the underlying function $f(\mathbf{x})$ than a single linear model can be tested as a

Table 2. Batch induction algorithm.

	function BatchInduction(Dataset ds)
1	perform statistical test on ds to determine whether to split
2	**if** probability of a linear model $< \alpha_{\text{split}}$ and stopping parameter $> \delta_0$
3	partition the data according to the split into ds_{left} and ds_{right}
4	*leftChild* = BatchInduction(ds_{left})
5	*rightChild* = BatchInduction(ds_{right})
6	**return** an internal node with children *leftChild* and *rightChild*
7	**else**
8	**return** a leaf node with a linear model constructed from ds using (4)
9	**end if**
	end function

hypothesis. The null hypothesis is that the underlying function is linear over the entire node ($H_0 : f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\theta}$) while the alternative hypothesis is that it is not. Three linear models are fitted to the examples in the node as in Figure 2(b); $\hat{f}(\mathbf{x})$ using all N examples, $\hat{f}_l(\mathbf{x})$ using the N_l examples lying to the left of the split, and $\hat{f}_r(\mathbf{x})$ using the N_r examples on the right. The residual sums of squares are also calculated for each linear model, and denoted RSS, RSS_l and RSS_r respectively. The two smaller linear models will always fit the data at least as well, and $\text{RSS}_l + \text{RSS}_r \leq \text{RSS}$. However if the alternative hypothesis is true $\text{RSS}_l + \text{RSS}_r$ will be significantly less than RSS and this can be tested using the Chow test, a standard statistical test for homogeneity amongst sub-samples (Chow, 1960). Under the null hypothesis it can be shown that the statistic

$$F_{\text{batch}} = \frac{(\text{RSS} - \text{RSS}_l - \text{RSS}_r) \times (N - 2d)}{(\text{RSS}_l + \text{RSS}_r) \times d} \tag{8}$$

is distributed according to Fisher's $\mathcal{F}$ distribution with d and $N - 2d$ degrees of freedom (d is the dimensionality as defined in Section 2). The candidate split least likely to occur under H_0 should make the best choice, and this corresponds to the F_{batch} statistic with the smallest associated p-value (probability in the tail of the distribution). Note that because N and RSS are constant over all candidate splits the p-value is minimised when $\text{RSS}_l + \text{RSS}_r$ is minimised and this splitting rule is therefore equivalent to that of RETIS (Karalič, 1992). In addition the statistic can determine whether or not to split. Denote the smallest p-value as α. A split should only be made if α is small enough to discount H_0 with the desired degree of confidence. This method was introduced by Sicilano and Mola (1994) to grow linear model trees, although they combine it with attribute selection and only give empirical results for an extremely small domain.

For larger numbers of examples it becomes intractable to test every axis-orthogonal split because a linear model must be built on each side of every split to calculate (8). Therefore a constant number κ of potential splits are defined in advance for each of the $d-1$ numeric regressors to give a total of $\kappa(d-1)$ candidate splits at each leaf node. The split values are chosen to uniformly partition the observed range of each regressor into $\kappa+1$ intervals. In the experimental evaluation $\kappa = 5$. Smaller values start to degrade the accuracy of the induced

trees, while larger values require more computation and tend not to reduce the prediction error.

It is necessary to introduce a multiple-comparison correction when testing the same hypothesis many times simultaneously. Therefore the Bonferroni correction is applied by dividing the desired maximum error by the number of tests. If the desired rate of falsely rejecting the null hypothesis is α_{split}, then the leaf should only be split if

$$\alpha < \frac{\alpha_{\text{split}}}{\kappa(d-1)} \tag{9}$$

The splitting rule can clearly be applied to any split that partitions the data into two parts, and can therefore also be applied to categorical attributes. For a categorical attribute that can take C values, there are $2^{C-1}-1$ distinct ways of partitioning these values into two sets, each of which forms a potential split. Unfortunately the greedy strategies discussed in Breiman et al. (1984) and Mehta, Agrawal, and Rissanen (1996) do not extend to the incremental framework presented in Section 5 where all potential splits must be fixed when a node is initialised. The number of categorical splits can be reduced if necessary by randomly numbering the categories and treating the attribute as numeric. The categorical attributes cannot contribute to the linear regression in each node and the residual sums of squares are calculated using only the numeric predictors. It is also possible to constrain the numeric attributes so that they only contribute to either the splitting process or the linear regressions as in Loh (2002).

4.2. *RA splitting rule (Batch-RA)*

SUPPORT (Chaudhuri et al., 1994) and GUIDE (Loh, 2002) are batch algorithms that use an alternative approach to splitting. The residuals from a linear model are computed and the distributions of the regressor values from the two sub-samples associated with the positive and negative residuals are compared. The RA splitting rule comes from SUPPORT, and is analysed here to show how the degree of confidence in the split is obtained.

Assume that $\hat{f}(\mathbf{x})$ is the linear model constructed from all N examples at the node. Each example has an associated residual which is the difference between the actual observed value and the prediction of the linear model. The N^+ examples with non-negative residuals are put into subset S^+, and the N^- with negative residuals are put into subset S^-. Label the regressor j for each example i in each subset as x_{ij}^+ or x_{ij}^-. Let $\bar{x}_j^+$ and $\bar{x}_j^-$ denote the mean of regressor j over the examples in each subset, and let s_j^2 denote the pooled variance estimate of regressor j over both subsets. The statistic

$$T_j^{(1)} = \frac{\bar{x}_j^+ - \bar{x}_j^-}{s_j\sqrt{\frac{1}{N^+} + \frac{1}{N^-}}} \tag{10}$$

tests for a difference in means. Define $z_{ij}^+ = |x_{ij}^+ - \bar{x}_j^+|$ and $z_{ij}^- = |x_{ij}^- - \bar{x}_j^-|$, let $\bar{z}_j^+$ and $\bar{z}_j^-$ denote the mean of the z values in each subset, and let w_j^2 denote the pooled variance of the

z values over both subsets. The statistic

$$T_j^{(2)} = \frac{\bar{z}_j^+ - \bar{z}_j^-}{w_j \sqrt{\frac{1}{N^+} + \frac{1}{N^-}}} \tag{11}$$

tests for a difference in variances. If the function being approximated is almost linear in the region of the node then the positive and negative residuals should be distributed evenly. However any curvature will result in different distributions of positive and negative residuals, and this can be tested using the above statistics. Under the null hypothesis that the residuals are distributed evenly, both statistics are distributed according to the Student's t distribution with $N-2$ degrees of freedom. The largest in absolute size $T = \max_{j,n} |T_j^{(n)}|$ is used to select the best split attribute j, and the corresponding split value is the average of the class means $\bar{x}_j^+$ and $\bar{x}_j^-$. Denote the probability under the Student's t distribution where $|t| > T$ as α. As for the RD splitting rule the Bonferroni correction is applied because two tests are being performed simultaneously along each of the $d-1$ numeric dimensions. Therefore a split is only made when

$$\alpha < \frac{\alpha_{\text{split}}}{2(d-1)}$$

where α_{split} is the desired rate of falsely rejecting the null hypothesis.

This statistic cannot be applied to splits on categorical attributes, which are therefore ignored when using the RA splitting rule.

4.3. Stopping rule

It is often desirable to limit the growth of the tree, especially for very large datasets. This can be achieved by estimating the contribution of a split to the overall tree accuracy, and only splitting if this contribution is large enough. The parameter

$$\delta = \frac{1}{s_y^2}\left(\frac{\text{RSS}}{N-d} - \frac{\text{RSS}_l + \text{RSS}_r}{N_l + N_r - 2d}\right) \tag{12}$$

is the scaled difference between the variance estimate using a single linear model and the pooled variance estimate using separate linear models on each side of a candidate split. Dividing by the estimated y variance over all training examples observed by the entire model tree ensures that the stopping rule is invariant to scaling in the y values. This parameter therefore gives the estimated reduction in variance if a leaf node is split. As the model tree grows and forms a more accurate approximation, δ decreases. Splitting is halted if δ falls below a certain threshold δ_0 (step 2 in Table 2). In addition a split is only made when the number of examples in each new leaf is at least three times the number of linear model parameters d.

When splitting with the RD rule RSS_l and RSS_r will have already been calculated when choosing the split, however when using the RA rule it is necessary to fit a linear model on each side of the candidate split in order to calculate (12).

5. Incremental induction algorithm

This section describes how the batch algorithm described above can be implemented incrementally. The resulting algorithm maintains a linear model tree, and performs updates to this tree with each new example. No examples are stored, and the memory usage and processing time per example are only dependent on the size of the tree, and not on the number of previous examples. Therefore the algorithm can operate on a sequential data stream and quickly provide an up-to-date model tree at any instant. In addition it can induce model trees on extremely large datasets.

Many aspects of the algorithm are the same as the batch version. The algorithm performs top-down induction starting at the root, and uses similar statistical tests and a similar stopping rule at each leaf to determine both whether a split should be made and the position of the split. Only axis-orthogonal splits are considered and the predictions are obtained by fitting a least squares multivariate linear model in each leaf using all numeric attributes. An online pruning technique is introduced to try and mitigate the effects of a misleading input data distribution.

At each time step i the incremental induction algorithm in Table 3 is called with the root node and the new example $\langle \mathbf{x}_i, y_i \rangle$ as parameters. Although multiple statistical tests are performed over time, the tests are highly dependent and any multiple-comparison correction should not be too large. Without knowing the eventual number of tests the only solution is to use a small value for α_{split}.

5.1. *RD splitting rule (Online-RD)*

The residual sums of squares are needed to calculate the RD splitting statistic (8). These are obtained by maintaining a fitted linear sub-model and associated RSS value on each side

Table 3. Incremental induction algorithm.

```
     function IncrementalInduction(Node n, Example ⟨x, y⟩)
1     update the linear model in node n with ⟨x, y⟩ using RLS (see Table 1)
2     if n is an internal node
3         if x is on the left of the split
4             IncrementalInduction(n.leftChild, ⟨x, y⟩)
5         else
6             IncrementalInduction(n.rightChild, ⟨x, y⟩)
7         end if
8         perform pruning test, and prune if required
9     else
10        (RD rule only) update the appropriate sub-models using RLS
11        perform statistical test returning a probability and a split
12        if probability of linear model < α_split and stopping parameter > δ_0
13            split the node along the split and initialise the two children
14        end if
15    end if
     end function
```

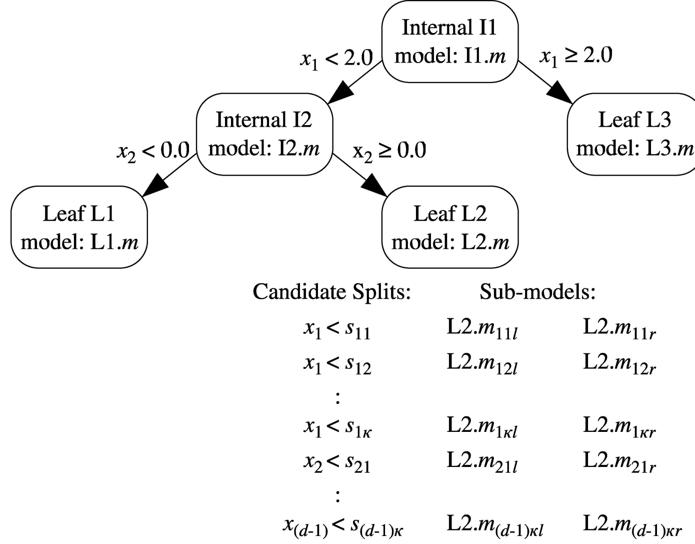


Figure 3. Labelling of the sub-models required for calculating the RD statistic.

of every candidate split using the RLS algorithm. The candidate split points are positioned uniformly over the observed range of each regressor when the node is initialised.

At each leaf node label the potential split values s_{jk} with j iterating over the $d-1$ regressors and k iterating over the κ potential splits along each regressor (see Figure 3). Step 10 in Table 3 steps through all these potential splits and updates either the left or right sub-model m_{jkl} or m_{jkr} according to whether the jth component of $\mathbf{x}$ lies to the left or right of s_{jk}.

It is possible to calculate F_{batch} as before, however in an online setting it is possible to take into account more information. When a split jk is made, the new leaf to the left of s_{jk} can be initialised with the model m_{jkl}, and the new right leaf can be initialised with m_{jkr}. For example if leaf L2 in Figure 3 is split using the rule $x_1 < s_{12}$ then L2 becomes an internal node and two new leaves are created. The models m_{12l} and m_{12r} were created from examples that lay either side of this new split value, therefore these models can form the main models m in each of the two new leaves. In each new leaf the new candidate split values are determined (by partitioning each regressor range in the leaf uniformly) and new empty sub-models are created on each side.[1] This means that the leaf model m is constructed from more examples than the sub-models in the leaf, and this invalidates the derivation of (8) which requires that $\hat{f}_l(\mathbf{x})$ and $\hat{f}_r(\mathbf{x})$ are built from the same examples as $\hat{f}(\mathbf{x})$.

Assume that the model m in a particular leaf node has been constructed from a total of N examples, and the sub-models on the left and right of the candidate split jk have been constructed from a total of N_l and N_r examples, where $N \geq N_l + N_r$. Therefore m has been formed from $N_0 = N - N_l - N_r$ more examples than the two sub-models. Under these more general conditions it can be shown (Eq. (A.1) in the Appendix with $p = 1$ and

$q = 2$) that the statistic

$$F_{\text{online}} = \frac{(\text{RSS} - \text{RSS}_l - \text{RSS}_r) \times (N_l + N_r - 2d)}{(\text{RSS}_l + \text{RSS}_r) \times (N_0 + d)} \tag{13}$$

is distributed according to the $\mathcal{F}$ distribution with $N_0 + d$ and $N_l + N_r - 2d$ degrees of freedom. This statistic is used in the incremental algorithm because it takes into account the additional information collected before the parent node was split. The p-values corresponding to each F_{online} statistic are calculated, and if the smallest one α conforms to Eq. (9) then the leaf is split. Categorical attributes are treated in the same manner as in the batch algorithm.

5.2. *RA splitting rule (Online-RA)*

The RA splitting rule only requires a single linear model to be maintained in each leaf. Unfortunately it is not possible to calculate either T statistic exactly online without storing all previous examples. This is because the classification of a residual as positive or negative requires the exact regression plane, which at any intermediate stage in a sequence of incremental calculations is an approximation to its final value. Similarly an exact calculation of z (defined in Section 4.2) requires an exact regressor mean $\bar{x}_{jk}$ which is also unavailable at intermediate stages. The incremental implementation forms an approximation to the T statistics by using the latest regression plane to classify a residual as positive or negative. Once classified the x_j components are used to update either $\bar{x}_j^+$ or $\bar{x}_j^-$, and s_j^2. The z values can then be determined and used to update either $\bar{z}_j^+$ or $\bar{z}_j^-$, and w_j^2.

When using the RA splitting rule, the stopping parameter δ cannot be calculated for each candidate split because RSS_l and RSS_r are not available in the leaf. Instead δ is calculated in the parent node, and passed down to the leaf as a parameter.

5.3. *Stopping rule*

The batch stopping rule is used online, the only difference being that the estimated y variance is incrementally updated using all examples observed so far at the root.

5.4. *Incremental pruning*

In the incremental algorithm, splits are made before the algorithm has processed all of the data. Therefore later examples may contradict the initial examples used to identify a split, and may suggest that it is more worthwhile to remove that part of the tree and replace it with a single linear model. This 'incremental' pruning should take place if the prediction accuracy of an internal node is deemed to be no worse than its corresponding sub-tree. An alternative minimum description length approach could also be taken (Robnik-Šikonja & Kononenko, 1998), but this has not been investigated. In the literature there are three major ways to estimate the prediction accuracy of a sub-tree on unseen examples: use a separate pruning set, use cross-validation on the training set, or estimate the accuracy directly from

the observed error rate on the training set. The first two methods require a set of examples to be stored and re-classified by the tree every time a pruning decision is made, and are therefore not feasible in an online algorithm. However the third method is suitable because the sum of squared errors RSS is known in each leaf (and can also be easily maintained in each internal node).

The original methods of pruning in this manner used a heuristic function to obtain the estimated prediction accuracy from the observed error rate (Cestnik & Bratko, 1991; Quinlan, 1993a). However the probabilistic framework developed above for splitting allows for a more rigorous approach that can determine the probability that pruning will be beneficial. Both the RA and RD splitting rules calculate the probability that all the examples in a node could have been generated by the null hypothesis that the true process is linear over the domain of the node. When this probability falls to a low enough value, the node is split. However if it then rises again to a high enough value α_{prune} the null hypothesis should again be accepted and the node's entire sub-tree pruned.

Both statistics require that a linear model is maintained at each internal node, and this model is updated using the RLS algorithm in step 1 of Table 3. The pruning decisions described below are made in step 8.

5.4.1. Pruning with the RD statistic. A superior RD statistic can be used that does not simply compare the internal node's model with the two models in the node's children, but with the entire piecewise linear approximation constructed by all the leaves in the node's sub-tree. The more general RD statistic (derived in the Appendix)

$$F_{\text{prune}} = \frac{(\text{RSS} - \text{RSS}_{\text{L}}) \times (N_{\text{L}} - qd)}{\text{RSS}_{\text{L}} \times (N - N_{\text{L}} + (q-1)d)} \tag{14}$$

is distributed according to the $\mathcal{F}$ distribution with $N - N_{\text{L}} + (q-1)d$ and $N_{\text{L}} - qd$ degrees of freedom, where RSS is calculated from all N examples observed at the node, RSS_{L} is the sum of the residual sums of squares in the leaves of the sub-tree, N_{L} is the total number of examples observed at the leaves and q is the number of leaves.

The node's entire sub-tree should be pruned if the p-value associated with this statistic is less than α_{prune}. No multiple-comparison correction is required because only a single F_{prune} statistic is calculated in each internal node.

5.4.2. Pruning with the RA statistic. If the same statistics used to determine splits are maintained in each internal node after splitting, then they can provide an α value at every internal node. With the Bonferroni correction, pruning should take place if

$$\alpha > \frac{\alpha_{\text{prune}}}{2(d-1)}$$

5.5. Training complexity

When the tree receives a training example it is passed down the path from the root to the corresponding leaf. At each internal node a single linear model is updated. Each RLS update

takes $O(d^2)$ and therefore it takes $O(hd^2)$ to pass the example to the leaf, where h is the maximum height of the tree. For the RD splitting rule $O(\kappa d)$ models are updated in the leaf, hence the overall training complexity is $O(N(h+\kappa d)d^2)$ where N is the total number of examples. For the RA splitting rule only a single model is updated in the leaf, and the overall training complexity is $O(Nhd^2)$.

If the tree size is limited (because of the particular input data or the stopping rule) then the algorithm fulfils our goal of scaling linearly with the number of examples. Also pleasing is the polynomial increase with dimensionality, and the fact that a strict bound can be placed on the worst case processing time for a training example when the tree has stopped growing.

Although incremental algorithms can be sensitive to the order in which the training examples are presented, results in Section 7.3 show that the tree size, and therefore the training complexity, does not vary a large amount with a shifting input distribution.

6. Smoothing

Although we have focussed on learning an approximation to the function f, what we are really interested in for control purposes is the derivative of f with respect to the state $\mathbf{z}$ and input $\mathbf{u}$ (using the notation in Eq. (1)). A popular method of non-linear control is to linearise the system about a certain operating point ($\mathbf{z}=\mathbf{z}_0$, $\mathbf{u}=\mathbf{u}_0$) using the Taylor series expansion

$$\dot{\mathbf{z}} = f(\mathbf{z}_0,\mathbf{u}_0) + \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}-\mathbf{z}_0) + \frac{\partial f}{\partial \mathbf{u}}(\mathbf{u}-\mathbf{u}_0) + \ldots \text{(higher order terms)} \tag{15}$$

where the derivatives are evaluated at the operating point. Ignoring the higher order terms gives the standard equations for a linear system

$$\dot{\mathbf{z}} = \mathbf{Fz} + \mathbf{Gu} + \mathbf{c} \tag{16}$$

which have been intensively studied in control theory. From these equations the local stability of the system and control policies to follow all possible local trajectories can be determined.

Comparing (15) and (16) shows that $\mathbf{F}$ and $\mathbf{G}$ correspond to the derivatives of f. When f (Figure 4(a)) is approximated by a linear model tree (Figure 4(b)) the gradient is constant within each leaf, and changes abruptly at the boundaries between leaves (Figure 4(c)). These discontinuities result in poor gradient estimates when the actual function is smooth. Smoothing removes the discontinuities and results in a better estimate (Figure 4(d)).

It is not the purpose of this paper to evaluate smoothing techniques, therefore a simple strategy of ‘soft’ splits is adopted as detailed in Murray-Smith (1994). Every leaf makes a prediction, and these are repeatedly combined at each internal node from the bottom up until the final prediction is produced at the root. At each internal node i the predictions from the children are weighted according to how far the data point being predicted lies from the split value s_i. If it lies on the split each child will contribute equally, whereas when it lies to one side the corresponding child will contribute more to the overall prediction. Figure 5 shows the left and right sub-tree weighting functions $\rho_{i,l}(x_j)$ and $\rho_{i,r}(x_j)$. The weighting functions are formed by two normalised Gaussian basis functions with standard deviation

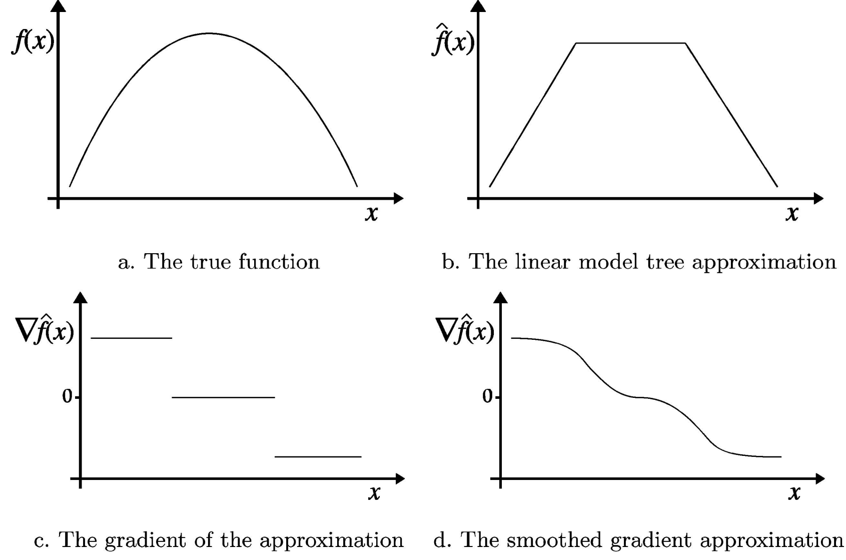


Figure 4. Smoothing the discontinuities in the gradient.

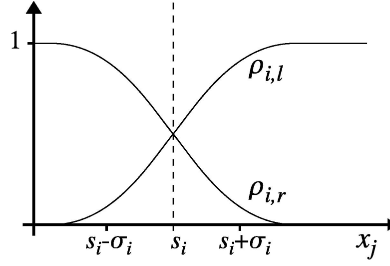


Figure 5. The left and right sub-tree weighting functions.

σ_i centred a distance σ_i each side of the split, and therefore

$$\rho_{i,l}(x_j) = \frac{1}{1 + \exp\left(\frac{2(x_j - s_i)}{\sigma_i}\right)} \qquad \rho_{i,r}(x_j) = \frac{1}{1 + \exp\left(-\frac{2(x_j - s_i)}{\sigma_i}\right)}$$

The value of σ_i is chosen to be c times the width of the internal node i, where c is a user-defined constant. For all experiments $c = \frac{1}{40}$, which is seen to give good results empirically.

The smoothing process is applied to both function and gradient estimates. It is clear for the linear model in Eq. (2) that the gradient estimate in each leaf k is simply the parameter vector $\hat{\boldsymbol{\theta}}_k$, hence the smoothed gradient estimate over all q leaves is

$$\frac{\partial \hat{f}(\mathbf{x})}{\partial \mathbf{x}} = \nabla \hat{f}(\mathbf{x}) = \sum_{k=1}^{q} \rho_{\text{cum},k}(\mathbf{x})\hat{\boldsymbol{\theta}}_k$$

where $\rho_{\text{cum},k}$ is the cumulative weight function for leaf k

$$\rho_{\text{cum},k}(\mathbf{x}) = \prod_{i=parent(k)}^{root} \rho_{i,subtree(i,k)}(\mathbf{x})$$

In this equation i iterates over the internal nodes in the path from the parent of k up to the root, and the function $subtree(i, k)$ determines whether leaf k is in the left or right sub-tree of internal node i.

7. Empirical evaluation

Firstly the algorithms are evaluated on a number of standard regression datasets from the Weka[2] machine learning repository. Comparisons are drawn with a linear model updated with the online RLS update (6), the Weka implementation of M5′ (Wang & Witten, 1997), Loh's (2002) implementation of GUIDE and Dobra and Gehrke's (2002) implementation of SECRET. Secondly the effect of the stopping rule is demonstrated when learning a 2D test function. Thirdly the ability of the incremental algorithms to withstand a shifting input distribution is examined. Finally all algorithms are demonstrated on a number of domains in which a very large number of examples can be generated. This enables the incremental algorithms to be tested on sequential data streams of one million or more examples. In these domains the incremental algorithms are compared with receptive field-weighted regression (RFWR) (Schaal & Atkeson, 1998). RFWR has since been adapted to improve its dimensionality reduction capability (Vijayakumar & Schaal, 2000), however the domains do not contain redundant dimensions and the original algorithm is more competitive. The effect of incremental pruning and the ability to predict the true model parameters are also investigated.

For all experiments the RD/RA splitting and pruning parameters are $\alpha_{\text{split}} = 0.01\%$ and $\alpha_{\text{prune}} = 0.1\%$. M5′ and GUIDE use their default parameters. 40% of the training data for SECRET is used for pruning, and the minimum node size considered for splitting is 1% of the number of training examples. SECRET is tested with both axis-orthogonal and oblique splits and results show the better of the two.

7.1. *Standard Weka datasets*

The algorithms are designed for large datasets where processing time and memory become practical limitations, therefore comparisons are only performed on the 19 Weka regression datasets with more than 1000 instances (listed in Appendix B). The number of instances varies from 4177 to 40768, and the number of attributes varies from 5 to 48.

7.1.1. Experimental methodology. The prediction capability of each algorithm is measured using 10-fold cross validation, and this process is repeated 10 times, each with a different permutation of the examples in the dataset. All algorithms use the same permutations so that they learn and test with identical partitions of data. The final estimate is the mean over all runs of the cross validation. Prediction errors are measured in terms of

normalised root mean square error (nRMSE) where the root mean square error over the n_t examples in the test set is normalised by the estimated standard deviation over the entire data set s_y

$$\text{nRMSE} = \frac{1}{s_y}\sqrt{\frac{1}{n_t}\sum_{i=1}^{n_t}(y_i - \hat{f}(\mathbf{x}_i))^2}$$

An error of 100% corresponds to always predicting the class mean. Confidence limits show one unbiased estimate of the standard deviation.

Batch/Online-RD/RA are used with no stopping rule. The incremental versions are trained on the same data as the batch versions, using their final models to make predictions.

7.1.2. Results. The detailed results are presented in Tables B.1 and B.2 in Appendix B. The reference algorithm in Table B.1 is Batch-RD, while the reference algorithm for the pruned online algorithms is the unpruned version. Algorithms are compared against the reference algorithm using the *Wilcoxon signed ranked paired test*. A $\pm$ sign indicates that for this dataset the prediction error of the algorithm is greater/less than the reference algorithm with a two-tailed p-value less than 5%. These results are summarised in Tables 4 and 5 which also show the number of wins/losses and significant wins/losses for the reference algorithm, and the p-value when the algorithm is compared against the reference algorithm using the *Wilcoxon signed ranked paired test* across all datasets. Each pair of data points consists of the average prediction error for the two algorithms on one dataset. The final two lines show the geometric means of the tree size and construction time for each algorithm over all datasets.

Batch-RD produces more accurate trees than all the other algorithms at a significance level of 97.7%. It is also faster than M5′ and GUIDE, although GUIDE does build smaller trees. Batch-RA is not nearly as effective, however it is exceptionally fast and produces trees that are much better than a linear model. SECRET is also very fast, although it fails on three of the datasets and can produce inconsistent results with a high variance across different trials. The effectiveness of Batch-RD may be partly due to the size of the datasets. More examples leads to more reliable statistical tests which may then be just as effective as the computationally expensive post-pruning techniques in M5′, GUIDE and SECRET.

Table 4. Summary of batch algorithm results for the Weka datasets.

	Batch-RD	Batch-RA	M5′	GUIDE	SECRET
Average error (nRMSE)	38.1%	46.3%	40.0%	39.1%	52.7%
Wins/losses (for Batch-RD)	–	18/1	13/6	15/4	16/0
Significant wins/losses	–	18/1	13/5	14/4	16/0
Wilcoxon two-tailed p-value	–	0.0000	0.0071	0.0230	0.0000
Number of tree leaves	27.8	9.5	35.9	11.6	8.3
Build time (seconds)	26.7	0.8	60.8	62.6	3.0

Table 5. Summary of incremental algorithm results for the Weka datasets.

	Online-RD		Online-RA		
	Unpruned	Pruned	Unpruned	Pruned	Linear
Average error (nRMSE)	41.5%	43.7%	48.2%	49.2%	61.9%
Wins/losses (for unpruned)	–	15/3	–	11/8	–
Significant wins/losses	–	12/2	–	7/3	–
Wilcoxon two-tailed p-value	–	0.0090	–	0.0874	–
Number of tree leaves	16.5	7.4	7.1	6.1	–
Build time (seconds)	64.0	61.9	3.0	2.9	0.2

As expected the incremental algorithms are slower and less accurate than their batch equivalents. However the differences in prediction error are not large, highlighting the effectiveness of the online implementations. Incremental pruning substantially reduces the tree size, in particular for Online-RD. However the pruned trees have a greater prediction error.

7.2. *Stopping rule*

In this section the effect of the stopping rule described in Section 4.3 is demonstrated when learning the same 2D test function as Schaal and Atkeson (1998) (see Figure 6)

$$y = \max\left\{e^{-10x_1^2}, e^{-50x_2^2}, 1.25e^{-5(x_1^2+x_2^2)}\right\} + \epsilon$$

where ϵ is independent zero-mean Gaussian noise with variance σ^2 and $\sigma = 0.1$. Training examples are drawn uniformly from the square $-1 \leq x_1 \leq +1$, $-1 \leq x_2 \leq +1$. Prediction

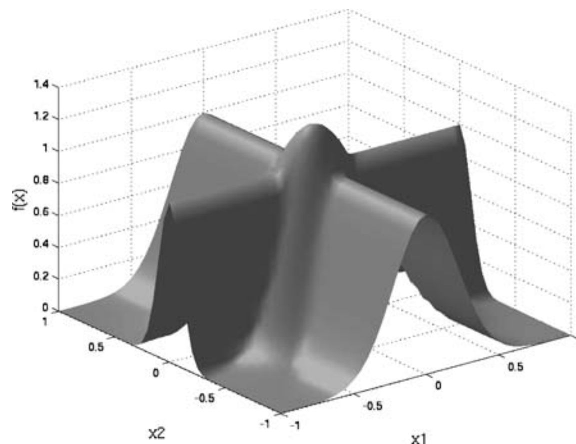
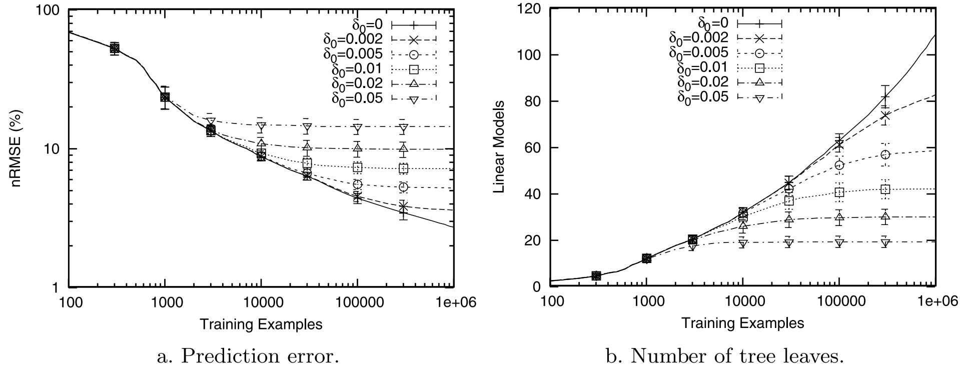
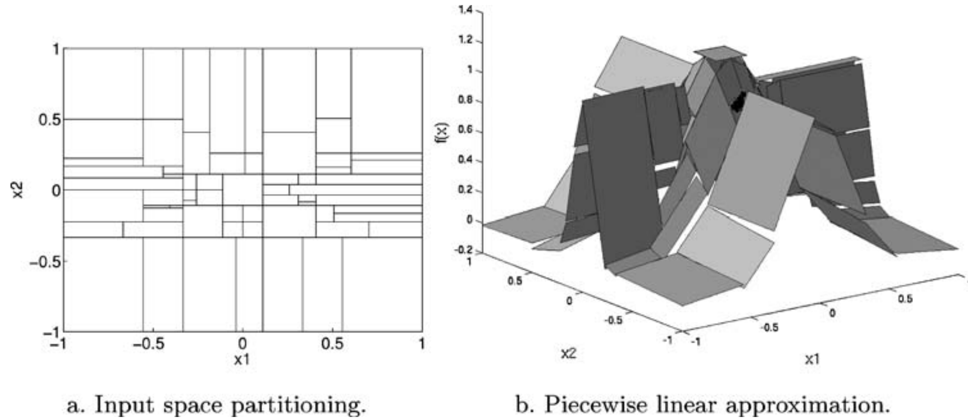


Figure 6. 2D test function.



a. Prediction error. b. Number of tree leaves.

Figure 7. Effect of stopping parameter δ_0 when learning the 2D test function with Online-RD.

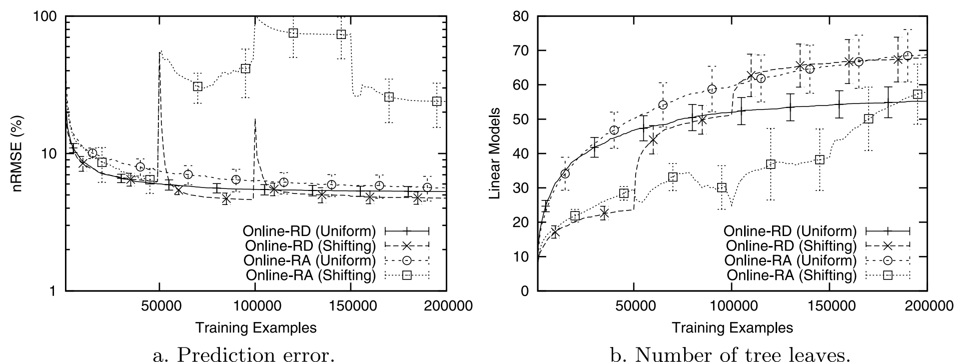


a. Input space partitioning. b. Piecewise linear approximation.

Figure 8. Model tree induced by Online-RD when learning the 2D test function.

errors are measured on a test set consisting of 2000 examples drawn in a similar manner, but without noise. Errors are measured in terms of nRMSE and results show the mean over 20 independent trials, while error bars indicate one unbiased estimate of the standard deviation.

Figure 7(a) shows the decrease in prediction error as the number of training examples presented to Online-RD is increased. Figure 7(b) shows the corresponding increase in tree size. Similar results are obtained for the RA splitting rule and the batch versions of the algorithms. It can be seen that the stopping parameter δ_0 controls a trade-off between the asymptotic prediction error and the size of the model tree. A smaller δ_0 results in a larger tree and more accurate predictions. If the stopping parameter is zero the size of the tree grows without bound, and the predictions become more and more accurate. The partitioning of the input space by the decision tree and the unsmoothed model tree predictions can be seen in Figure 8 for $\delta_0 = 0.005$.



a. Prediction error. b. Number of tree leaves.

Figure 9. Performance of the incremental algorithms on a shifting input distribution.

7.3. Shifting input distribution

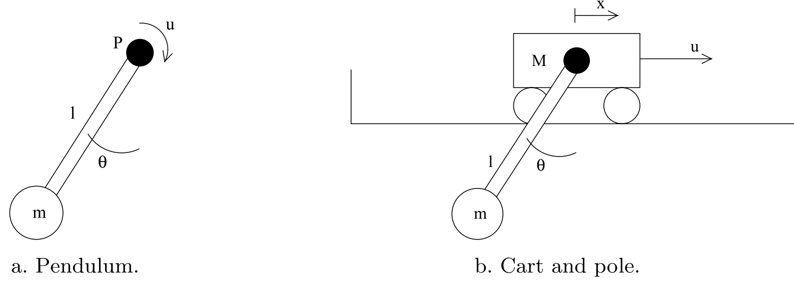
To test the resilience of the new incremental algorithms to a shifting input distribution we repeat the experiment in Schaal and Atkeson (1998) using the 2D test function defined above. The training data are presented in three distinct slightly overlapping batches. For the first 50,000 examples the training data and test data are uniformly drawn from $-1 \leq x_1 \leq -0.2$ (with x_2 ranging as before from -1 to $+1$). The second 50,000 training examples are drawn from $-0.4 \leq x_1 \leq +0.4$ while testing is performed with test data drawn from $-1 \leq x_1 \leq +0.4$. The next 50,000 examples are drawn from $+0.2 \leq x_1 \leq +1$ and the predictions tested over the entire function $-1 \leq x_1 \leq +1$. Subsequent training data is drawn uniformly from $-1 \leq x_1 \leq +1$.

Figure 9 compares the behaviour of the new incremental algorithms (with pruning), both with a uniform input distribution as before and with a shifting input distribution as described above. An increase in prediction error can clearly be observed when the shifting input distribution changes after 50,000 and 100,000 examples. Online-RD is able to quickly learn an accurate approximation to the new distribution while retaining the model associated with the old distribution. Online-RA, however, fails in this regard and cannot adjust to the new distribution. Figure 9(b) shows that the shifting distribution results in a small increase in model complexity for the Online-RD algorithm.

7.4. Dynamic domains

In this section the algorithms are tested in three dynamic domains where the task is to learn a model of the environment. The domains are:

1. A basic pendulum driven by a torque at its pivot, in both continuous and discrete time.
2. A cart and pole (also known as a pole-balancer).
3. A complex flight simulator in which 9 state variables and 4 actions form a 13 dimensional system identification problem.



a. Pendulum. b. Cart and pole.

Figure 10. Simple dynamic domains.

Results are also given for the 2D test function described in Section 7.2 and for a similar function rotated 45° about the origin (with the training and test examples obtained in an identical manner). The rotated test function removes any advantage the model tree learners may have by being restricted to axis-orthogonal splits. A brief description of the dynamic domains is now given.

7.4.1. Pendulum in continuous time. A pendulum rotating 360° around a pivot P (see Figure 10(a)) is a simple non-linear dynamic environment. The dynamic model of the pendulum is

$$ml^2\ddot{\theta} = u - mgl\sin\theta - \mu\dot{\theta} \tag{17}$$

where θ is the pendulum angle, g is gravity, $m = l = 1$ are the mass and length of the pendulum, $\mu = 0.1$ is a drag coefficient and u is the torque applied to the pendulum. This can be written in the form of Eq. (1) if $\mathbf{z} = [\theta\ \dot{\theta}]^T$. Define $\mathbf{x} = [\mathbf{z}^T\ u]^T$ and $\mathbf{y} = \dot{\mathbf{z}} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is a vector of independent zero-mean Gaussian noise with variance σ^2 and $\sigma = 0.1$. Training examples of $\langle\mathbf{x}, \mathbf{y}\rangle$ are drawn uniformly from the input domain $-\pi \leq \theta \leq +\pi$, $-5 \leq \dot{\theta} \leq +5$ and $-5 \leq u \leq +5$. The algorithms are tested using 5000 examples drawn in a similar manner, but without noise.

7.4.2. Pendulum in discrete time. The pendulum model can also be formulated in discrete time as

$$\mathbf{z}_{k+1} = f(\mathbf{z}_k, u_k) \tag{18}$$

where k is the time step index. The observed state vector $\mathbf{y}_k = \mathbf{z}_{k+1} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ has the same characteristics as above, and the regressors for $\mathbf{y}_k$ are the previous state $\mathbf{z}_k$ and action u_k. Examples of $\mathbf{x}_k = [\mathbf{z}_k^T\ u_k]^T$ are drawn uniformly from the same input domain as for the continuous time case. It is not possible to determine a closed form for f and therefore the next state of the system is calculated using 5 successive Euler integrations with a time step of 0.01 seconds to give an overall sampling rate of 20 times per second. The system identification task is to learn an approximation to the model f given examples of $\langle\mathbf{x}_k, \mathbf{y}_k\rangle$.

The algorithms are tested using 5000 examples drawn in the same manner as the training data, but without noise.

7.4.3. Cart and pole. The problem of swinging up a pendulum on a cart demonstrates the behaviour of the algorithms in a more complex domain (see Figure 10(b)). The system is highly non-linear when the pendulum is allowed to rotate through 360°. The simplified dynamic model (not taking into account frictional effects) is

$$0 = \ddot{x}\cos\theta - l\ddot{\theta} - g\sin\theta$$
$$u = (m + M)\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta$$

where x is the position of the cart (limited to ± 2), θ is the angle of the pendulum, $l = 1$ is the length of the pendulum, g is gravity, $M = m = 1$ are the masses of the cart and pendulum respectively, and u is the lateral force applied to the cart (limited to ± 7). The discrete time model (18) is formulated with $\mathbf{z}_k = [x_k\ \dot{x}_k\ \theta_k\ \dot{\theta}_k]^T$, and the next state is calculated in the same manner as for the discrete pendulum.

Instead of sampling randomly across the state space, the system is initialised at rest with the pendulum hanging vertically downward. A simple hand-coded control strategy repeatedly swings up the pendulum and balances it for a short period using the observed state vector $\mathbf{y}_k = \mathbf{z}_{k+1} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is a vector of independent zero-mean Gaussian noise with variance σ^2 and $\sigma = 0.1$. The regressors for $\mathbf{y}_k$ are the previous state $\mathbf{z}_k$ and action u_k. The sequence of states generated is given directly to the learner without changing the order, so that consecutive regressors are highly correlated. The algorithms are tested using 10,000 examples randomly drawn from a similar sequence, but without noise.

7.4.4. Flight simulator. Learning to fly an aeroplane is a complex high-dimensional task. These experiments use a flight simulator (see Figure 11) based on a high-fidelity flight model of a Pilatus PC-9 aerobatic aircraft, an extremely fast and manoeuvrable propeller plane used by the Royal Australian Air Force as a 'lead in fighter' for training pilots before

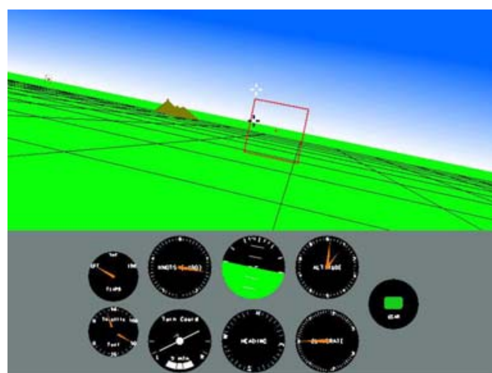


Figure 11. Flight simulator.

they progress to jet fighters. The model was provided by the Australian Defence Science and Technology Organisation and is based on wind tunnel and in-flight performance data. The same simulator has also been used in previous work (Isaac & Sammut, 2003).

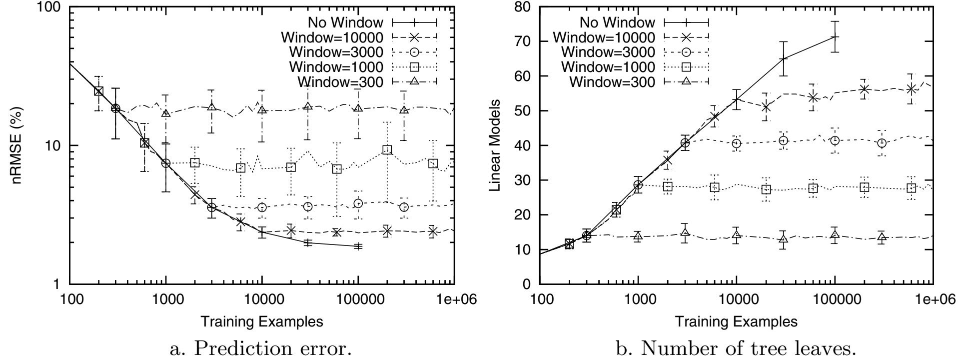
The system is sampled 4 times per second, and 9 state variables are recorded (altitude, roll, pitch, yaw rate, roll rate, pitch rate, climb rate, air speed and a Boolean variable indicating whether the plane is on the ground) along with 4 action variables (ailerons, elevator, throttle and the categorical flaps setting which can take the values 'normal', 'take off' and 'landing'). These state variables were selected so that a dynamic model of the plane could be learnt, and therefore the absolute position and heading were disregarded. The combination of states and actions results in a 13 dimensional regressor vector $\mathbf{x}$. The learning task is to predict the 8 continuous values of the next state.

As for the pendulum on a cart the training examples are taken directly from a trace of the aircraft flying so that successive regressors are highly correlated. The trace involves repeatedly taking off, flying a loop and landing back on the same runway. Simulated turbulence is set to a high level resulting in complex noise characteristics that deviate substantially from the independent Gaussian assumption, and additional noise was added to the inputs to excite the system and provide a richer source of training examples. The algorithms are tested using 10,000 examples randomly drawn from a similar trace.

7.4.5. Experimental methodology. Given a stream of examples labelled from 1 to N, the performance of a batch algorithm on n examples is estimated by training the learner with the examples labelled from 1 to n. As n is increased up to N the prediction capability of the learned model on an independent test set should improve. The test set consists of n_t examples drawn uniformly from a similar but independent stream of data.

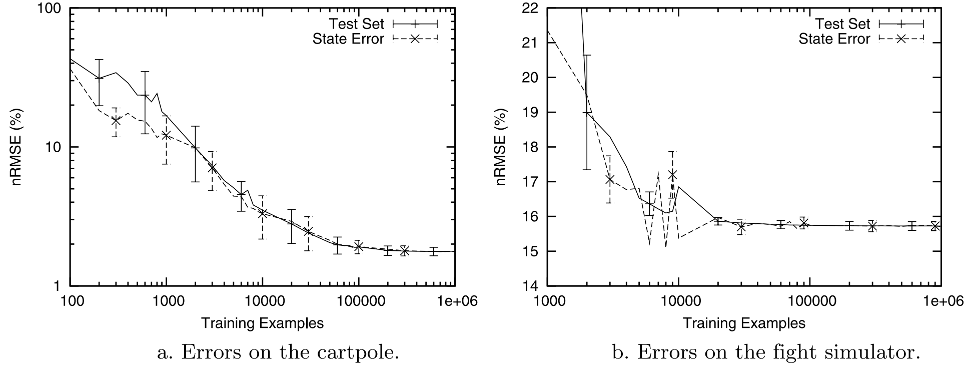
In practise a sliding window approach is often used where at time n the learner is trained with examples in the window from $n - w$ to n where w is the window size. This is very good for tracking concept drift, however in this work the environments are assumed to be time-invariant, and therefore re-training the learner with the same number of examples at a later time does not improve the estimate of the overall dynamic model. This can be seen in Figure 12 which shows the Batch-RD prediction errors and tree sizes for the cartpole domain with various window sizes. The window places a limit on the prediction accuracy and tree size. In the remaining experiments windows are not used and the batch learners are trained with all n examples. Due to time and memory limitations the maximum number of examples N for the batch learners is set to either 30 or 100 thousand.

Training the incremental learners on the N examples is straightforward, although there are two alternative testing strategies. If the prediction capability on the next example is critical (e.g. when predicting financial time series) then these 'training set' errors can be collected and averaged. However if the overall model quality is important then it is better to test on a large independent test set. When used for planning the overall model needs to be accurate so that future trajectories can be simulated, and even controllers are more dependent on the local function gradient than the quality of the next prediction. For these reasons testing is performed with the same sets of n_t examples used to test the batch algorithms. This also allows comparisons between the incremental and batch learners. In fact the two testing strategies are very similar as shown by Figure 13 which compares the error when



a. Prediction error. b. Number of tree leaves.

Figure 12. Effect of batch window sizes on the cart and pole with Batch-RD.



a. Errors on the cartpole. b. Errors on the fight simulator.

Figure 13. Difference between testing strategies with Online-RD.

predicting the next state and the error on an independent test set for both the cartpole and flight simulator. In the other domains the examples are uncorrelated and the strategies are equivalent. The incremental learners are trained with either 1 or 10 million examples.

RFWR requires normalised data, therefore its regressors and y values are scaled to the range $(-1, +1)$. Gradients are extracted from RFWR by applying the same weighting kernels to the gradients of each local model that are applied to the predictions themselves. Gradients could not be obtained from M5$'$. Gradient errors are measured using the *normalised root mean square gradient error* (Potts, 2004b) over the n_t examples in the test set

$$\text{nRMSE(Grad)} = \sqrt{\frac{\sum_{i=1}^{n_t} |\nabla(f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i))|^2}{\sum_{i=1}^{n_t} |\nabla f(\mathbf{x}_i)|^2}}$$

so that when the gradient estimate is correct everywhere nRMSE(Grad) is 0%, and when the gradient estimate is zero everywhere nRMSE(Grad) is 100%. Gradient errors can only be measured for the 2D test function and the continuous pendulum because they can be differentiated exactly.

For each domain, 25 independent streams of examples are generated. Five are used to set the stopping parameter δ_0 in the Batch/Online-RD/RA algorithms, and optimise the parameters in RFWR. For the 2D test function RFWR uses the parameters published in Schaal and Atkeson (1998) and the default learning rates in their software distribution. The stopping parameter δ_0 is set so that the Batch/Online-RD/RA prediction errors match those of RFWR. In the dynamic domains δ_0 is reduced until the prediction errors stop improving significantly. The RFWR parameters are then varied to approximately match the number of receptive fields with the number of leaves in the model tree algorithms. This enables a fair comparison of the final prediction errors and learning effectiveness. The final parameters are specified in Tables C.1 and C.2 in Appendix C. The remaining 20 streams are used for training and testing.

7.4.6. Results. The detailed results in Appendix C show the means over 20 trials, while error bars and numerical errors in tables indicate one unbiased estimate of the standard deviations. For each domain the graphs show how the errors, number of leaves/receptive fields and training times vary with respect to the number of training examples presented to the algorithms. Tables 6 to 13 give the final values for each algorithm in each domain after training with the maximum number of examples N.

Surprisingly in these domains there is not a significant difference in prediction accuracy between when GUIDE selects from all split points and when it uses a faster split point

Table 6. Summary of batch algorithm errors (nRMSE %) for the dynamic domains.

	Batch-RD	Batch-RA	M5′	GUIDE	SECRET†
2D test	5.1 ± 0.4	6.6 ± 0.4	5.5 ± 0.2	7.2 ± 0.3	7.8 ± 0.4
2D rotated	5.5 ± 0.4	6.9 ± 0.3	9.2 ± 0.2	17.7 ± 0.9	12.1 ± 3.9*
Pend. cont.	0.20 ± 0.00	0.20 ± 0.00	1.7 ± 0.1	1.3 ± 0.0	1.0 ± 2.4
Pend. disc.	0.21 ± 0.00	0.20 ± 0.00	2.5 ± 0.0	5.3 ± 0.1	0.36 ± 0.22*
Cartpole	1.9 ± 0.1	5.5 ± 0.1	3.6 ± 0.2	3.7 ± 0.1	4.1 ± 0.3*
Flight	15.2 ± 0.2	16.2 ± 0.4	15.4 ± 0.1	15.2 ± 0.1	15.8 ± 0.4

†Shows results for the algorithm with the best predictions between SECRET with axis-orthogonal splits and SECRET with oblique splits (*).

Table 7. Summary of incremental algorithm errors (nRMSE %) for the dynamic domains.

	Online-RD		Online-RA			
	Unpruned	Pruned	Unpruned	Pruned	RFWR	Linear
2D test	5.1 ± 0.5	5.2 ± 0.5	4.9 ± 1.2	4.8 ± 1.0	6.4 ± 1.8	100.0 ± 0.1
2D rotated	5.3 ± 0.4	5.3 ± 0.3	4.6 ± 0.5	4.6 ± 0.5	6.8 ± 0.2	100.0 ± 0.0
Pend. cont.	0.20 ± 0.01	0.20 ± 0.01	0.18 ± 0.01	0.18 ± 0.01	0.37 ± 0.06	40.9 ± 0.4
Pend. disc.	0.17 ± 0.03	0.17 ± 0.03	0.18 ± 0.00	0.18 ± 0.00	0.50 ± 0.44	5.3 ± 0.1
Cartpole	1.7 ± 0.2	1.8 ± 0.1	4.0 ± 3.3	3.3 ± 2.0	3.8 ± 0.2	12.6 ± 0.1
Flight	15.5 ± 0.1	15.7 ± 0.1	19.2 ± 1.9	19.1 ± 1.9	16.8 ± 0.7	21.2 ± 0.1

Table 8. Summary of batch algorithm model sizes (number of tree leaves) for the dynamic domains.

	Batch-RD	Batch-RA	M5′	GUIDE	SECRET†
2D test	64.0 ± 2.6	67.8 ± 2.6	116.8 ± 6.7	45.8 ± 1.3	57.6 ± 2.4
2D rotated	177.6 ± 5.3	117.4 ± 4.4	198.8 ± 11.4	60.9 ± 1.8	64.6 ± 3.3*
Pend. cont.	25.0 ± 0.0	25.0 ± 0.0	143.2 ± 13.5	9.0 ± 0.0	59.2 ± 9.7
Pend. disc.	13.9 ± 0.8	14.0 ± 0.8	534.8 ± 15.8	2.0 ± 0.0	57.0 ± 10.9*
Cartpole	71.3 ± 4.4	77.0 ± 3.4	793.2 ± 36.1	42.6 ± 2.2	181.0 ± 9.5*
Flight	51.7 ± 5.2	51.6 ± 5.7	371.6 ± 14.5	70.9 ± 6.1	298.4 ± 22.8

†Shows results for the algorithm with the best predictions between SECRET with axis-orthogonal splits and SECRET with oblique splits (*).

Table 9. Summary of incremental algorithm model sizes (number of tree leaves/receptive fields) for the dynamic domains.

	Online-RD		Online-RA		
	Unpruned	Pruned	Unpruned	Pruned	RFWR
2D test	58.0 ± 4.4	58.4 ± 5.2	85.8 ± 10.9	87.0 ± 10.1	92.2 ± 4.3
2D rotated	186.8 ± 5.9	185.7 ± 5.8	234.5 ± 7.5	232.4 ± 8.9	100.0 ± 2.7
Pend. cont.	26.3 ± 1.6	26.2 ± 1.6	26.4 ± 1.1	26.4 ± 1.1	19.2 ± 3.3
Pend. disc.	15.8 ± 1.1	15.7 ± 1.1	17.0 ± 0.0	17.0 ± 0.0	15.9 ± 4.0
Cartpole	233.0 ± 127.4	107.8 ± 46.6	127.9 ± 58.8	119.2 ± 35.6	140.4 ± 6.0
Flight	110.9 ± 21.8	55.0 ± 14.7	41.2 ± 23.9	35.4 ± 19.4	26.2 ± 4.9

Table 10. Summary of batch algorithm gradient errors (nRMSE(Grad) %) for the dynamic domains.

	Batch-RD	Batch-RA	GUIDE	SECRET†
2D test	29.4 ± 1.7	33.8 ± 1.6	41.8 ± 1.9	47.6 ± 1.4
2D rotated	33.4 ± 1.4	39.6 ± 1.2	67.9 ± 1.6	50.4 ± 7.6*
Pend. cont.	3.6 ± 0.1	3.7 ± 0.0	14.5 ± 0.2	6.7 ± 5.6

†Shows results for the algorithm with the best predictions between SECRET with axis-orthogonal splits and SECRET with oblique splits (*).

Table 11. Summary of incremental algorithm gradient errors (nRMSE(Grad) %) for the dynamic domains.

	Online-RD		Online-RA			
	Unpruned	Pruned	Unpruned	Pruned	RFWR	Linear
2D test	30.2 ± 1.8	30.5 ± 1.9	31.1 ± 2.7	30.5 ± 2.1	28.3 ± 2.5	100.0 ± 0.0
2D rotated	29.8 ± 1.0	29.6 ± 0.9	29.9 ± 1.8	30.3 ± 1.6	25.5 ± 1.1	100.0 ± 0.0
Pend. cont.	3.6 ± 0.2	3.6 ± 0.2	3.6 ± 0.1	3.6 ± 0.1	4.8 ± 0.8	76.1 ± 0.4

Table 12. Summary of batch algorithm training times (ms per example) for the dynamic domains.

	Batch-RD	Batch-RA	M5′	GUIDE	GUIDE*	SECRET†
2D test	0.29 ± 0.00	0.04 ± 0.00	21.6 ± 0.3	20.3 ± 0.5	0.48 ± 0.00	0.14 ± 0.00
2D rotated	0.35 ± 0.01	0.05 ± 0.00	25.9 ± 0.3	32.0 ± 0.3	0.48 ± 0.01	0.15 ± 0.00*
Pend. cont.	0.44 ± 0.01	0.05 ± 0.00	28.3 ± 0.2	19.9 ± 0.1	0.21 ± 0.01	0.33 ± 0.00
Pend. disc.	0.39 ± 0.01	0.04 ± 0.00	29.2 ± 0.2	0.05 ± 0.01	0.05 ± 0.01	0.33 ± 0.00*
Cartpole	1.8 ± 0.1	0.09 ± 0.00	67.7 ± 0.6	73.8 ± 4.5	1.3 ± 0.0	0.91 ± 0.02*
Flight	6.9 ± 0.1	0.24 ± 0.01	90.7 ± 0.6	31.7 ± 2.4	5.7 ± 0.2	2.8 ± 0.2

†Shows results for the algorithm with the best predictions between SECRET with axis-orthogonal splits and SECRET with oblique splits (*).

Table 13. Summary of incremental algorithm training times (ms per example) for the dynamic domains.

	Online-RD		Online-RA			
	Unpruned	Pruned	Unpruned	Pruned	RFWR	Linear
2D test	0.19 ± 0.00	0.19 ± 0.00	0.08 ± 0.00	0.10 ± 0.01	0.63 ± 0.03	0.01 ± 0.00
2D rotated	0.23 ± 0.00	0.23 ± 0.00	0.10 ± 0.00	0.12 ± 0.00	0.68 ± 0.02	0.01 ± 0.00
Pend. cont.	0.59 ± 0.01	0.60 ± 0.01	0.10 ± 0.00	0.11 ± 0.00	0.31 ± 0.04	0.01 ± 0.00
Pend. disc.	0.53 ± 0.02	0.53 ± 0.02	0.09 ± 0.00	0.09 ± 0.00	0.41 ± 0.08	0.01 ± 0.00
Cartpole	2.7 ± 0.1	2.7 ± 0.1	0.30 ± 0.05	0.31 ± 0.04	3.6 ± 0.1	0.02 ± 0.00
Flight	41.2 ± 2.2	39.8 ± 1.1	1.1 ± 0.2	1.1 ± 0.1	5.0 ± 0.5	0.06 ± 0.00

selection based on sample quantiles. The training times for this faster method are shown as GUIDE*.

The main points to note from these results are:

– Batch-RD builds an accurate model for all domains and while not as fast as SECRET, is as fast as GUIDE* and faster than GUIDE and M5′. Batch-RA is much faster than all other algorithms and although it fails to find a very good model for the cartpole, it is very effective in the other domains. Again it is thought that the reason for the competitiveness of Batch-RD/RA is the large sizes of the datasets which allow the statistical stopping tests to be as effective as post-pruning.
– For an equal number of training examples Online-RD/RA are not quite as effective as their batch equivalents. However the difference is often small and the incremental algorithms are able to scale up to many more examples. Comparison with RFWR shows the incremental algorithms to be very competitive. Online-RD is more computationally demanding, however it forms a more accurate approximation from fewer examples than either RFWR or Online-RA. As observed previously Online-RA is less effective when presented with a shifting input distribution, and this explains its poor performance at the cartpole and flight simulator.

- Batch/Online-RD/RA all have the luxury of a parameter to control the trade-off between tree size and model accuracy. Although parameters are undesirable, it is seen that M5′, SECRET and GUIDE are unable to form a reasonable trade-off in all domains. M5′ always constructs unnecessarily large trees and SECRET can also fail to recognise when a tree is too big. On the other hand GUIDE often builds trees that are too small. The Batch/Online-RD/RA stopping parameter is related to the prediction error, and therefore allows the algorithms to adapt well to the rotated test function by building larger trees.
- Both RFWR and SECRET can represent the rotated test function more efficiently than the other algorithms. This is because the receptive fields in RFWR are free to extend in any direction and SECRET is using oblique splits.
- As discussed in Section 6 it is the function gradient that relates to the dynamic model. GUIDE and SECRET are poor at predicting gradients because they do not use smoothing. RFWR forms its predictions from a sum of Gaussians, and is therefore well suited to representing the 2D test function and its gradient. The smoothing technique of Batch/Online-RD/RA is seen to be very effective and gives the best gradient predictions for the continuous pendulum.
- Online pruning has little effect in the smaller domains, but for the cartpole and flight simulator significantly reduces the tree size with little or no impact on prediction accuracy.

8. Discussion

This article introduces two new incremental learners, Online-RD and Online-RA, that can form prediction models from a stream of data. The prior knowledge required is minimal as the stopping parameter δ_0 can simply be set to zero (as in Section 7.1). However the alternative incremental learner RFWR requires more initial knowledge. The range of each regressor and predictor variable y are needed to normalise the data, and three learning parameters must be defined; the initial distance metric $\mathbf{D}_0$, the penalty γ and the learning rates. In fact there are two learning rates, one for standard gradient descent and one for meta-learning, but these are taken to be the same. The initial distance metric affects how many linear models are allocated initially, the penalty affects the final number of local models, and the learning rates affect how quickly the size and shape of the local models change. In practice it proved hard to balance the effects of these parameters and obtain good learning performance. If the learning rates are too high the algorithm becomes unstable, and if too low no perceptible change in the size and shape of the local models is observed.

Assuming that the number of local models in their piecewise linear approximations has stabilised, both Online-RD/RA and RFWR scale linearly with the number of examples and their memory usage is capped. This enables them to operate on an unlimited stream of data. When processing a training example Online-RD updates $\kappa(d-1)$ more models than Online-RA, and is therefore significantly slower. As expected the difference between the RD and RA splitting rules becomes more pronounced in higher dimensional domains. There is clearly a trade-off between the more complex RD rule that learns from fewer examples and the simpler RA rule that requires more training data to form an accurate approximation. If processing power is not a concern or the input distribution is known to be non-stationary then the RD rule is preferred, otherwise the RA rule may be more suitable.

When representing dynamic models the model tree algorithms induce a separate tree for each state component. For example for the continuous pendulum (17), one model tree is built for $\dot{\theta}$ and one for $\ddot{\theta}$. With regressors θ, $\dot{\theta}$ and u, clearly $\dot{\theta}$ can be represented by a single linear model, but $\ddot{\theta}$ contains a sine wave component in the θ direction. A single linear model is therefore induced for $\dot{\theta}$, and a piecewise linear approximation constructed from multiple linear models for $\ddot{\theta}$. On the other hand the local RFWR models predict all components simultaneously and are not able to efficiently represent this function. It is possible to build a separate RFWR model for each component (in the same way that separate model trees are induced for each component), however this does not improve predictions and uses a lot more computation. A significant reduction in the number of models can only be achieved by setting different RFWR parameters for the two components, but this would be providing the algorithm with additional prior knowledge. The ability to efficiently approximate functions containing large variations in curvature may be one reason why model trees learn from fewer examples in these domains.

9. Conclusions and future work

This article describes and evaluates two versions of an algorithm that induces linear model trees in both batch and online settings. Batch-RD is seen to be more effective than M5′, GUIDE and SECRET over a number of larger standard datasets and some non-standard system identification tasks. Batch-RA has worse prediction power and takes more examples to learn, but is computationally a lot less demanding. The incremental versions are competitive with the alternative online learner RFWR, and require less prior knowledge and fewer parameters to be tuned. Indeed Online-RD consistently learns a more accurate approximation from fewer examples than RFWR. Online-RA may require more examples initially to form a good approximation and is unable to adapt to a shifting input distribution, although its final approximations are often good and it is very fast.

The trade-off between the asymptotic tree size and the final approximation error can easily be controlled using the single stopping parameter δ_0, which in the absence of any prior knowledge can simply be set to zero. In addition there are no learning rates to be tuned, thus avoiding a major cause of instability in many gradient descent systems. Moreover no initial knowledge regarding the size of the input domain is required. Categorical attributes can be easily incorporated into the trees when using the RD splitting rule. Incremental pruning has little impact in the smaller domains, but significantly reduces the tree sizes in the larger domains, with only a small effect on prediction error.

Having developed a method for rapidly learning non-linear models of dynamic environments, future work will concentrate on applications in control. Nakanishi, Farrell, and Schaal (2004) have developed a provably stable adaptive controller based on the representation learnt by RFWR, and perhaps a similar approach can be applied to incrementally induced linear model trees. The interplay of control and system identification may also be mutually beneficial. Poorly modelled regions of the state-space can be deliberately explored, and the full complexities of the dynamic model can be exposed by stimulating the system enough to satisfy the system identification persistence of excitation conditions (Ljung, 1987). A more accurate model will then enable better control within these regions.

Appendix A

This appendix derives the residual difference (RD) statistical test used in both splitting and pruning when inducing a tree T incrementally. Label the internal nodes of a particular sub-tree of T rooted at I1 as I1 ... Ip, and the leaves as L1 ... Lq (see Figure A.1).

The sub-tree was grown incrementally, and initially only consisted of the sub-tree root node I1. Therefore there are a number of examples $N_{0,\mathrm{I}1}$ that were observed at I1 but not at any lower node. Similarly at each internal node j there are $N_{0,\mathrm{I}j}$ examples that were observed before the node had any children. A linear model is constructed at each internal node j from the $N_{0,\mathrm{I}j}$ examples observed before splitting, and the corresponding residual sums of squares $\mathrm{RSS}_{0,\mathrm{I}j}$ are calculated. In each leaf i a linear model is formed from the $N_{\mathrm{L}i}$ examples that reached the leaf, and the residual sums of squares $\mathrm{RSS}_{\mathrm{L}i}$ are also calculated.

If the sum of $N_{\mathrm{L}i}$ over all leaves is denoted as N_L and the sum of $N_{0,\mathrm{I}j}$ over all internal nodes is denoted $N_{0,\mathrm{I}}$ then the sub-tree has observed a total of $N = N_\mathrm{L} + N_{0,\mathrm{I}}$ examples. A single linear model is constructed at the root I1 from all N examples and the corresponding residual sum of squares RSS is calculated. Also denote the sum of $\mathrm{RSS}_{\mathrm{L}i}$ over all leaves as RSS_L and the sum of $\mathrm{RSS}_{0,\mathrm{I}j}$ over all internal nodes as $\mathrm{RSS}_{0,\mathrm{I}}$.

Making the assumptions that the observation noise is independent, zero-mean and Gaussian with variance σ^2, and that the regressor matrix in each regression defined above has full rank d, then we can form the null hypothesis H_0 that all the observed examples were generated by a single linear model. The alternative hypothesis is that the data is better explained by the linear models in the leaves of the tree.

Using standard analysis of covariance techniques generalised to multiple regressions (Kullback & Rosenblatt, 1957) it can be shown that the three expressions on the right-hand side of the identity

$$\mathrm{RSS} \equiv \mathrm{RSS}_\mathrm{L} + \mathrm{RSS}_{0,\mathrm{I}} + (\mathrm{RSS} - \mathrm{RSS}_\mathrm{L} - \mathrm{RSS}_{0,\mathrm{I}})$$

are distributed independently as $\chi^2(N_\mathrm{L} - qd)\sigma^2$, $\chi^2(N_{0,\mathrm{I}} - pd)\sigma^2$ and $\chi^2((p+q-1)d)\sigma^2$. To obtain a better comparison between the null and alternative hypotheses, the effect of the internal nodes is removed. Adding the last two expressions above gives $(\mathrm{RSS} - \mathrm{RSS}_\mathrm{L})$ which is distributed as $\chi^2(N_{0,\mathrm{I}} + (q-1)d)\sigma^2$ by the summation of two independent χ^2-distributed variables. This distribution is clearly affected if H_0 does not hold, whereas RSS_L has the same distribution regardless. Following Chow (1960) H_0 can therefore be tested by

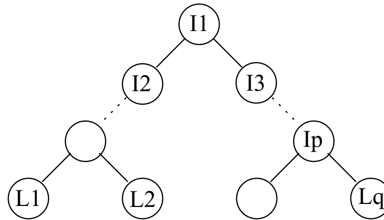


Figure A.1. Labelling of tree nodes.

the statistic

$$F_{\mathrm{RD}} = \frac{(\mathrm{RSS} - \mathrm{RSS_L}) \times (N_\mathrm{L} - qd)}{\mathrm{RSS_L} \times (N_{0,\mathrm{I}} + (q-1)d)} \tag{A.1}$$

which is distributed according to Fisher's $\mathcal{F}$ distribution with $N_{0,\mathrm{I}}+(q-1)d$ and $N_\mathrm{L}-qd$ degrees of freedom by the definition of the $\mathcal{F}$ distribution as the ratio of two independent χ^2-distributed variables.

Appendix B

This appendix gives the prediction errors for each algorithm on each Weka dataset, as discussed in Section 7.1.

Table B.1. Batch algorithm prediction errors (nRMSE%) for the Weka datasets.

Dataset	Batch-RD	Batch-RA	M5′	GUIDE	SECRET†
2dplanes	22.7 ± 0.2	+54.3 ± 0.5	−22.7 ± 0.2	−22.7 ± 0.2	+53.7 ± 4.2*
abalone	65.3 ± 3.9	+67.2 ± 4.1	+65.8 ± 4.1	+66.4 ± 4.2	+66.8 ± 5.1*
ailerons	40.0 ± 1.2	+41.6 ± 1.2	−39.7 ± 1.1	40.0 ± 1.1	–
bank32nh	67.0 ± 2.6	−66.2 ± 2.7	+67.3 ± 2.6	−66.6 ± 2.7	+67.3 ± 2.7*
bank8FM	19.2 ± 0.6	+20.9 ± 0.7	+20.0 ± 0.7	+19.8 ± 0.6	+20.0 ± 0.7*
cal_housing	47.0 ± 1.4	+51.5 ± 1.5	+48.4 ± 2.1	−45.9 ± 1.4	+54.2 ± 5.3*
cpu_act	12.6 ± 0.6	+12.9 ± 0.5	+14.8 ± 1.7	+13.0 ± 1.7	+51.8 ± 6.9*
cpu_small	15.3 ± 0.6	+16.4 ± 0.5	+17.3 ± 2.0	+15.9 ± 0.6	+27.1 ± 14.9*
delta_ailerons	53.5 ± 2.2	+54.4 ± 2.4	+54.4 ± 2.3	+54.2 ± 2.3	+55.6 ± 2.5*
delta_elevators	59.8 ± 1.7	+60.2 ± 1.7	+60.0 ± 1.7	+59.9 ± 1.7	+60.8 ± 1.7
elevators	31.1 ± 1.0	+35.4 ± 1.3	+32.2 ± 1.1	+33.1 ± 1.1	–
fried	20.6 ± 0.2	+21.5 ± 0.4	+27.8 ± 0.5	+20.7 ± 0.2	+21.4 ± 0.5
house_16H	68.6 ± 3.7	+71.6 ± 3.8	−67.7 ± 4.2	−66.7 ± 3.7	+85.3 ± 5.2*
house_8L	59.8 ± 3.2	+62.5 ± 3.1	59.8 ± 5.0	+60.2 ± 3.3	+77.7 ± 5.1*
kin8nm	46.6 ± 1.4	+69.0 ± 2.6	+60.9 ± 2.0	+52.8 ± 2.1	+47.5 ± 2.5*
mv	0.2 ± 0.0	+5.2 ± 1.5	+1.4 ± 0.6	+0.8 ± 0.1	+13.4 ± 15.0*
pol	15.6 ± 1.2	+26.0 ± 1.7	−15.1 ± 1.3	+20.1 ± 1.6	–
puma32H	21.8 ± 0.5	+84.6 ± 15.0	+27.1 ± 0.6	+27.0 ± 5.4	+83.0 ± 12.4
puma8NH	57.0 ± 1.2	+57.5 ± 1.5	−56.9 ± 1.3	+57.2 ± 1.2	+57.7 ± 1.4
Average error	38.1%	46.3%	40.0%	39.1%	52.7%
Tree size	27.8	9.5	35.9	11.6	8.3
Build time (s)	26.7	0.8	60.8	62.6	3.0

† Shows the best result between SECRET with axis-orthogonal splits and SECRET with oblique splits (*). SECRET fails for the datasets ailerons, elevators and pol.

Table B.2. Incremental algorithm prediction errors (nRMSE%) for the Weka datasets.

	Online-RD		Online-RA		
Dataset	Unpruned	Pruned	Unpruned	Pruned	Linear
2dplanes	22.7 ± 0.3	22.7 ± 0.3	54.3 ± 0.5	-54.3 ± 0.5	54.3 ± 0.5
abalone	66.8 ± 4.3	$+67.8 \pm 4.8$	69.3 ± 5.8	$+70.0 \pm 6.2$	69.3 ± 4.6
ailerons	40.4 ± 1.1	$+41.6 \pm 1.9$	41.6 ± 1.3	-41.2 ± 1.4	42.9 ± 1.4
bank32nh	67.5 ± 2.8	67.3 ± 2.6	68.2 ± 2.7	68.1 ± 2.6	68.5 ± 2.5
bank8FM	19.9 ± 0.7	$+20.7 \pm 0.8$	21.4 ± 0.8	21.3 ± 0.7	25.5 ± 1.3
cal_housing	52.8 ± 2.3	$+59.9 \pm 6.2$	61.9 ± 9.3	$+76.5 \pm 13.8$	60.3 ± 1.6
cpu_act	26.4 ± 7.1	-21.8 ± 7.7	15.2 ± 3.2	-14.6 ± 1.8	52.4 ± 4.1
cpu_small	18.2 ± 3.9	-17.4 ± 3.0	17.2 ± 0.8	17.1 ± 0.7	53.5 ± 4.0
delta_ailerons	55.2 ± 2.4	$+56.7 \pm 2.3$	55.9 ± 2.7	56.1 ± 2.8	56.8 ± 2.3
delta_elevators	60.6 ± 1.8	$+60.7 \pm 1.7$	60.1 ± 1.8	$+60.9 \pm 2.0$	61.0 ± 1.7
elevators	33.6 ± 1.3	$+34.7 \pm 1.6$	35.2 ± 1.4	$+37.1 \pm 3.2$	43.2 ± 2.1
fried	20.9 ± 0.2	20.9 ± 0.2	22.0 ± 0.5	$+22.2 \pm 0.5$	52.6 ± 0.6
house_16H	72.0 ± 3.7	$+81.0 \pm 5.6$	75.2 ± 4.1	$+75.7 \pm 3.9$	86.1 ± 4.1
house_8L	63.3 ± 3.5	$+72.6 \pm 5.7$	63.4 ± 3.3	$+64.0 \pm 3.5$	78.7 ± 3.6
kin8nm	59.5 ± 4.8	$+61.6 \pm 5.2$	72.5 ± 1.8	72.6 ± 1.8	76.6 ± 1.8
mv	1.5 ± 1.2	$+1.8 \pm 1.8$	7.9 ± 2.2	7.6 ± 1.9	53.7 ± 0.8
pol	25.0 ± 3.2	$+38.6 \pm 14.6$	33.4 ± 5.5	33.9 ± 6.2	73.1 ± 0.8
puma32H	24.6 ± 0.9	24.6 ± 0.9	82.3 ± 15.7	82.7 ± 15.4	88.5 ± 2.2
puma8NH	57.8 ± 1.2	57.9 ± 1.2	58.5 ± 1.5	58.5 ± 1.5	79.4 ± 1.6
Average error	41.5%	43.7%	48.2%	49.2%	61.9%
Tree size	16.5	7.4	7.1	6.1	–
Build time (s)	64.0	61.9	3.0	2.9	0.2

Appendix C

This appendix gives the detailed results for each algorithm in the dynamic domains, as discussed in Section 7.4.

C.1. Parameter values

Table C.1. Stopping parameter values for the batch and incremental model tree learners.

Domain	Batch-RD	Batch-RA	Online-RD	Online-RA
2D test	3×10^{-3}	5×10^{-3}	5×10^{-3}	10^{-2}
2D rotated	3×10^{-3}	5×10^{-3}	5×10^{-3}	10^{-2}
Pend. cont.	2×10^{-5}	5×10^{-4}	2×10^{-5}	2×10^{-4}
Pend. disc.	5×10^{-6}	5×10^{-5}	5×10^{-6}	5×10^{-5}
Cartpole	5×10^{-4}	10^{-4}	10^{-3}	10^{-3}
Flight	2×10^{-3}	5×10^{-3}	10^{-2}	5×10^{-3}

Table C.2. RFWR parameter values.

Domain	Initial distance metric $\mathbf{D}_0$	Penalty γ	Learning rates
2D test	25**I** (**I** is the identity matrix)	10^{-7}	250
2D rotated	25**I**	10^{-7}	250
Pend. cont.	5**I**	10^{-9}	1000
Pend. disc.	5**I**	10^{-9}	5000
Cartpole	10**I**	10^{-7}	1000
Flight	2.5**I**	10^{-5}	1000

C.2. 2D test function detailed results

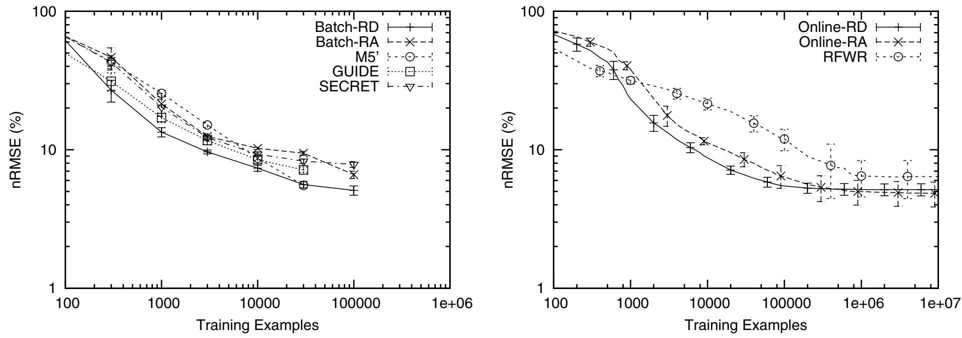


Figure C.1. Prediction errors on the 2D test function.

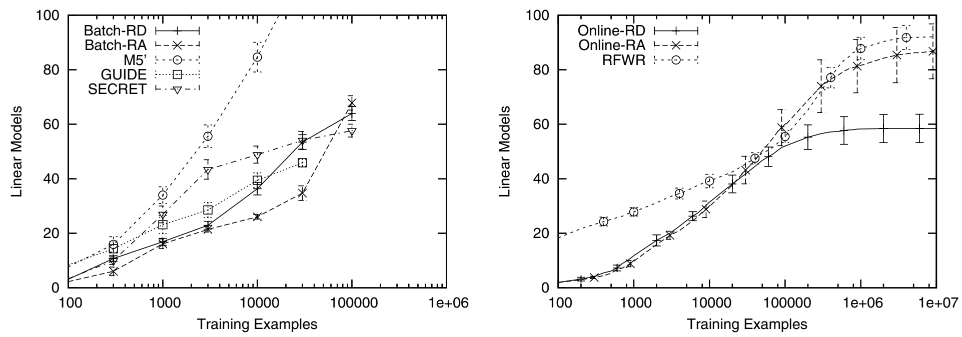


Figure C.2. Number of leaves/receptive fields on the 2D test function.

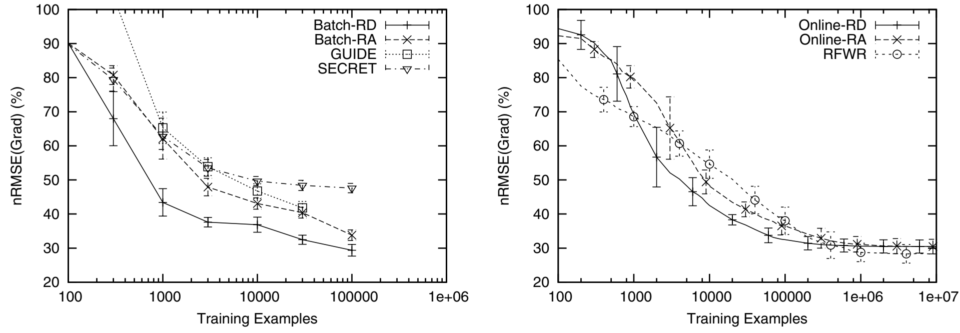


Figure C.3. Gradient errors on the 2D test function.

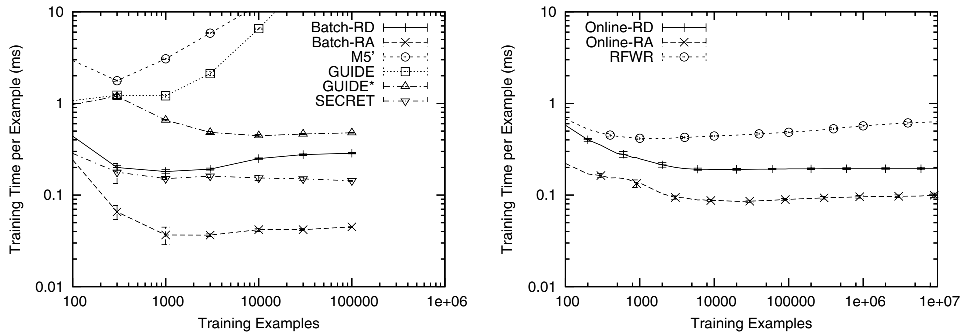


Figure C.4. Performance on the 2D test function.

C.3. Rotated 2D test function detailed results

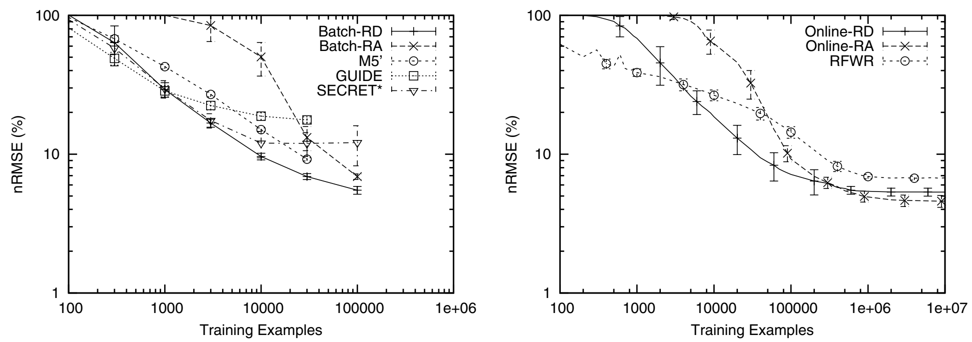


Figure C.5. Prediction errors on the rotated 2D test function.

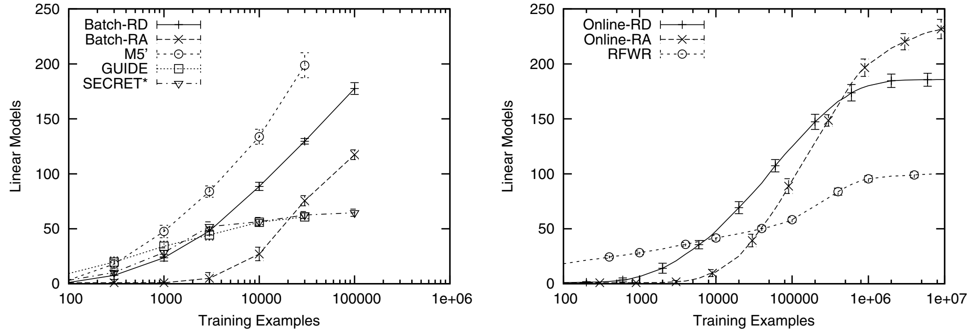


Figure C.6. Number of leaves/receptive fields on the rotated 2D test function.

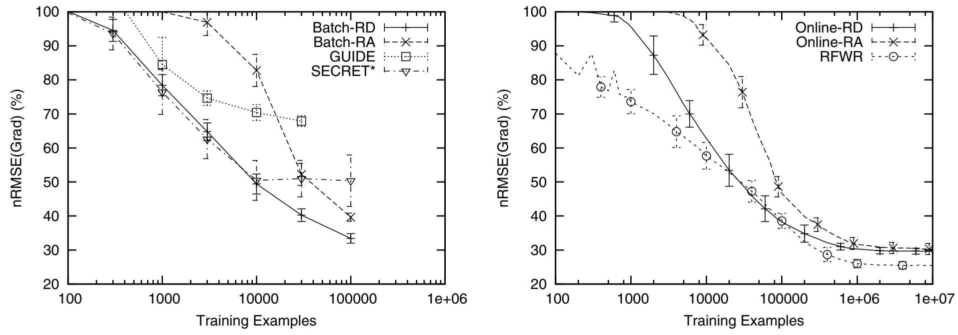


Figure C.7. Gradient errors on the rotated 2D test function.

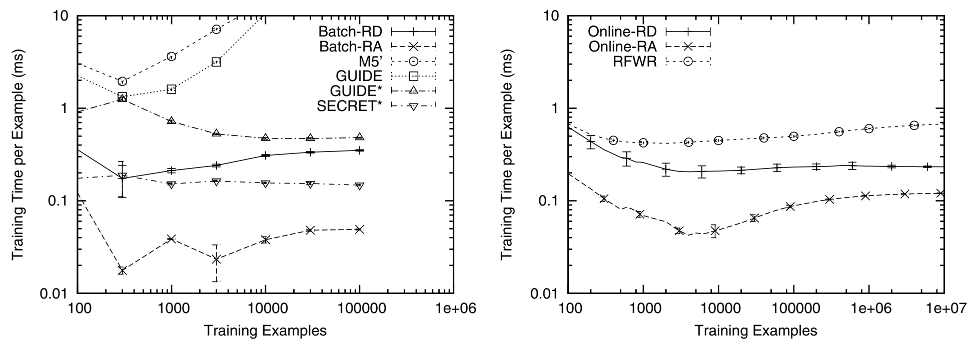


Figure C.8. Performance on the rotated 2D test function.

C.4. Continuous pendulum detailed results

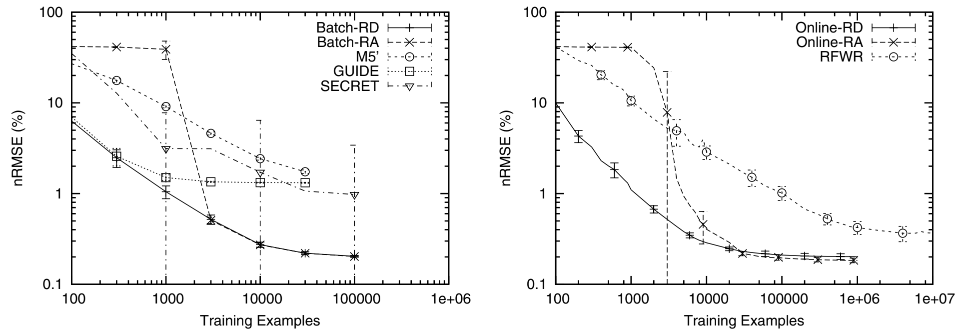


Figure C.9. Prediction errors on the continuous pendulum.

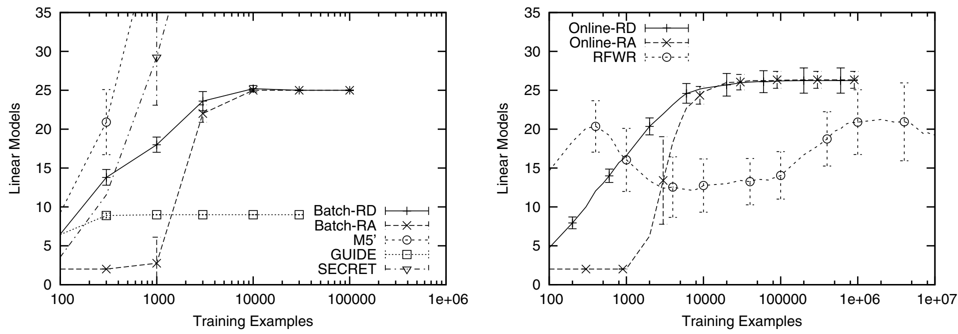


Figure C.10. Number of leaves/receptive fields on the continuous pendulum.

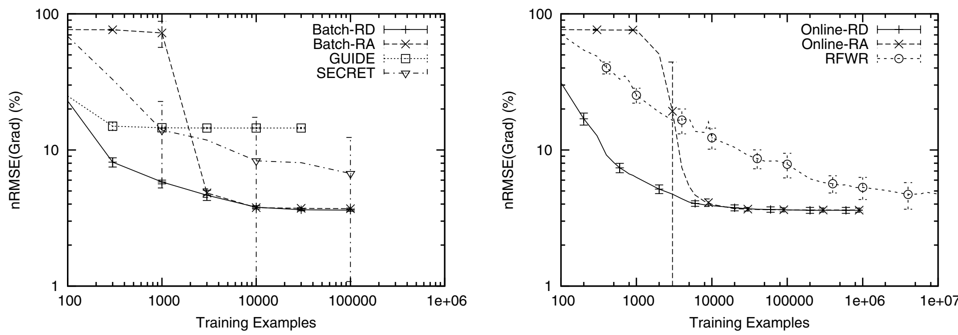


Figure C.11. Gradient errors on the continuous pendulum.

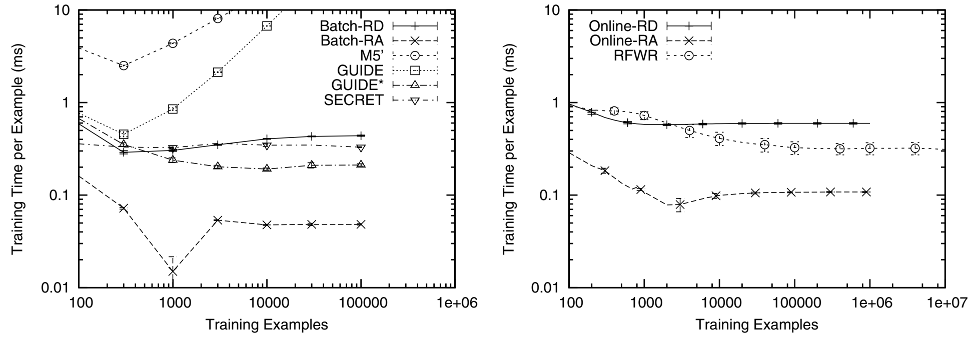


Figure C.12. Performance on the continuous pendulum.

C.5. Discrete pendulum detailed results

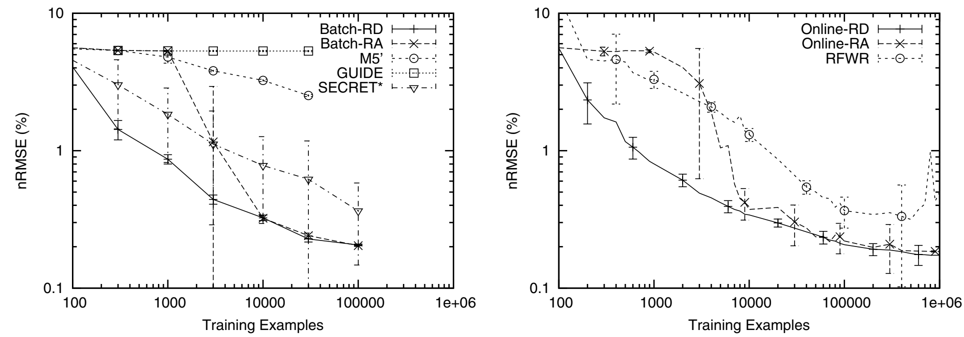


Figure C.13. Prediction errors on the discrete pendulum.

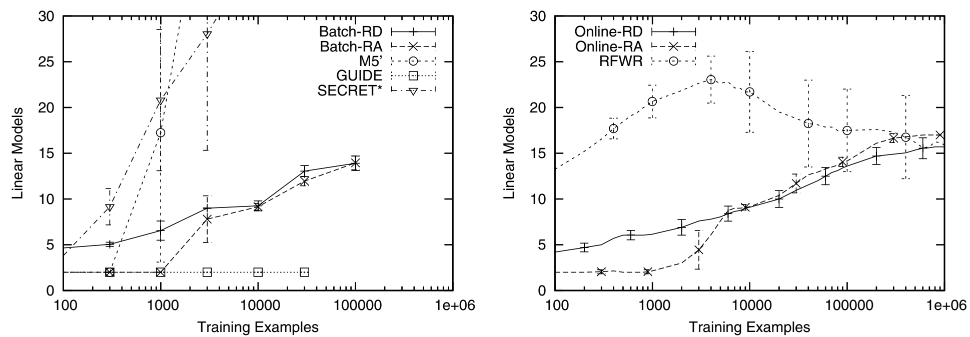


Figure C.14. Number of leaves/receptive fields on the discrete pendulum.

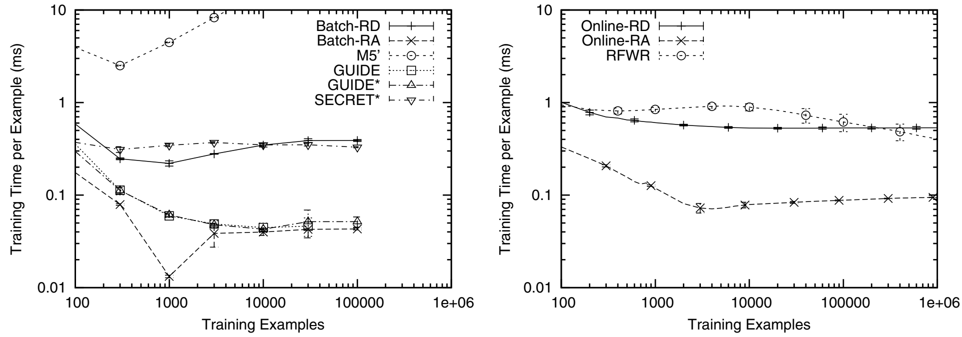


Figure C.15. Performance on the discrete pendulum.

C.6. Cart and pole detailed results

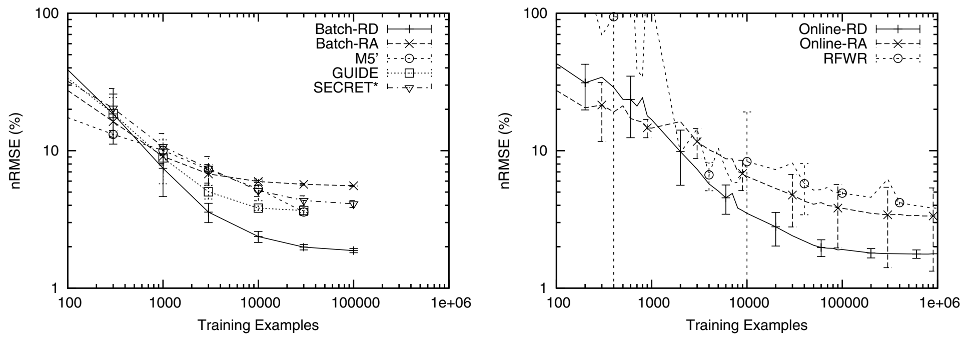


Figure C.16. Prediction errors on the cart and pole.

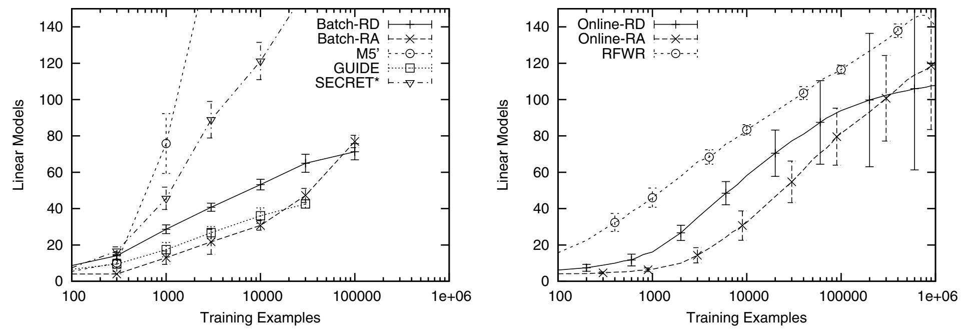


Figure C.17. Number of leaves/receptive fields on the cart and pole.

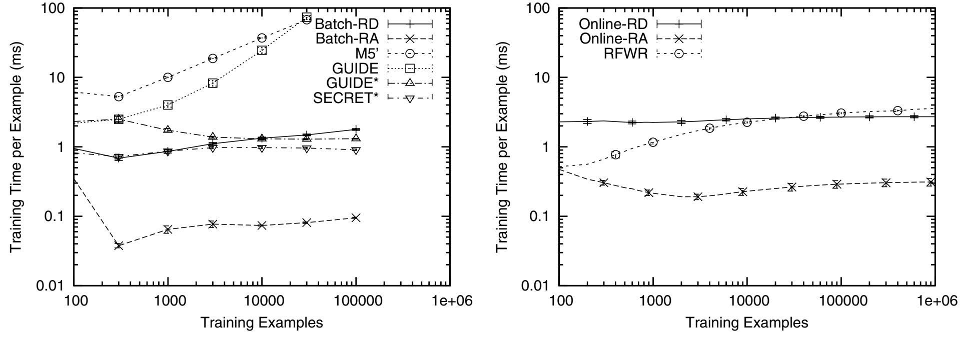


Figure C.18. Performance on the cart and pole.

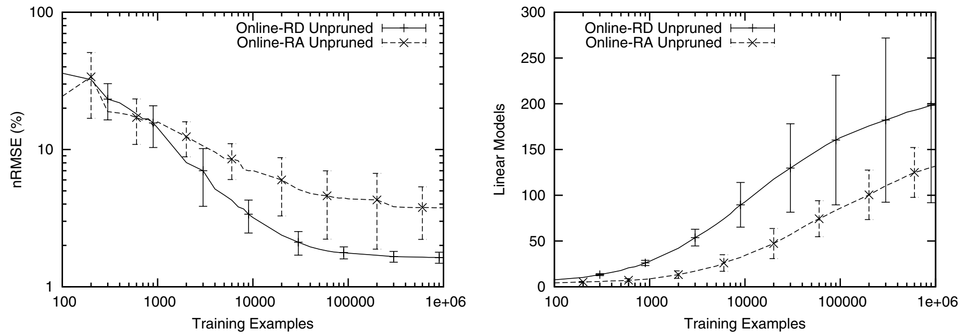


Figure C.19. Unpruned model trees on the cart and pole.

C.7. Flight simulator detailed results

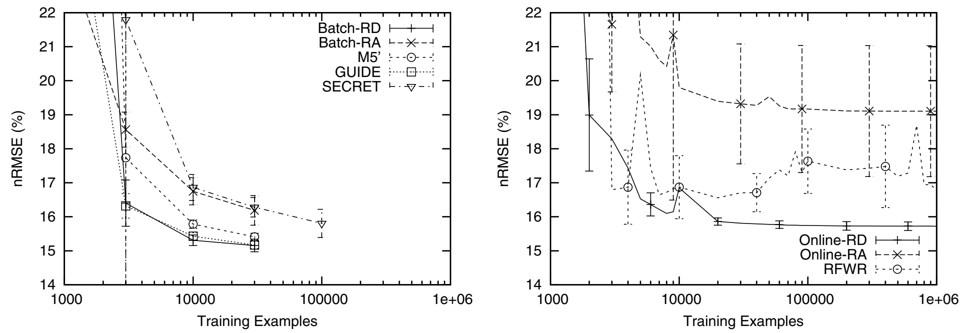


Figure C.20. Prediction errors on the flight simulator.

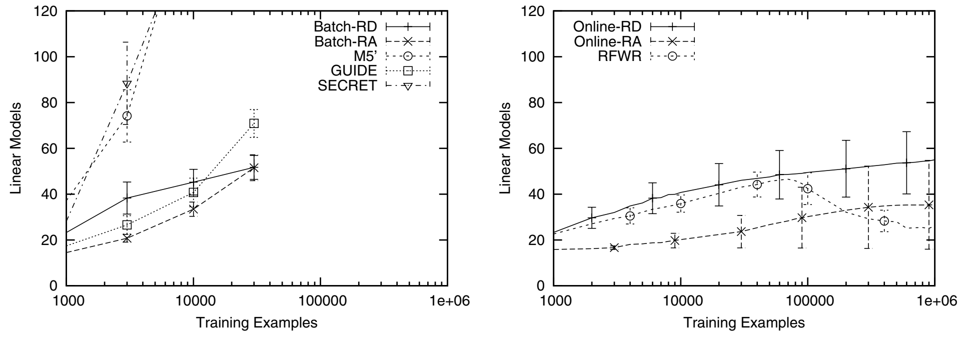


Figure C.21. Number of leaves/receptive fields on the flight simulator.

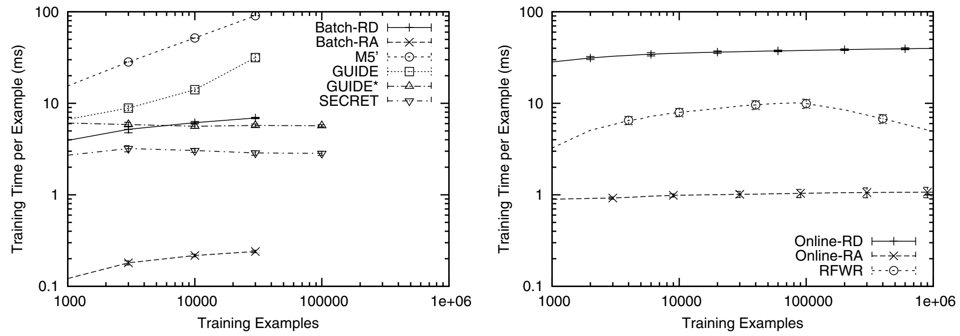


Figure C.22. Performance on the flight simulator.

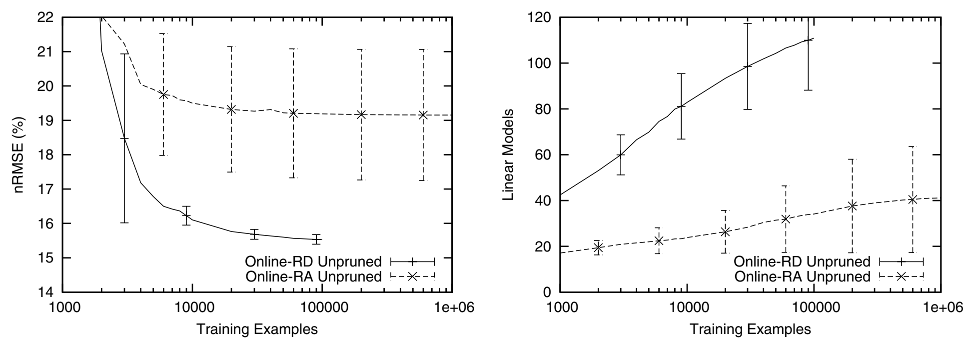


Figure C.23. Unpruned model trees on the flight simulator.

Acknowledgments

The authors would like to thank the anonymous referees for their detailed and helpful comments, and also acknowledge the financial support of the CSE Postgraduate Research Scholarship.

Notes

1. In some cases it is possible to initialise some of the new sub-models using the parent's sub-models, but this does not affect the discussion.
2. See http://www.cs.waikato.ac.nz/∼ml/

References

Alexander, W., & Grimshaw, S. (1996). Treed regression. *Journal of Computational and Graphical Statistics, 5*, 156–175.

Atkeson, C., Moore, A., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review, 11*, 11–73.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Wadsworth.

Cestnik, B., & Bratko, I. (1991). On estimating probabilities in tree pruning. In Y. Kodratoff (Ed.), *Proceedings of the European Working Session on Learning*, vol. 482 of *Lecture Notes in Artificial Intelligence* (pp. 138–150). Springer.

Chaudhuri, P., Huang, M., Loh, W., & Yao, R. (1994). Piecewise-polynomial regression trees. *Statistica Sinica, 4*, 143–167.

Chow, G. (1960). Tests of equality between sets of coefficients in two linear regressions. *Econometrica, 28*:3, 591–605.

Dobra, A., & Gehrke, J. (2002). SECRET: A scalable linear regression tree algorithm. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 481–487). ACM.

Frank, E., Wang, Y., Inglis, S., Holmes, G., & Witten, I. (1998). Using model trees for classification. *Machine Learning, 32*:1, 63–76.

Gama, J. (2004). Functional trees. *Machine Learning, 55*:3, 219–250.

Gama, J., Rocha, R., & Medas, P. (2003). Accurate decision trees for mining high-speed data streams. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 523–528). ACM.

Hastie, T., & Loader, C. (1993). Local regression: Automatic kernel carpentry. *Statistical Science, 8*:2, 120–143.

Haykin, S. (2002). *Adaptive Filter Theory*. Prentice-Hall.

Isaac, A., & Sammut, C. (2003). Goal-directed learning to fly. In T. Fawcett & N. Mishra (Eds.), *Proceedings of the 20th International Conference of Machine Learning* (pp. 258–265). AAAI Press.

Karalič, A. (1992). Employing linear regression in regression tree leaves. In B. Neumann (Ed.), *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 440–441). Wiley.

Kullback, S., & Rosenblatt, H. (1957). On the analysis of multiple regression in *k* categories. *Biometrika, 44*, 67–83.

Last, M. (2002). Online classification of non-stationary data streams. *Intelligent Data Analysis, 6*, 129–147.

Li, K., Lue, H., & Chen, C. (2000). Interactive tree-structured regression via principal Hessian directions. *Journal of the American Statistical Association, 95*, 547–560.

Ljung, L. (1987). *System Identification: Theory for the User*. Prentice-Hall.

Loh, W. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica, 12*, 361–386.

Malerba, D., Esposito, F., Ceci, M., & Appice, A. (2004). Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*, 612–625.

Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In P. Apers, M. Bouzeghoub, & G. Gardarin (Eds.), *Proceedings of the 5th International Conference on Extending Database Technology*, vol. 1057 of *Lecture Notes in Computer Science* (pp. 18–32). Springer.

Moore, A., & Atkeson, C. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning, 21*, 199–233.

Munos, R., & Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning, 49*, 291–323.

Murray-Smith, R. (1994). A local model network approach to nonlinear modelling. Ph.D. thesis, University of Strathclyde, Strathclyde, UK.

Nakanishi, J., Farrell, J., & Schaal, S. (2004). Learning composite adaptive control for a class of nonlinear systems. In *IEEE International Conference on Robotics and Automation* (pp. 2647–2652).

Nelles, O. (2001) *Nonlinear System Identification*. Springer.

Potts, D. (2004a). Fast incremental learning of linear model trees. In J. Carbonell & J. Siekmann (Eds.), *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence*, vol. 3157 of *Lecture Notes in Artificial Intelligence* (pp. 221–230). Springer.

Potts, D. (2004b). Incremental learning of linear model trees. In R. Greiner & D. Schuurmans (Eds.), *Proceedings of the 21st International Conference on Machine Learning* (pp. 663–670). ACM.

Quinlan, J. (1993a). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Quinlan, J. (1993b). Combining instance-based and model-based learning. In *Proceedings of the 10th International Conference on Machine Learning* (pp. 236–243). Morgan Kaufmann.

Robnik-Šikonja, M., & Kononenko, I. (1998). Pruning regression trees with MDL. In H. Prade (Ed.), *Proceedings of the 13th European Conference on Artificial Intelligence* (pp. 455–459). Wiley.

Schaal, S., & Atkeson, C. (1998). Constructive incremental learning from only local information. *Neural Computation, 10*, 2047–2084.

Schlimmer, J., & Fisher, D. (1986). A case study of incremental concept induction. In *Proceedings of the 5th National Conference on Artificial Intelligence* (pp. 496–501). AAAI Press.

Sicilano, R., & Mola, F. (1994). Modelling for recursive partitioning and variable selection. In R. Dutter & W. Grossmann (Eds.), *Proceedings in Computational Statistics: COMPSTAT '94* (pp. 172–177). Physica Verlag.

Slotine, J., & Li, W. (1991). *Applied nonlinear control*. Prentice-Hall.

Šuc, D., Vladušič, D., & Bratko, I. (2004). Qualitatively faithful quantitative prediction. *Artificial Intelligence, 158*:2, 189–214.

Torgo, L. (1997). Functional models for regression tree leaves. In D. Fisher (Ed.), *Proceedings of the 14th International Conference on Machine Learning* (pp. 385–393). Morgan Kaufmann.

Torgo, L. (2002). Computationally efficient linear regression trees. In K. Jajuga, A. Sokolowski, & H.-H. Bock (Eds.), *Classification, Clustering and Data Analysis: Recent Advances and Applications*. Springer.

Utgoff, P., Berkman, N., & Clouse, J. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning, 29*, 5–44.

Vijayakumar, S., & Schaal, S. (2000). Locally weighted projection regression: Incremental real time learning in high dimensional space. In P. Langley (Ed.), *Proceedings of the 17th International Conference on Machine Learning* (pp. 1079–1086). Morgan Kaufmann.

Wang, Y., & Witten, I. (1997). Inducing model trees for continuous classes. In *Proceedings of Poster Papers, 9th European Conference on Machine Learning*.

Received June 14, 2004
Revised March 4, 2005
Accepted March 7, 2005